Structure of the Course

Part I. Introduction to Algorithms (Chapter 2)  
- algorithms: exhaustive search, greedy, dynamic programming  
- problems: motif finding, sequence alignment, gene finding

Part II. Fundamental Techniques (Chapters 4 - 6)  
- algorithms: graph-, string-, and tree-algorithms  
- problems: DNA sequencing, pattern finding, phylogeny

Part III Advanced Algorithms (Chapters 7 - 10)  
- algorithms: HMM, stochastic grammars, Markov networks  
- problems: decoding, learning, inference algorithms
Structure of the Course

- Part I. Introduction to Algorithms (Chapter 2)
  - algorithms: exhaustive search, greedy, dynamic programming
  - problems: motif finding, sequence alignment, gene finding

- Part II. Fundamental Techniques (Chapters 4 - 6)
  - algorithms: graph-, string-, and tree-algorithms
  - problems: DNA sequencing, pattern finding, phylogeny

- Part III Advanced Algorithms (Chapters 7 - 10)
  - algorithms: HMM, stochastic grammars, Markov networks
  - problems: decoding, learning, inference algorithms

- Part IV Probabilistic Methods (Chapters 11 - 12)
Structure of the Course

- Part I. Introduction to Algorithms (Chapter 2)
- Part II. Fundamental Techniques (Chapters 4 - 6)
Structure of the Course

- Part I. Introduction to Algorithms (Chapter 2)
- Part II. Fundamental Techniques (Chapters 4 - 6)
  - algorithms: exhaustive search, greedy, dynamic programming
- Part III. Advanced Algorithms (Chapters 7 - 10)
  - algorithms: graph-, string-, and tree-algorithms
  - problems: DNA sequencing, pattern finding, phylogeny
- Part IV. Probabilistic Methods (Chapters 11 - 12)
  - algorithms: HMM, stochastic grammars, Markov networks
  - problems: decoding, learning, inference algorithms
Structure of the Course

▶ Part I. Introduction to Algorithms (Chapter 2)
▶ Part II. Fundamental Techniques (Chapters 4 - 6)
  algorithms: exhaustive search, greedy, dynamic programming
  problems: motif finding, sequence alignment, gene finding
▶ Part III Advanced Algorithms (Chapters 7 - 10)
  algorithms: graph-, string-, and tree-algorithms
  problems: DNA sequencing, pattern finding, phylogeny
▶ Part IV Probabilistic Methods (Chapters 11 - 12)
  algorithms: HMM, stochastic grammars, Markov networks
  problems: decoding, learning, inference algorithms
Structure of the Course

- Part I. Introduction to Algorithms (Chapter 2)
- Part II. Fundamental Techniques (Chapters 4 - 6)
  - algorithms: exhaustive search, greedy, dynamic programming
  - problems: motif finding, sequence alignment, gene finding
- Part III Advanced Algorithms (Chapters 7 - 10)
Structure of the Course

- Part I. Introduction to Algorithms (Chapter 2)
- Part II. Fundamental Techniques (Chapters 4 - 6)
  - algorithms: exhaustive search, greedy, dynamic programming
  - problems: motif finding, sequence alignment, gene finding
- Part III Advanced Algorithms (Chapters 7 - 10)
  - algorithms: graph-, string-, and tree-algorithms
Structure of the Course

- Part I. Introduction to Algorithms (Chapter 2)
- Part II. Fundamental Techniques (Chapters 4 - 6)
  - Algorithms: exhaustive search, greedy, dynamic programming
  - Problems: motif finding, sequence alignment, gene finding
- Part III Advanced Algorithms (Chapters 7 - 10)
  - Algorithms: graph-, string-, and tree-algorithms
  - Problems: DNA sequencing, pattern finding, phylogeny
Structure of the Course

- Part I. Introduction to Algorithms (Chapter 2)
- Part II. Fundamental Techniques (Chapters 4 - 6)
  - algorithms: exhaustive search, greedy, dynamic programming
  - problems: motif finding, sequence alignment, gene finding
- Part III Advanced Algorithms (Chapters 7 - 10)
  - algorithms: graph-, string-, and tree-algorithms
  - problems: DNA sequencing, pattern finding, phylogeny
- Part IV Probabilistic Methods (Chapters 11 - 12)
Structure of the Course

- Part I. Introduction to Algorithms (Chapter 2)

- Part II. Fundamental Techniques (Chapters 4 - 6)
  algorithms: exhaustive search, greedy, dynamic programming
  problems: motif finding, sequence alignment, gene finding

- Part III Advanced Algorithms (Chapters 7 - 10)
  algorithms: graph-, string-, and tree-algorithms
  problems: DNA sequencing, pattern finding, phylogeny

- Part IV Probabilistic Methods (Chapters 11 - 12)
  algorithms: HMM, stochastic grammars, Markov networks
Structure of the Course

- **Part I. Introduction to Algorithms (Chapter 2)**
- **Part II. Fundamental Techniques (Chapters 4 - 6)**
  - algorithms: exhaustive search, greedy, dynamic programming
  - problems: motif finding, sequence alignment, gene finding
- **Part III Advanced Algorithms (Chapters 7 - 10)**
  - algorithms: graph-, string-, and tree-algorithms
  - problems: DNA sequencing, pattern finding, phylogeny
- **Part IV Probabilistic Methods (Chapters 11 - 12)**
  - algorithms: HMM, stochastic grammars, Markov networks
  - problems: decoding, learning, inference algorithms
Part I. Introduction to Algorithms

Chapter 2. Algorithms and Complexity

Algorithm: a well-defined procedure that takes an input and produces an output.

Example: Algorithm $\text{MAX}$;

Input: List $x = \{a_1, \cdots, a_n\}$;

Body (a finite series of instructions);

Output: $y$, the maximum of $a_1, \cdots, a_n$.

An algorithm: a finite process to compute a function or a relation.
Part I. Introduction to Algorithms

Chapter 2. Algorithms and Complexity
Chapter 2. Algorithms and Complexity

*Algorithm*: a well-defined procedure that takes an input and produces an output.

\[
Input(x) \Rightarrow Body \Rightarrow Output(y)
\]
Chapter 2. Algorithms and Complexity

*Algorithm*: a well-defined procedure that takes an input and produces an output.

\[ \text{Input}(x) \Rightarrow \text{Body} \Rightarrow \text{Output}(y) \]

Example: Algorithm \( MAX \);

*Input*: List \( x = \{a_1, \cdots, a_n\} \);

*Body* (a finite series of instructions);

*Output*: \( y \), the maximum of \( a_1, \cdots, a_n \).
Chapter 2. Algorithms and Complexity

*Algorithm*: a well-defined procedure that takes an input and produces an output.

\[
Input(x) \Rightarrow Body \Rightarrow Output(y)
\]

Example: **Algorithm** $MAX$;

*Input*: List $x = \{a_1, \ldots, a_n\}$;

*Body* (a finite series of instructions);

*Output*: $y$, the maximum of $a_1, \ldots, a_n$.

An algorithm: a *finite process* to compute a function or a relation.
Chapter 2. Algorithms and Complexity

Notation conventions for algorithm writing (pseudo-code)
Chapter 2. Algorithms and Complexity

Notation conventions for algorithm writing (pseudo-code)

Memory: *variables, arrays, arguments, parameters*
Chapter 2. Algorithms and Complexity

**Notation conventions for algorithm writing (pseudo-code)**

Memory: *variables, arrays, arguments, parameters*

Array Access: $a_i$: the $i$th location of array $a$
Chapter 2. Algorithms and Complexity

Notation conventions for algorithm writing (pseudo-code)

Memory: variables, arrays, arguments, parameters
Array Access: $a_i$: the $i$th location of array $a$
Assignment: $a \leftarrow b$
Chapter 2. Algorithms and Complexity

Notation conventions for algorithm writing (pseudo-code)

Memory: variables, arrays, arguments, parameters
Array Access: $a_i$: the $i$th location of array $a$
Assignment: $a \leftarrow b$
Arithmetic: $a + b$, $a - b$, $a \times b$, $a/b$, $a^b$
Notation conventions for algorithm writing (pseudo-code)

Memory: variables, arrays, arguments, parameters
Array Access: $a_i$: the $i$th location of array $a$
Assignment: $a \leftarrow b$
Arithmetic: $a + b, a - b, a \times b, a/b, a^b$
Conditional: if condition is true

```
  body 1
```
else

```
  body 2
```
Chapter 2. Algorithms and Complexity

Notation conventions for algorithm writing (pseudo-code)

Memory: \textit{variables, arrays, arguments, parameters}

Array Access: \( a_i \): the \( i \)th location of array \( a \)

Assignment: \( a \leftarrow b \)

Arithmetic: \( a + b, a - b, a \times b, a/b, a^b \)

Conditional: \texttt{if condition} is true

\hspace{1cm} \texttt{body 1}

\hspace{1cm} \texttt{else}

\hspace{2cm} \texttt{body 2}

For Loop: \texttt{for } i \leftarrow \textit{low} \texttt{ to } \textit{high}

\hspace{1cm} \texttt{body}
Chapter 2. Algorithms and Complexity

Notation conventions for algorithm writing (pseudo-code)

Memory: variables, arrays, arguments, parameters

Array Access: \( a_i \): the \( i \)th location of array \( a \)

Assignment: \( a \leftarrow b \)

Arithmetic: \( a + b, a - b, a \times b, a/b, a^b \)

Conditional: \textbf{if} condition is true

\hspace{1cm} body 1

\hspace{1cm} else

\hspace{2cm} body 2

For Loop: \textbf{for} \( i \leftarrow \text{low} \) to \text{high}

\hspace{1cm} body

While Loop: \textbf{while} condition is true

\hspace{1cm} body
Chapter 2. Algorithms and Complexity

*Example:* an iterative algorithm computing the $n$th number in the Fibonacci series 1, 1, 2, 3, 5, 8, 13, 21, ....

```
Fibonacci(n)
1. F1 ← 1
2. F2 ← 1
3. for i ← 3 to n
4.     Fi ← Fi−1 + Fi−2
5. return (Fn)
```
Example: an iterative algorithm computing the $n$th number in the Fibonacci series 1, 1, 2, 3, 5, 8, 13, 21, ....

Fibonacci ($n$)
1. $F_1 \leftarrow 1$
2. $F_2 \leftarrow 1$
3. for $i \leftarrow 3$ to $n$
4. $F_i \leftarrow F_{i-1} + F_{i-2}$
5. return ($F_n$)
Example: an iterative algorithm computing the $n$th number in the Fibonacci series 1, 1, 2, 3, 5, 8, 13, 21, ....

Fibonacci ($n$)
1. $F_1 \leftarrow 1$
2. $F_2 \leftarrow 1$
3. for $i \leftarrow 3$ to $n$
4. $F_i \leftarrow F_{i-1} + F_{i-2}$
5. return ($F_n$)

How is the algorithm executed?
Chapter 2. Algorithms and Complexity

Recursive algorithms

Rec-Fibonacci \( n \)

1. if \( n \) = 1 OR \( n \) = 2, return (1)
2. else
3. \( T_1 = \) Rec-Fibonacci \( (n - 1) \);
4. \( T_2 = \) Rec-Fibonacci \( (n - 2) \);
5. return \( (T_1 + T_2) \);
Chapter 2. Algorithms and Complexity

Recursive algorithms

Rec-Fibonacci($n$)
1. if $n = 1$ OR $n = 2$, return (1)
2. else
3. $T_1 = \text{Rec-Fibonacci}(n - 1)$;
4. $T_2 = \text{Rec-Fibonacci}(n - 2)$;
5. return ($T_1 + T_2$);
Recursive algorithms

\texttt{Rec-Fibonacci}(n)
1. \textbf{if} $n = 1 \ \text{OR} \ n = 2$, \textbf{return} (1)
2. \textbf{else}
3. \hspace{1em} $T_1 = \texttt{Rec-Fibonacci}(n - 1)$;
4. \hspace{1em} $T_2 = \texttt{Rec-Fibonacci}(n - 2)$;
5. \hspace{1em} \textbf{return} ($T_1 + T_2$);

How is the algorithm executed?
Chapter 2. Algorithms and Complexity

```
RECFIBONACCI(n)
if n = 1 or n = 2, return (1);
else
  T1 = RECFIBONACCI(n - 1);
  T2 = RECFIBONACCI(n - 2);
  return (T1 + T2);
```

```
RECFIBONACCI(n)
if n = 1 or n = 2, return (1);
else
  T1 = RECFIBONACCI(n - 1);
  T2 = RECFIBONACCI(n - 2);
  return (T1 + T2);
```
Chapter 2. Algorithms and Complexity
Chapter 2. Algorithms and Complexity

![Diagram of Fibonacci sequence]

The Fibonacci sequence is defined by the recurrence relation:

- \( F(0) = 0 \)
- \( F(1) = 1 \)
- \( F(n) = F(n-1) + F(n-2) \) for \( n > 1 \)
Chapter 2. Algorithms and Complexity

Advantages with recursive algorithms
Chapter 2. Algorithms and Complexity

Advantages with recursive algorithms

Example: recursive StringCopy
Advantages with recursive algorithms

Example: recursive StringCopy

Key ingredients for admitting recursive algorithms

More examples
Advantages with recursive algorithms

Example: recursive StringCopy

Key ingredients for admitting recursive algorithms

More examples
  Sum(n), linear search,
Advantages with recursive algorithms

Example: \textit{recursive StringCopy}

Key ingredients for admitting recursive algorithms

More examples
  \texttt{Sum(n)}, linear search,
  summation over numbers in a 'triangle', etc.
A more complex example: Towers of Hanoi
A more complex example: **Towers of Hanoi**

TowersOfHanoi(First, Second, Third, n)
A more complex example: **Towers of Hanoi**

TowersOfHanoi(First, Second, Third, n)

1. **if** \( n = 1 \), MoveOne(First, Third)
Chapter 2. Algorithms and Complexity

A more complex example: **Towers of Hanoi**

TowersOfHanoi(First, Second, Third, n)
1. **if** $n = 1$, MoveOne(First, Third)
2. **else**
A more complex example: **Towers of Hanoi**

TowersOfHanoi(First, Second, Third, n)
1. **if** \( n = 1 \), MoveOne(First, Third)
2. **else**
   TowersOfHanoi(First, Third, Second, n-1)
A more complex example: **Towers of Hanoi**

TowersOfHanoi(First, Second, Third, n)
1. *if* $n = 1$, MoveOne(First, Third)
2. *else*
   TowersOfHanoi(First, Third, Second, n-1)
   MoveOne(First, Third)
Chapter 2. Algorithms and Complexity

A more complex example: **Towers of Hanoi**

TowersOfHanoi(First, Second, Third, n)
1. if \( n = 1 \), MoveOne(First, Third)
2. else
   TowersOfHanoi(First, Third, Second, n-1)
   MoveOne(First, Third)
   TowersOfHanoi(Second, First, Third, n-1)
Chapter 2. Algorithms and Complexity

Disadvantages with recursive algorithms

May be *inefficient*!
Chapter 2. Algorithms and Complexity

Disadvantages with recursive algorithms

May be *inefficient*

(1) overhead: use of stacks: push/pop
Disadvantages with recursive algorithms

May be *inefficient*!

(1) overhead: use of stacks: push/pop
(2) possible re-computation:

   e.g., RecFib(5) may be computed several times
Algorithm Efficiency: how is it defined?
Chapter 2. Algorithms and Complexity

**Algorithm Efficiency**: how is it defined?

Counting the number of basic operations used
Algorithm Efficiency: how is it defined?

Counting the number of basic operations used

Example

\textbf{SelectionSort}(a, n)
1 \hspace{1em} \textbf{for} \hspace{0.5em} i \leftarrow 1 \hspace{0.5em} \textbf{to} \hspace{0.5em} n - 1
2. \hspace{1em} j \leftarrow i \hspace{1em} \{ \text{starting of inner loop, assume } a_i \text{ to be the smallest elements in } a_i, \ldots, a_n \}
3. \hspace{1em} \textbf{for} \hspace{0.5em} k \leftarrow i + 1 \hspace{0.5em} \textbf{to} \hspace{0.5em} n \hspace{1em} \{ \text{search for the smallest element} \}
4. \hspace{1em} \textbf{if} \hspace{0.5em} a_k < a_j
5. \hspace{1em} j \leftarrow k
6. \hspace{1em} \text{Swap } a_i \text{ and } a_j
7 \hspace{1em} \textbf{return} \hspace{0.5em} \text{array } a
Algorithm Efficiency (cont’)

SelectionSort(a, n)
1 for i ← 1 to n - 1
2. j ← i
3. for k ← i + 1 to n
4. if ak < aj
5. j ← k
6. Swap ai and aj
7 return array a
Algorithm Efficiency (cont’)

SelectionSort\((a, n)\)
1. \textbf{for} \(i \leftarrow 1 \textbf{ to } n - 1\)
2. \(j \leftarrow i\)
3. \textbf{for} \(k \leftarrow i + 1 \textbf{ to } n\)
4. \textbf{if} \(a_k < a_j\)
5. \(j \leftarrow k\)
6. Swap \(a_i\) and \(a_j\)
7. \textbf{return} array \(a\)

Count the total number of basic operations needed:
Algorithm Efficiency (cont’)

SelectionSort$(a, n)$
1. for $i \leftarrow 1$ to $n - 1$
2. $j \leftarrow i$
3. for $k \leftarrow i + 1$ to $n$
4. if $a_k < a_j$
5. $j \leftarrow k$
6. Swap $a_i$ and $a_j$
7. return array $a$

Count the total number of basic operations needed:

\[
= c_1 \times n + c_2 \times (n - 1) + c_3 \times \sum_{i=1}^{n-1} (n - i) + c_4,5 \times \sum_{i=1}^{n-1} (n - i - 1) + c_6 \times (n - 1) + c_7
\]

\[
= an^2 + b^n + c \quad \text{for some constant } a > 0, b, c
\]
Chapter 2. Algorithms and Complexity

How to count basic operations in recursive algorithms?
Chapter 2. Algorithms and Complexity

How to count basic operations in recursive algorithms?

RecFib(n)
1. if $n = 1$ OR $n = 2$ return (1)
2. else
   return (RecFib($n - 1$) + RecFib($n - 2$))
How to count basic operations in recursive algorithms?

RecFib($n$)
1. if $n = 1$ OR $n = 2$ return (1)
2. else
   return (RecFib($n - 1$) + RecFib($n - 2$))

Deriving and solving recurrences!
Chapter 2. Algorithms and Complexity

How to count basic operations in recursive algorithms?

RecFib($n$)
1. if $n = 1$ OR $n = 2$ return (1)
2. else
   return (RecFib($n - 1$) + RecFib($n - 2$))

Deriving and solving recurrences!

- Let $t(n)$ be the time needed for computing RecFib($n$)
Chapter 2. Algorithms and Complexity

How to count basic operations in recursive algorithms?

RecFib\(n\)
1. if \(n = 1\) OR \(n = 2\) return \((1)\)
2. else
   return \((\text{RecFib}(n - 1) + \text{RecFib}(n - 2))\)

Deriving and solving recurrences!

- Let \(t(n)\) be the time needed for computing RecFib\(n\)
- then
  \[
  t(n) = c + t(n - 1) + t(n - 2)
  \]
Chapter 2. Algorithms and Complexity

How to count basic operations in recursive algorithms?

RecFib(n)
1. if \(n = 1\) OR \(n = 2\) return (1)
2. else
   return (RecFib(n − 1) + RecFib(n − 2))

Deriving and solving recurrences!

- Let \(t(n)\) be the time needed for computing RecFib(n)
- then
  \[ t(n) = c + t(n − 1) + t(n − 2) \]

  \[ t(n) = 1, \quad \text{when } n = 1, 2 \]
Chapter 2. Algorithms and Complexity

How to count basic operations in recursive algorithms?

RecFib(n)
1. if $n = 1$ OR $n = 2$ return (1)
2. else
   return (RecFib($n - 1$) + RecFib($n - 2$))

Deriving and solving recurrences!

- Let $t(n)$ be the time needed for computing RecFib($n$)
- then

$$t(n) = c + t(n - 1) + t(n - 2)$$

$$t(n) = 1, \text{ when } n = 1, 2$$

- solve it exactly,
Chapter 2. Algorithms and Complexity

How to count basic operations in recursive algorithms?

RecFib(n)
1. if \( n = 1 \) OR \( n = 2 \) return (1)
2. else
   return (RecFib(\( n - 1 \)) + RecFib(\( n - 2 \)))

Deriving and solving recurrences!

- Let \( t(n) \) be the time needed for computing RecFib(n)
- then
\[
t(n) = c + t(n - 1) + t(n - 2)
\]

\[
t(n) = 1, \quad \text{when } n = 1, 2
\]

- solve it exactly, or
Chapter 2. Algorithms and Complexity

How to count basic operations in recursive algorithms?

RecFib(n)
1. if $n = 1$ OR $n = 2$ return (1)
2. else
   return (RecFib($n - 1$) + RecFib($n - 2$))

Deriving and solving recurrences!

- Let $t(n)$ be the time needed for computing RecFib($n$)
- then
  \[ t(n) = c + t(n - 1) + t(n - 2) \]

  \[ t(n) = 1, \text{ when } n = 1, 2 \]

- solve it exactly, or

- estimate lower and upper bounds.
Chapter 2. Algorithms and Complexity

\[ t(n) = c + t(n - 1) + t(n - 2) \]
Chapter 2. Algorithms and Complexity

\[ t(n) = c + t(n - 1) + t(n - 2) \]

where \( t(n) = 1 \), when \( n = 1, 2 \)
Chapter 2. Algorithms and Complexity

\[ t(n) = c + t(n - 1) + t(n - 2) \]

where \( t(n) = 1 \), when \( n = 1, 2 \)

- Estimate the lower and upper bounds of \( t(n) \),

\[ t(n) \geq \sqrt{2} n - 0.414 \]
\[ t(n) \leq 2n \]

So

\[ 0.414 n < t(n) < 2n \]

Solve it exactly: assume \( t(n) = \alpha n \), we have

\[ \alpha n = c + \alpha (n - 1) + \alpha (n - 2) \]
\[ \alpha^2 - \alpha - 1 = 0 \]
\[ \alpha = \frac{1 + \sqrt{5}}{2} \approx 1.618 \]

So \( t(n) \approx 1.618 n \)
Chapter 2. Algorithms and Complexity

\[ t(n) = c + t(n - 1) + t(n - 2) \]

where \( t(n) = 1 \), when \( n = 1, 2 \)

- Estimate the lower and upper bounds of \( t(n) \), based on the recursive tree structure:
Chapter 2. Algorithms and Complexity

\[ t(n) = c + t(n-1) + t(n-2) \]

where \( t(n) = 1 \), when \( n = 1, 2 \)

- Estimate the lower and upper bounds of \( t(n) \), based on the recursive tree structure:
  
  lower bound: \( t(n) \geq 2^{\frac{n}{2}} = \sqrt{2}^n > 1.414^n \)
Chapter 2. Algorithms and Complexity

\[ t(n) = c + t(n - 1) + t(n - 2) \]

where \( t(n) = 1 \), when \( n = 1, 2 \)

- Estimate the lower and upper bounds of \( t(n) \), based on the recursive tree structure:

  lower bound: \( t(n) \geq 2^{\frac{n}{2}} = \sqrt{2^n} > 1.414^n \)
  upper bound: \( t(n) \leq 2^n \)
Chapter 2. Algorithms and Complexity

\[ t(n) = c + t(n - 1) + t(n - 2) \]

where \( t(n) = 1 \), when \( n = 1, 2 \)

- Estimate the lower and upper bounds of \( t(n) \), based on the recursive tree structure:

  lower bound: \( t(n) \geq 2^{\frac{n}{2}} = \sqrt{2^n} > 1.414^n \)
  upper bound: \( t(n) \leq 2^n \)

So \( 1.414^n < t(n) < 2^n \)
Chapter 2. Algorithms and Complexity

\[ t(n) = c + t(n-1) + t(n-2) \]

where \( t(n) = 1 \), when \( n = 1, 2 \)

- Estimate the lower and upper bounds of \( t(n) \), based on the recursive tree structure:

  lower bound: \( t(n) \geq 2^{\frac{n}{2}} = \sqrt{2^n} > 1.414^n \)
  upper bound: \( t(n) \leq 2^n \)
  So \( 1.414^n < t(n) < 2^n \)

- Solve it exactly: assume \( t(n) = \alpha^n \), we have
Chapter 2. Algorithms and Complexity

\[ t(n) = c + t(n-1) + t(n-2) \]

where \( t(n) = 1 \), when \( n = 1, 2 \)

- Estimate the lower and upper bounds of \( t(n) \), based on the recursive tree structure:

  lower bound: \( t(n) \geq 2^{\frac{n}{2}} = \sqrt{2}^n > 1.414^n \)
  upper bound: \( t(n) \leq 2^n \)

  So \( 1.414^n < t(n) < 2^n \)

- Solve it exactly: assume \( t(n) = \alpha^n \), we have

  \[ \alpha^n = c + \alpha^{n-1} + \alpha^{n-2} \]
Chapter 2. Algorithms and Complexity

\[ t(n) = c + t(n - 1) + t(n - 2) \]

where \( t(n) = 1 \), when \( n = 1, 2 \)

- Estimate the lower and upper bounds of \( t(n) \), based on the recursive tree structure:

  lower bound: \( t(n) \geq 2^{\frac{n}{2}} = \sqrt{2}^n > 1.414^n \)
  upper bound: \( t(n) \leq 2^n \)
  So \( 1.414^n < t(n) < 2^n \)

- Solve it exactly: assume \( t(n) = \alpha^n \), we have

\[
\alpha^n = c + \alpha^{n-1} + \alpha^{n-2} \\
\alpha^2 = \alpha + 1
\]
\[ t(n) = c + t(n - 1) + t(n - 2) \]

where \( t(n) = 1 \), when \( n = 1, 2 \)

- Estimate the lower and upper bounds of \( t(n) \), based on the recursive tree structure:
  
  lower bound: \( t(n) \geq 2^{\frac{n}{2}} = \sqrt{2}^n > 1.414^n \)
  
  upper bound: \( t(n) \leq 2^n \)
  
  So \( 1.414^n < t(n) < 2^n \)

- Solve it exactly: assume \( t(n) = \alpha^n \), we have
  
  \[ \alpha^n = c + \alpha^{n-1} + \alpha^{n-2} \]
  
  \[ \alpha^2 = \alpha + 1 \quad \text{i.e.,} \quad \alpha^2 - \alpha - 1 = 0 \]
Chapter 2. Algorithms and Complexity

\[ t(n) = c + t(n - 1) + t(n - 2) \]

where \( t(n) = 1 \), when \( n = 1, 2 \)

- Estimate the lower and upper bounds of \( t(n) \), based on the recursive tree structure:

  \[
  \text{lower bound: } t(n) \geq 2 \frac{n}{2} = \sqrt{2^n} > 1.414^n \\
  \text{upper bound: } t(n) \leq 2^n
  \]

  So \( 1.414^n < t(n) < 2^n \)

- Solve it exactly: assume \( t(n) = \alpha^n \), we have

  \[
  \alpha^n = c + \alpha^{n-1} + \alpha^{n-2} \\
  \alpha^2 = \alpha + 1 \quad \text{i.e., } \alpha^2 - \alpha - 1 = 0 \\
  \alpha = \frac{(1 + \sqrt{5})}{2} \approx 1.618
  \]
Chapter 2. Algorithms and Complexity

\[ t(n) = c + t(n - 1) + t(n - 2) \]

where \( t(n) = 1 \), when \( n = 1, 2 \)

- Estimate the lower and upper bounds of \( t(n) \), based on the recursive tree structure:

  lower bound: \( t(n) \geq 2^{\frac{n}{2}} = \sqrt{2}^n > 1.414^n \)
  upper bound: \( t(n) \leq 2^n \)
  So \( 1.414^n < t(n) < 2^n \)

- Solve it exactly: assume \( t(n) = \alpha^n \), we have

\[
\alpha^n = c + \alpha^{n-1} + \alpha^{n-2}
\]
\[
\alpha^2 = \alpha + 1 \quad \text{i.e.,} \quad \alpha^2 - \alpha - 1 = 0
\]
\[
\alpha = \frac{(1 + \sqrt{5})}{2} \approx 1.618
\]
So \( t(n) \approx 1.618^n \)
Analysis of recursive algorithms (cont')

Another example: Search a sorted list (of indexes $i, \ldots, j$) for a key

\begin{verbatim}
BinarySearch(L, i, j, key)
1. if $i > j$ return (0)  
   { base case, list is empty, not found }
2. else
3. let $m \leftarrow \lceil i + j \rceil$  
   { get mid point index }
4. if $key = L_m$ return ($m$)  
   { found }
5. else
6. if $key < L_m$ then
7.   BinarySearch(L, i, $m - 1$, key)  
   { search the left half list }
8. else
9.   BinarySearch(L, $m + 1$, j, key)  
   { search the right half list }
\end{verbatim}

Can you write an iterative algorithm for binary search?
Analysis of recursive algorithms (cont’)

Another example: Search a sorted list (of indexes \(i, \ldots, j\)) for a key

\[
\text{BinarySearch} (L, i, j, key) \\
1. \text{if } i > j \ \text{return } (0) \quad \{\text{base case, list is empty, not found}\} \\
2. \text{else} \\
3. \quad \text{let } m \leftarrow \left\lceil \frac{i+j}{2} \right\rceil \quad \{\text{get mid point index}\} \\
4. \quad \text{if } key = L_m \ \text{return}(m) \quad \{\text{found}\} \\
5. \quad \text{else} \\
6. \quad \quad \text{if } key < L_m \ \text{BinarySearch} (L, i, m - 1, key) \quad \{\text{search the left half list}\} \\
7. \quad \quad \text{else} \quad \{key > L_m\} \\
8. \quad \quad \text{BinarySearch} (L, m + 1, j, key) \quad \{\text{search the right half list}\}
\]
Analysis of recursive algorithms (cont’)

Another example: Search a sorted list (of indexes $i, \ldots, j$) for a key

\textbf{BinarySearch ($L, i, j, key$)}
1. if $i > j$ return (0) \{base case, list is empty, not found\}
2. else
3. let $m \leftarrow \lceil \frac{i+j}{2} \rceil$ \{get mid point index \}
4. if $key = L_m$ return($m$) \{ found \}
5. else
6. if $key < L_m$ BinarySearch ($L, i, m - 1, key$) \{ search the left half list \}
7. else \{ $key > L_m$ \}
8. BinarySearch ($L, m + 1, j, key$) \{ search the right half list \}

Can you write an iterative algorithm for binary search?
Chapter 2. Algorithms and Complexity

Assume $t(n)$ to be total time to for $\text{BinarySearch} (L, i, j, key)$ where $n = j - i + 1$, the length of the list to be searched.
Chapter 2. Algorithms and Complexity

Assume $t(n)$ to be total time to for `BINARYSEARCH (L, i, j, key)` where $n = j - i + 1$, the length of the list to be searched.

$$t(n) = c + t\left(\left\lfloor \frac{n}{2} \right\rfloor \right),$$
Chapter 2. Algorithms and Complexity

Assume $t(n)$ to be total time to for `BinarySearch` ($L, i, j, key$) where $n = j - i + 1$, the length of the list to be searched.

$$t(n) = c + t\left(\left\lfloor \frac{n}{2} \right\rfloor \right), \text{ with base case } t(0) = c'$$
Chapter 2. Algorithms and Complexity

Assume \( t(n) \) to be total time to for `BinarySearch` \((L, i, j, key)\) where \( n = j - i + 1 \), the length of the list to be searched.

\[
t(n) = c + t(\lfloor \frac{n}{2} \rfloor), \quad \text{with base case } t(0) = c'
\]

then
Chapter 2. Algorithms and Complexity

Assume $t(n)$ to be total time to for `BinarySearch` $(L, i, j, key)$ where $n = j - i + 1$, the length of the list to be searched.

$$t(n) = c + t\left(\lfloor \frac{n}{2} \rfloor \right), \text{ with base case } t(0) = c'$$

then

$$t(n) = c + t\left(\lfloor \frac{n}{2} \rfloor \right)$$
Chapter 2. Algorithms and Complexity

Assume $t(n)$ to be total time to for `BinarySearch` $(L, i, j, key)$ where $n = j - i + 1$, the length of the list to be searched.

$$t(n) = c + t(\lfloor \frac{n}{2} \rfloor), \text{ with base case } t(0) = c'$$

then

$$t(n) = c + t(\lfloor \frac{n}{2} \rfloor)$$
$$t(n) = c + c + t(\lfloor \frac{n}{2^2} \rfloor)$$
Assume $t(n)$ to be total time to for $\text{BinarySearch} (L, i, j, key)$ where $n = j - i + 1$, the length of the list to be searched.

$$t(n) = c + t(\lfloor \frac{n}{2} \rfloor), \text{ with base case } t(0) = c'$$

then

$$t(n) = c + t(\lfloor \frac{n}{2} \rfloor)$$

$$t(n) = c + c + t(\lfloor \frac{n}{2^2} \rfloor)$$

$$t(n) = c + c + c + t(\lfloor \frac{n}{2^3} \rfloor)$$
Chapter 2. Algorithms and Complexity

Assume $t(n)$ to be total time to for \texttt{BinarySearch} ($L, i, j, key$) where $n = j - i + 1$, the length of the list to be searched.

\[
t(n) = c + t\left(\left\lfloor \frac{n}{2} \right\rfloor \right), \quad \text{with base case } t(0) = c'
\]

then

\[
\begin{align*}
t(n) &= c + t\left(\left\lfloor \frac{n}{2} \right\rfloor \right) \\
t(n) &= c + c + t\left(\left\lfloor \frac{n}{2^2} \right\rfloor \right) \\
t(n) &= c + c + c + t\left(\left\lfloor \frac{n}{2^3} \right\rfloor \right) \\
&\ldots
\end{align*}
\]
Chapter 2. Algorithms and Complexity

Assume $t(n)$ to be total time to for BINARYSEARCH $(L, i, j, key)$ where $n = j - i + 1$, the length of the list to be searched.

\[ t(n) = c + t(\lceil \frac{n}{2} \rceil), \]  
with base case $t(0) = c'$

then

\[ t(n) = c + t(\lceil \frac{n}{2} \rceil) \]
\[ t(n) = c + c + t(\lceil \frac{n}{2^2} \rceil) \]
\[ t(n) = c + c + c + t(\lceil \frac{n}{2^3} \rceil) \]
\[ \ldots \]
\[ t(n) = c + c + c + \cdots + c + t(\lceil \frac{n}{2^k} \rceil) \]
Chapter 2. Algorithms and Complexity

Assume $t(n)$ to be total time to for BinarySearch $(L, i, j, key)$ where $n = j - i + 1$, the length of the list to be searched.

$$t(n) = c + t\left(\left\lfloor \frac{n}{2} \right\rfloor \right), \text{ with base case } t(0) = c'$$

then

$$t(n) = c + t\left(\left\lfloor \frac{n}{2} \right\rfloor \right)$$
$$t(n) = c + c + t\left(\left\lfloor \frac{n}{2^2} \right\rfloor \right)$$
$$t(n) = c + c + c + t\left(\left\lfloor \frac{n}{2^3} \right\rfloor \right)$$

$$\vdots$$

$$t(n) = c + c + c + \cdots + c + t\left(\left\lfloor \frac{n}{2^k} \right\rfloor \right) \text{ where } \frac{n}{2^k} = 1$$
Chapter 2. Algorithms and Complexity

Assume \( t(n) \) to be total time to for \texttt{BinarySearch} \((L, i, j, key)\) where \( n = j - i + 1 \), the length of the list to be searched.

\[
t(n) = c + t(\lfloor \frac{n}{2} \rfloor), \text{ with base case } t(0) = c'
\]

then

\[
t(n) = c + t(\lfloor \frac{n}{2} \rfloor) \]
\[
t(n) = c + c + t(\lfloor \frac{n}{2^2} \rfloor) \]
\[
t(n) = c + c + c + t(\lfloor \frac{n}{2^3} \rfloor) \]

\ldots

\[
t(n) = c + c + c + \cdots + c + t(\lfloor \frac{n}{2^k} \rfloor) \text{ where } \frac{n}{2^k} = 1
\]
\[
t(n) = c + c + c + \cdots + c + c + t(0)
\]
Chapter 2. Algorithms and Complexity

Assume \( t(n) \) to be total time to for \textsc{BinarySearch} \((L, i, j, key)\) where \( n = j - i + 1 \), the length of the list to be searched.

\[
t(n) = c + t\left(\left\lfloor \frac{n}{2} \right\rfloor\right), \quad \text{with base case } t(0) = c'
\]

then

\[
\begin{align*}
t(n) &= c + t\left(\left\lfloor \frac{n}{2} \right\rfloor\right) \\
t(n) &= c + c + t\left(\left\lfloor \frac{n}{2^2} \right\rfloor\right) \\
t(n) &= c + c + c + t\left(\left\lfloor \frac{n}{2^3} \right\rfloor\right) \\
&\quad \ldots \\
t(n) &= c + c + c + \cdots + c + t\left(\left\lfloor \frac{n}{2^k} \right\rfloor\right) \quad \text{where } \frac{n}{2^k} = 1 \\
t(n) &= c + c + c + \cdots + c + c + t(0) = k \times c + c,
\end{align*}
\]

where \( k = \log_2 n \).
Chapter 2. Algorithms and Complexity

Assume $t(n)$ to be total time to for `BinarySearch` $(L, i, j, key)$ where $n = j - i + 1$, the length of the list to be searched.

$$t(n) = c + t\left(\left\lfloor \frac{n}{2} \right\rfloor \right), \text{ with base case } t(0) = c'$$

then

$$t(n) = c + t\left(\left\lfloor \frac{n}{2} \right\rfloor \right)$$
$$t(n) = c + c + t\left(\left\lfloor \frac{n}{2^2} \right\rfloor \right)$$
$$t(n) = c + c + c + t\left(\left\lfloor \frac{n}{2^3} \right\rfloor \right)$$

... 

$$t(n) = c + c + c + \cdots + c + t\left(\left\lfloor \frac{n}{2^k} \right\rfloor \right) \text{ where } \frac{n}{2^k} = 1$$
$$t(n) = c + c + c + \cdots + c + c + t(0) = k \times c + c,$$
    where $k = \log_2 n$.
So $t(n) = c(\log_2 n + 1)$
Chapter 2. Algorithms and Complexity

**Big-$O$ notation for complexity**

$O(n)$ includes $n$, $3n + 15$, $1000n$, $0.001n$, etc.

$O(n)$ also includes $\sqrt{n}$, $\log_2 n$, etc.

$O(n^2)$ does not include $n^2$, $3n^2$, etc.

$O(n^2)$ also includes $n$, $\sqrt{n}$, etc.

$O(n^{100})$ does not include $2n$, $nn$, $n!$, etc.

Polynomial time vs exponential time. I.e., tractable problems vs intractable problems.
Big-$O$ notation for complexity

$O(n)$ includes $n$, $3n + 15$, $1000n$, $0.001n$, etc.
Chapter 2. Algorithms and Complexity

Big-O notation for complexity

\( O(n) \) includes \( n, 3n + 15, 1000n, 0.001n \), etc.
\( O(n) \) also includes \( \sqrt{n}, \log_2 n \), etc.
Chapter 2. Algorithms and Complexity

**Big-O notation for complexity**

\( O(n) \) includes \( n, 3n + 15, 1000n, 0.001n \), etc.

\( O(n) \) also includes \( \sqrt{n}, \log_2 n \), etc.

\( O(n) \) does not include \( n^2, n \log_2 n \), etc.
Chapter 2. Algorithms and Complexity

Big-$O$ notation for complexity

$O(n)$ includes $n$, $3n + 15$, $1000n$, $0.001n$, etc.
$O(n)$ also includes $\sqrt{n}$, $\log_2 n$, etc.
$O(n)$ does not include $n^2$, $n \log_2 n$, etc.
$O(n^2)$ includes $n^2$, $3n^2$, etc.
Big-$O$ notation for complexity

$O(n)$ includes $n$, $3n + 15$, $1000n$, $0.001n$, etc.
$O(n)$ also includes $\sqrt{n}$, $\log_2 n$, etc.
$O(n)$ does not include $n^2$, $n \log_2 n$, etc.
$O(n^2)$ includes $n^2$, $3n^2$, etc.
$O(n^2)$ also include $n$, $\sqrt{n}$, etc.
Chapter 2. Algorithms and Complexity

**Big-O notation for complexity**

- $O(n)$ includes $n$, $3n + 15$, $1000n$, $0.001n$, etc.
- $O(n)$ also includes $\sqrt{n}$, $\log_2 n$, etc.
- $O(n)$ does not include $n^2$, $n \log_2 n$, etc.
- $O(n^2)$ includes $n^2$, $3n^2$, etc.
- $O(n^2)$ also include $n$, $\sqrt{n}$, etc.

- $O(n^{100})$ does not include $2^n$, $n^n$, $n!$, etc.
Chapter 2. Algorithms and Complexity

**Big-O notation for complexity**

$O(n)$ includes $n$, $3n + 15$, $1000n$, $0.001n$, etc.
$O(n)$ also includes $\sqrt{n}$, $\log_2 n$, etc.
$O(n)$ does not include $n^2$, $n \log_2 n$, etc.
$O(n^2)$ includes $n^2$, $3n^2$, etc.
$O(n^2)$ also include $n$, $\sqrt{n}$, etc.

$O(n^{100})$ does not include $2^n$, $n^n$, $n!$, etc.

*polynomial time vs exponential time.*

i.e, tractable problems vs intractable problems
Chapter 2. Algorithms and Complexity

Algorithm design techniques (included in this class)
Algorithm design techniques (included in this class)

exhaustive search (including branch-and-bound)
Chapter 2. Algorithms and Complexity

**Algorithm design techniques** (included in this class)

- exhaustive search (including branch-and-bound)
- greed algorithms
Algorithm design techniques (included in this class)

exhaustive search (including branch-and-bound)
greed algorithms
dynamic programming
Chapter 2. Algorithms and Complexity

Algorithm design techniques (included in this class)

exhaustive search (including branch-and-bound)
greed algorithms
dynamic programming
divide-and-conquer
Chapter 2. Algorithms and Complexity

**Algorithm design techniques** (included in this class)

- exhaustive search (including branch-and-bound)
- greed algorithms
- dynamic programming
- divide-and-conquer
- based on combinatorics, graph theory, etc.
Algorithm design techniques (included in this class)

- exhaustive search (including branch-and-bound)
- greed algorithms
- dynamic programming
- divide-and-conquer
- based on combinatorics, graph theory, etc.
- machine learning
Chapter 2. Algorithms and Complexity

**Algorithm design techniques** (included in this class)

- exhaustive search (including branch-and-bound)
- greed algorithms
- dynamic programming
- divide-and-conquer
- based on combinatorics, graph theory, etc.
- machine learning
- randomized algorithms
Part II. Fundamental Techniques

Chapter 4. Exhaustive Search
  motif finding, median string problems

Chapter 5. Greedy Algorithms
  genome rearrangement

Chapter 6. Dynamic Programming
  sequence alignment, multiple alignment, gene finding
4.4 Regulatory motifs in DNA sequences

*Sequence motifs* regulate (turn on/off) gene expression

Example:

transcriptional binding sites TCGGGGATTCC
transcriptional factor: protein that binds to the site
allows RNA polymerase to transcribe downstream genes
called *l-mers*
Chapter 4. Exhaustive Search

Upstream sequences of genes

CGGGGCTATGCAACTGGGTCGTCACATTCCCTTTTCGATA
TTTGAGGGTGCCCAATAATGCAACTCCAAAGCGGACAAA
GGATGCAACTGATGCCGTTTGACGACCTAAATCAACGGCC
AAGGATGCAACTCCAGGAGCGCCTTTGCTGGTTCTACCTG
AATTTTCTAAAAAGATTATAATGTGGTCCATGCAACTCC
CTGCTGTACAACCTGAGATCATGCTGCTGCAATCGCAACTTTCAAC
TACATGATCTTTTGATGCAACTTTGGATGATGAGGGAATGC  motifs are underscored.
Chapter 4. Exhaustive Search

Upstream sequences of genes

CGGGGCTATGCAACTGGGTCGTCACATTTCCCTTTTCGATA
TTTGGAGGCTGCCCAATAATATGCAACTTCCAAAGCGGACAAA
GGATGCAACTGATGCGCCTTTTGGACGACCTAAATCAACGGCC
AAGGATGCAACTCCAGGAGCGCCTTTGCTGGTTCTACCTG
AATTTTCTAAAAAGATTATAATGTGGGGTCCATGCAACTTC
CTGCTGTACAACTGAGATCATGCTGGCATGCAACTTTTCAAC
TACATGATCTTTTTGATGCAACTTTGGATGATGAGGGAATGC

motifs are underscored.

CGGGGCTATGCAACTGGGTCGTCACATTTCCCTTTTCGATA
TTTGGAGGCTGCCCAATAATATGCAACTTCCAAAGCGGACAAA
GGATGCAACTGATGCGCCTTTTGGACGACCTAAATCAACGGCC
AAGGATGCAACTCCAGGAGCGCCTTTGCTGGTTCTACCTG
AATTTTCTAAAAAGATTATAATGTGGGGTCCATGCAACTTC
CTGCTGTACAACTGAGATCATGCTGGCATGCAACTTTTCAAC
TACATGATCTTTTTGATGCAACTTTGGATGATGAGGGAATGC

underlines are removed.
Chapter 4. Exhaustive Search

Upstream sequences of genes

CGGGGCTATGCAACTGGGTCGTCACATTCCCCTTTTCGATA
TTTGAGGGTGCCCACAATAAATGCAACTCTCAAAAGCGGACAAA
GGATGCAACTGATGCCGTGTTTGAGCAGACCTAAATCAACGGCC
AAGGATGCAACTCCAGGAGCCCTTTTGCAGTTCTACCTG
AATTTCTAAAAAGATTATAATGTCGGTCCATGCAACTTC
CTGCTGTGACAACCTGAGATCATGCTGCGATGCAACTTTCAAC
TACATGATCTTTTTGATGCAACTTGGATGATGAGGGAATGC motifs are underscored.

CGGGGCTATGCAACTGGGTCGTCACATTCCCCTTTTCGATA
TTTGAGGGTGCCCACAATAAATGCAACTCTCAAAAGCGGACAAA
GGATGCAACTGATGCCGTGTTTGAGCAGACCTAAATCAACGGCC
AAGGATGCAACTCCAGGAGCCCTTTTGCAGTTCTACCTG
AATTTCTAAAAAGATTATAATGTCGGTCCATGCAACTTC
CTGCTGTGACAACCTGAGATCATGCTGCGATGCAACTTTCAAC
TACATGATCTTTTTGATGCAACTTGGATGATGAGGGAATGC underlines are removed.

All motifs are the same ATGCAACT.
two mutations in every motif.

CGGGGCTATcCAgCTGGGTCGTCACATTTCCCCCTTTTCGATA
TTTGAGGGGTGCCCAATAAggGCAACTCCAAAGCGGACAAA
GGATGgAtCTGATGCCGTTTTGACGACCTAAATCAACCGGCC
AAGGAaGCAACcCCAGGAGCGCCCTTTGCTGGTTCTACCTG
AATTTTCTAAAAAGATTATAATGTCGGTGCTCtTGgAaCTTC
CTGCTGTACAAACTGAGATCATGCTGCATGCAaTTTCAAC
TACATGATCTTTTTGATGgcACTTGGATGATGAGGGAATGC
Chapter 4. Exhaustive Search

two mutations in every motif.

CGGGGCTATcCAgCTGGGTCGTCACATTTCCCCCTTTTCGATA  
TTTGAGGGTGCCCAGATTAAAaggGCAACTCCAAAGCGGACAAA  
GGATGgAtCTGATGCGCTTTTCGACGACCTAAATCAACGGCC  
AAGGAaGCAACcCCAGGAGCGCCCTTTGCTGGTTCTACCTG  
AATTTTCTAAAAAGATTATAATGTCGGTCtTGgAACTTC  
CTGCTGATCAACTGAGATCATGCTGCATGCAaTTTCAAC  
TACATGATCTTTTTGATGgcACTTGGATGATGAGGGAATGC

underscores are removed.

CGGGGCTATccAGCTGGGTCGTCACATTTCCCCCTTTTCGATA  
TTTGAGGGTGCCCAGATTAAAaggGCAACTCCAAAGCGGACAAA  
GGATGgAtCTGATGCGCTTTTCGACGACCTAAATCAACGGCC  
AAGGAaGCAACcCCAGGAGCGCCCTTTGCTGGTTCTACCTG  
AATTTTCTAAAAAGATTATAATGTCGGTCtTGgAACTTC  
CTGCTGATCAACTGAGATCATGCTGCATGCAaTTTCAAC  
TACATGATCTTTTTGATGgcACTTGGATGATGAGGGAATGC
two mutations in every motif.

CGGGGCTATcCAgCTGGGTCGTCACATTCCCCTTTTCGATA
TTTGAGGGTGCCCCAATAAagGCAACCTCCAAGCGGACAAA
GGATGgAtCTGATGCCGTTTTGACGACCTAAATCAACGGGCC
AAGGAaGCAACCcCCAGGAGCGCCTTTTGGCTGGTTCTACCTG
AATTTTCTAAAAAGATTATAATGTCGGGTCcTgAACTTC
CTGCTGTACAACTGAGATCATGCTGCAATGCAatTTTCAAC
TACATGATCTTTTTGATGcACCTTGGATGATGAGGGGAATGC

CGGGGCTATCCAGCTGGGTCGTCACATTCCCCTTTTCGATA
TTTGAGGGTGCCCCAATAAAGGGCAACCTCCAAGCGGACAAA
GGATGGATCTGATGCGTCGTTTTGACGACCTAAATCAACGGGCC
AAGGAAGCAACCcCCAGGAGCGCCTTTTGGCTGGTTCTACCTG
AATTTTCTAAAAAGATTATAATGTCGGGTCcTgAACTTC
CTGCTGTACAACTGAGATCATGCTGCAATGCAatTTTCAAC
TACATGATCTTTTTGATGcACCTTGGATGATGAGGGGAATGC    underscores are removed.

How do identify these motifs?
Chapter 4. Exhaustive Search

4.5 Profiles for motifs

CGGGGCTatatccagctGGGTGTCGTCACATTCCCTTTTCGATA
TTTGAGGGTGCCCAATAAgggcaactCCAAAGCGGACAAA
GGatggatctGATGCCGTTTGACGACCTAAATCAACGGGCC
AAGGaagcaaccCCAGGAGCGCCTTTGCTGGTCTGTTCTACCTG
CTAAAAGATTATAATGTCGGTCCttggaactTC
CTGTACATCATGCTGCatgccattTTCAAC
TACATGATCTTTTGTatggcactTGGATGATGAGGGAATGC

motifs are in lower case; they are aligned to build a profile.
Chapter 4. Exhaustive Search

4.5 Profiles for motifs

CGGGGCTatccagctGGGTCGTCACATTCCCTTTTCGATA
TTTGAGGGTGCCCAATAAgggcaactCCAAAGCGGACAAA
GGatggatctGATGCCGTTTGACGACCTAAATCAACGGGCC
AAGGaagcaacctCCAGGAGCGCCTTTGCTGGTTCTACCTG
CTAAAAAGATTATAATGTCGGTCCttggaactTC
CTGTACATCATGCTGCatgccattTTCAAC
TACATGATCTTTTGatggcactTGGATGATGAGGGAATGC

motifs are in lower case; they are aligned to build a profile.

Let $s = \{8, 19, 3, 5, 24, 17, 15\}$ be set of starting positions in sample sequences.
Chapter 4. Exhaustive Search

Then profile $P(s)$

CGGGGCTatccagctGGTGTGTCACATTCCCTTTTCGATA
TTTGAGGGGTGCCCAATAAgggcaactCCAAAGCGGACAAA
GGatggatctGATGCCGTTTGACGACCTAAATCAACGGCC
AAGGaagcaaccCCAGGAGCGCTTTGCTGGTTCTACCTG
CTAAAGATTATAATGTCGGTCTCttggaactTC
CTGTACATCATGCTGCatgccattTTCAAC
TACATGATCTTTTTGatggcactTGGATGATGAGGGAATGC
Chapter 4. Exhaustive Search

Then profile $P(s)$

CGGGGCTatccagctGGGTGTCGTCACATTCCCTTTTCGATA
TTTGAGGGTGGCAATAAaggcaactCCAAAGCGGACAAA
GGatggatctGATGCCGTTTGACGACCTAAATCAACGGCC
AAGGaagcaaccCCAGGACGCCTTTTGCTGTTTCTACCTG

Consensus of 7 motifs (of length 8)

<table>
<thead>
<tr>
<th>nucleotide</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>G</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>Consensus</td>
<td>A</td>
<td>T</td>
<td>G</td>
<td>C</td>
<td>A</td>
<td>A</td>
<td>C</td>
<td>T</td>
</tr>
</tbody>
</table>
Chapter 4. Exhaustive Search

4.6 The motif finding problem

Given profile $P(s)$

<table>
<thead>
<tr>
<th>nucleotide/position</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>G</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>6</td>
</tr>
</tbody>
</table>

Consensus: A T G C A A C T
4.6 The motif finding problem

Given profile \( P(s) \)

<table>
<thead>
<tr>
<th>nucleotide/position</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>G</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>Consensus</td>
<td>A</td>
<td>T</td>
<td>G</td>
<td>C</td>
<td>A</td>
<td>A</td>
<td>C</td>
<td>T</td>
</tr>
</tbody>
</table>

Let \( M_{P(s)}(j) \) be the largest count in column \( j \), e.g., \( M_{P(s)}(1) = 5 \)
4.6 The motif finding problem

Given profile $P(s)$

<table>
<thead>
<tr>
<th>nucleotide/position</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>G</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>Consensus</td>
<td>A</td>
<td>T</td>
<td>G</td>
<td>C</td>
<td>A</td>
<td>A</td>
<td>C</td>
<td>T</td>
</tr>
</tbody>
</table>

Let $M_{P(s)}(j)$ be the largest count in column $j$, e.g., $M_{P(s)}(1) = 5$

Then *consensus score* $Score(s, DNA) = \sum_{j=1}^{l} M_{P(s)}(j)$.

e.g. $Score(s, DNA) = 5 + 5 + 6 + 4 + 5 + 5 + 6 + 6 = 42$
Chapter 4. Exhaustive Search

4.6 The motif finding problem

Given profile $P(s)$

<table>
<thead>
<tr>
<th>nucleotide/position</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>G</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>Consensus</td>
<td>A</td>
<td>T</td>
<td>G</td>
<td>C</td>
<td>A</td>
<td>A</td>
<td>C</td>
<td>T</td>
</tr>
</tbody>
</table>

Let $M_{P(s)}(j)$ be the largest count in column $j$, e.g., $M_{P(s)}(1) = 5$

Then **consensus score** $Score(s, DNA) = \sum_{j=1}^{l} M_{P(s)}(j)$.

E.g. $Score(s, DNA) = 5 + 5 + 6 + 4 + 5 + 5 + 6 + 6 = 42$

If the motif has length $l$ and there are $t$ sequences, then
The best possible alignment has score: $l \times t$
The worst possible alignment score is $\frac{lt}{4}$
Motif Finding Problem: Given a set of DNA sequences, find a set of $l$-mers, one from each sequence, that maximizes the consensus score.
Chapter 4. Exhaustive Search

**Motif Finding Problem**: Given a set of DNA sequences, find a set of $l$-mers, one from each sequence, that maximizes the consensus score.

Input: A $t \times n$ matrix of DNA sequences and $l$, length of the pattern
Chapter 4. Exhaustive Search

Motif Finding Problem: Given a set of DNA sequences, find a set of $l$-mers, one from each sequence, that maximizes the consensus score.

Input: A $t \times n$ matrix of DNA sequences and $l$, length of the pattern
Output: An array of $t$ starting positions $s = (s_1, s_2, \ldots, s_t)$ to maximize $\text{Score}(s, DNA)$. 
**Motif Finding Problem:** Given a set of DNA sequences, find a set of $l$-mers, one from each sequence, that maximizes the consensus score.

Input: A $t \times n$ matrix of DNA sequences and $l$, length of the pattern
Output: An array of $t$ starting positions $s = (s_1, s_2, \ldots, s_t)$ to maximize $Score(s, DNA)$.

A related problem is finding a *median string*. 
Given two $l$-mers $v$ and $w$, we can compute the Hamming distance between $v$ and $w$:

$$d_H(v, w)$$

as the number of positions that differ in $v$ and $w$. 

By abusing the notation a little, let $d_H(v, s_i)$ be the Hamming distance between $v$ and the $l$-mer starting at position $s_i$ in the $i$th sequence. And define $d_H(v, s) = \sum_{i=1}^{t} d_H(v, s_i)$ where $s = \{s_1, s_2, ..., s_t\}$. 

Chapter 4. Exhaustive Search

Given two $l$-mers $v$ and $w$, we can compute the *Hamming distance* between $v$ and $w$:

$$d_H(v, w)$$ as the number of positions that differ in $v$ and $w$.

e.g., $d_H(\text{ATTGTC}, \text{ACTCTC}) = 2$
Chapter 4. Exhaustive Search

Given two $l$-mers $v$ and $w$, we can compute the Hamming distance between $v$ and $w$:

$$d_H(v, w)$$

as the number of positions that differ in $v$ and $w$.

e.g., $d_H(ATTGTC, ACTCTC) = 2$

By abusing the notation a little, let $d_H(v, s_i)$ be the Hamming distance between $v$ and the $l$-mer starting at position $s_i$ in the $i$th sequence.
Given two $l$-mers $v$ and $w$, we can compute the *Hamming distance* between $v$ and $w$:

$$d_H(v, w)$$ as the number of positions that differ in $v$ and $w$.

* e.g., $d_H(ATTGTC, ACTCTC) = 2$

By abusing the notation a little, let $d_H(v, s_i)$ be the Hamming distance between $v$ and the $l$-mer starting at position $s_i$ in the $i$th sequence.

And define

$$d_H(v, s) = \sum_{i=1}^{t} d_H(v, s_i)$$

where $s = \{s_1, s_2, \ldots, s_t\}$.
Define

\[ \text{TotalDistance}(v, DNA) = \min_{\text{all } s} (d_H(v, s)) \]

where the minimization is taken over all \( s \)'s.
Chapter 4. Exhaustive Search

Define

\[ \text{TotalDistance}(v, DNA) = \min_{s} (d_H(v, s)) \]

where the minimization is taken over all \( s \)’s.

A string \( v \) is a \textit{median string} among the set of DNA sequences if \( \text{TotalDistance}(v, DNA) \) achieves the minimum.
Define

\[ \text{TotalDistance}(v, \text{DNA}) = \min_{\text{all } s} (d_H(v, s)) \]

where the minimization is taken over all \( s \)’s.

A string \( v \) is a \textit{median string} among the set of DNA sequences if \( \text{TotalDistance}(v, \text{DNA}) \) achieves the minimum.

\textbf{Median String Problem:} Given a set of DNA sequences, find a median string.
Chapter 4. Exhaustive Search

Define

$$TotalDistance(v, DNA) = \min_{all \ s}(d_H(v, s))$$

where the minimization is taken over all $s$’s.

A string $v$ is a **median string** among the set of DNA sequences if $TotalDistance(v, DNA)$ achieves the minimum.

**Median String Problem:** Given a set of DNA sequences, find a median string.

Input: A $t \times n$ matrix DNA sequences and length $l$
Output: A string $v$ of length $l$ that minimizes $TotalDistance(v, DNA)$ over all strings of length $l$. 
Chapter 4. Exhaustive Search

Median String Problem and Motif Finding Problem are computationally equivalent!

\[ d_H(w,s) = \frac{l - \text{Score}(s, \text{DNA})}{e} \]

\[ e.g., w = ATGCAACT, s = \{8, 19, 3, 5, 24, 17, 15\}, l = 8, t = 7, \text{Score}(s, \text{DNA}) = 42. \text{ Indeed, } d_H(w,s) = 2 \times 7 = 14 = 7 \times 8 - 42. \]

But why?
Chapter 4. Exhaustive Search

**Median String Problem** and **Motif Finding Problem** are computationally equivalent!

Let \( s \) be the starting positions with consensus score \( \text{Score}(s, DNA) \)
Let \( w \) be the consensus string of the corresponding profile. Then

\[ d_{H}(w, s) = l \times t = 8 \times 7 = 56 - 42 \]

Indeed, \( d_{H}(w, s) = 14 = 7 \times 8 - 42 \)
**Median String Problem** and **Motif Finding Problem** are computationally equivalent!

Let $s$ be the starting positions with consensus score $\text{Score}(s, \text{DNA})$

Let $w$ be the consensus string of the corresponding profile. Then

$$d_H(w, s) = lt - \text{Score}(s, \text{DNA})$$
**Chapter 4. Exhaustive Search**

**Median String Problem** and **Motif Finding Problem** are computationally equivalent!

Let $s$ be the starting positions with consensus score $\text{Score}(s, DNA)$
Let $w$ be the consensus string of the corresponding profile. Then

$$d_H(w, s) = lt - \text{Score}(s, DNA)$$

e.g., $w = \text{ATGCAACT}$, $s = \{8, 19, 3, 5, 24, 17, 15\}$, $l = 8$, $t = 7$, $\text{Score}(s, DNA) = 42$. Indeed,

$$d_H(w, s) = 2 \times 7 = 14 = 7 \times 8 - 42$$
**Median String Problem** and **Motif Finding Problem** are computationally equivalent!

Let $s$ be the starting positions with consensus score $Score(s, DNA)$
Let $w$ be the consensus string of the corresponding profile. Then

$$d_H(w, s) = lt - Score(s, DNA)$$

E.g., $w =$ ATGCAACT, $s = \{8, 19, 3, 5, 24, 17, 15\}$, $l = 8$, $t = 7$, $Score(s, DNA) = 42$. Indeed,

$$d_H(w, s) = 2 \times 7 = 14 = 7 \times 8 - 42$$

But why?
The consensus string $w$ minimizes $d_H(v, s)$ over all choices of $v$ and it maximizes score $Score(s, DNA)$:
Chapter 4. Exhaustive Search

The consensus string $w$ minimizes $d_H(v, s)$ over all choices of $v$ and it maximizes score $Score(s, DNA)$:

$$d_H(w, s) = \min_{all \ v} d_H(v, s) = lt - Score(s, DNA)$$
Chapter 4. Exhaustive Search

The consensus string $w$ minimizes $d_H(v, s)$ over all choices of $v$ and it maximizes score $Score(s, DNA)$:

$$d_H(w, s) = \min_{all \ v} d_H(v, s) = lt - Score(s, DNA)$$

$$\min_{all \ s} \min_{all \ v} d_H(v, s) = lt - \max_{all \ s} Score(s, DNA)$$
Chapter 4. Exhaustive Search

The consensus string $w$ minimizes $d_H(v, s)$ over all choices of $v$ and it maximizes score $\text{Score}(s, DNA)$:

$$d_H(w, s) = \min_{all \, v} d_H(v, s) = lt - \text{Score}(s, DNA)$$

$$\min_{all \, s} \min_{all \, v} d_H(v, s) = lt - \max_{all \, s} \text{Score}(s, DNA)$$

Left is the goal of **Median Finding Problem**; and right is the goal of **Motif Finding Problem**.
The two problems can also be solved using the same technique!
Chapter 4. Exhaustive Search

The two problems can also be solved using the same technique!

Exhaustive search for **Motif Finding Problem**: By considering all \((n - l + 1)^t\) positions \(s\).
Chapter 4. Exhaustive Search

The two problems can also be solved using the same technique!

Exhaustive search for **Motif Finding Problem**: By considering all \((n - l + 1)^t\) positions \(s\).

Exhaustive search for **Median String Problem**: By considering all \(4^t\) \(l\)-mers.
Chapter 4. Exhaustive Search

The two problems can also be solved using the same technique!

Exhaustive search for **Motif Finding Problem**: By considering all \((n - l + 1)^t\) positions \(s\).

Exhaustive search for **Median String Problem**: By considering all \(4^l\) \(l\)-mers.

The two searches are similar if we consider the 4 nucleotides to be numbers.
Chapter 4. Exhaustive Search

The two problems can also be solved using the same technique!

Exhaustive search for **Motif Finding Problem**: By considering all \((n - l + 1)^t\) positions \(s\).

Exhaustive search for **Median String Problem**: By considering all \(4^l\) \(l\)-mers.

The two searches are similar if we consider the 4 nucleotides to be numbers.

*The main general issue is to consider all \(k^L\) \(L\)-mers for \(k\)-letter alphabet.*
Chapter 4. Exhaustive Search

The two problems can also be solved using the same technique!

Exhaustive search for **Motif Finding Problem**: By considering all \((n - l + 1)^t\) positions \(s\).

Exhaustive search for **Median String Problem**: By considering all \(4^l\) \(l\)-mers.

The two searches are similar if we consider the 4 nucleotides to be numbers.

*The main general issue is to consider all \(k^L\) \(L\)-mers for \(k\)-letter alphabet.*

How to enumerate them?
4.7 Search trees

The function `NextLeaf` takes an `L`-mer `a` and an integer `k` as inputs. It modifies `a` according to the following rules:

1. For `i` from `L` down to 1,
2. If `a_i` < `k`, then `a_i` ← `a_i` + 1,
3. Else, return `a`,
4. `a_i` ← 1,
5. Return `a`.

Where `a` is an `L`-mer, an array of length `L` (indexed 1 to `L`), and for `i`, 1 ≤ `i` ≤ `L`, element `a_i` has value ranging from 1 to `k`. What does the function `NextLeaf` do?
Chapter 4. Exhaustive Search

4.7 Search trees

\texttt{NextLeaf}(a, L, k)
1. \textbf{for} \; i \leftarrow L \; \textbf{to} \; 1
2. \textbf{if} \; a_i < k
3. \quad a_i \leftarrow a_i + 1
4. \quad \textbf{return} \; a
5. \quad a_i \leftarrow 1
6. \quad \textbf{return} \; a
4.7 Search trees

NextLeaf(\(a, L, k\))
1. \(\text{for } i \leftarrow L \text{ to } 1\)
2. \(\text{if } a_i < k\)
3. \(a_i \leftarrow a_i + 1\)
4. \(\text{return } a\)
5. \(a_i \leftarrow 1\)
6. \(\text{return } a\)

where

(a is an \(L\)-mer, an array of length \(L\) (indexed 1 to \(L\));

for \(i, 1 \leq i \leq L\), element \(a_i\) has value ranging from 1 to \(k\);
4.7 Search trees

NextLeaf\((a, L, k)\)
1. \textbf{for} \(i \leftarrow L\) \textbf{to} 1
2. \textbf{if} \(a_i < k\)
3. \hspace{1em} \(a_i \leftarrow a_i + 1\)
4. \hspace{1em} \textbf{return} \(a\)
5. \hspace{1em} \(a_i \leftarrow 1\)
6. \hspace{1em} \textbf{return} \(a\)

where
\(a\) is an \(L\)-mer, an array of length \(L\) (indexed 1 to \(L\));

for \(i, 1 \leq i \leq L\), element \(a_i\) has value ranging from 1 to \(k\);

What does the function \texttt{NextLeaf} do?
Chapter 4. Exhaustive Search

\texttt{NextLeaf}(a, L, k) \{ \text{with comments} \}

1. \textbf{for} \; i \leftarrow L \textbf{ to } 1 \{ \text{from low to high digits} \}

2. \textbf{if} \; a_i < k

3. \quad a_i \leftarrow a_i + 1 \{ \text{increment the first digit not yet reaching } k \} 

4. \quad \textbf{return} \; a

5. \quad a_i \leftarrow 1 \{ \text{set the digit back to 1 if having reached } k, \}
   \quad \text{carried to the next higher digit} 

6. \quad \textbf{return} \; a
Chapter 4. Exhaustive Search

\begin{align*}
\text{NextLeaf}(a, L, k) \{ \text{ with comments } \} \\
1. & \quad \textbf{for } i \gets L \textbf{ to } 1 \{ \text{ from low to high digits } \} \\
2. & \quad \textbf{if } a_i < k \\
3. & \quad a_i \gets a_i + 1 \{ \text{increment the first digit not yet reaching } k \} \\
4. & \quad \textbf{return } a \\
5. & \quad a_i \gets 1 \{ \text{set the digit back to 1 if having reached } k, \text{ carried to the next higher digit} \} \\
6. & \quad \textbf{return } a
\end{align*}

Search tree:
- each node can have \( k \) children;
- there are \( L \) levels of nodes;
- leaves are \( L \)-mers;
- \texttt{NextLeaf} is used to navigate from one leaf to the next one;
Chapter 4. Exhaustive Search

Enumerate all $L$-mers for a $k$-letter alphabet
Chapter 4. Exhaustive Search

Enumerate all $L$-mers for a $k$-letter alphabet

$\text{ALLLEAVES}(L, k)$
1. $a \leftarrow (1, \ldots, 1)$
2. $\text{continue} \leftarrow \text{TRUE}$
3. while $\text{continue}$
4. print $a$
5. $\text{NEXTLEAF}(a, L, k)$
6. if $a = (1, \ldots, 1)$
7. $\text{continue} \leftarrow \text{FALSE}$
8. return

Only go through all leaves, not internal nodes.
Enumerate all $L$-mers for a $k$-letter alphabet

\textbf{ALLLeaves}(L, k)
1. $a \leftarrow (1, \ldots, 1)$
2. $continue \leftarrow \text{TRUE}$
3. \textbf{while} continue
4. \hspace{1em} \textbf{print} $a$
5. \hspace{1em} \textbf{NextLeaf}(a, L, k)
6. \hspace{1em} \textbf{if} $a = (1, \ldots, 1)$
7. \hspace{2em} $continue \leftarrow \text{FALSE}$
8. \hspace{1em} \textbf{return}

Only go through all leaves, not internal nodes.
Chapter 4. Exhaustive Search

How big is such a search tree?
Chapter 4. Exhaustive Search

How big is such a search tree?

- Number of leaves is $k^L$ for a $k$-letter alphabet.
How big is such a search tree?

- Number of leaves is $k^L$ for a $k$-letter alphabet.
- Number of internal nodes is $(k^L - 1)/(k - 1)$. 
How big is such a search tree?

- Number of leaves is \( k^L \) for a \( k \)-letter alphabet.
- Number of internal nodes is \( (k^L - 1)/(k - 1) \).
- Total number of nodes is \( (k^{L+1} - 1)/(k - 1) \).
Chapter 4. Exhaustive Search

Figure 4.6 All 4-mers in the two-letter alphabet \( \{1, 2\} \) can be represented as leaves in a tree.
Chapter 4. Exhaustive Search

An alternative search program (going through internal nodes):

\texttt{NextVertex}(a, i, L, k)
1. \textbf{if} \ i < L \quad \{ \text{not yet at the bottom level, go one level} \}
2. \quad a_{i+1} \leftarrow 1 \quad \{ \text{deeper, follow the leftmost branch} \}
3. \quad \textbf{return} \ (a, i + 1)
4. \quad \textbf{else} \quad \{ \text{do as \texttt{NextLeaf}} \}
5. \quad \textbf{for} \ j \leftarrow L \ \textbf{to} \ 1 \quad \{ \text{when this starts, } j = L, \text{ bottom level} \}
6. \quad \quad \textbf{if} \ a_j < k \quad \{ \text{when } j \neq L, \text{ it is not at bottom level} \}
7. \quad \quad \quad a_j \leftarrow a_j + 1 \quad \{ \text{but an internal node} \}
8. \quad \quad \textbf{return}(a, j)
9. \quad \textbf{return} \ (a, 0)

Why going through internal nodes?
For the purpose of pruning tree branches (avoiding unnecessary enumerations) to save time!
Chapter 4. Exhaustive Search

An alternative search program (going through internal nodes):

\textsc{NextVertex}(a, i, L, k)
1. \textbf{if} \( i < L \) \{ not yet at the bottom level, go one level \}
2. \( a_{i+1} \leftarrow 1 \) \{ deeper, follow the leftmost branch \}
3. \textbf{return} \((a, i + 1)\)
4. \textbf{else} \{ do as \textsc{NextLeaf} \}
5. \textbf{for} \( j \leftarrow L \) \textbf{to} 1 \{ when this starts, \( j = L \), bottom level \}
6. \textbf{if} \( a_j < k \) \{ when \( j \neq L \), it is not at bottom level \}
7. \( a_j \leftarrow a_j + 1 \) \{ but an internal node \}
8. \textbf{return} \((a, j)\)
9. \textbf{return} \((a, 0)\)

Why going through internal nodes?
Chapter 4. Exhaustive Search

An alternative search program (going through internal nodes):

NextVertex \(a, i, L, k\)

1. \(\text{if } i < L \quad \{ \text{not yet at the bottom level, go one level} \}\)
2. \(a_{i+1} \leftarrow 1 \quad \{ \text{deeper, follow the leftmost branch} \}\)
3. \(\text{return } (a, i + 1)\)
4. \(\text{else} \quad \{ \text{do as NextLeaf} \}\)
5. \(\text{for } j \leftarrow L \text{ to } 1 \quad \{ \text{when this starts, } j = L, \text{ bottom level} \}\)
6. \(\text{if } a_j < k \quad \{ \text{when } j \neq L, \text{ it is not at bottom level} \}\)
7. \(a_j \leftarrow a_j + 1 \quad \{ \text{but an internal node} \}\)
8. \(\text{return } (a, j)\)
9. \(\text{return } (a, 0)\)

Why going through internal nodes?

For the purpose of pruning tree branches
(avoiding unnecessary enumerations) to save time!
Chapter 4. Exhaustive Search

The method of **branch-and-bound**: 

1. for $j ← i$ to 1
2. if $a_j < k$
3. $a_j ← a_j + 1$
4. return $(a, j)$
5. return $(a, 0)$
Chapter 4. Exhaustive Search

The method of \texttt{branch-and-bound}: 

While traversing a search tree, it is possible to skip a whole subtree rooted at certain vertex.
Chapter 4. Exhaustive Search

The method of **branch-and-bound**: While traversing a search tree, it is possible to **skip a whole subtree rooted at certain vertex**.

How?
Chapter 4. Exhaustive Search

The method of **branch-and-bound**: 

While traversing a search tree, it is possible to skip a whole subtree rooted at certain vertex. 

How? 

At each vertex, we calculate a bound – the most optimistic score of any leaves within its subtree (which will be discussed later). 

And using the following function to skip: 

```
ByPass(a,i,L,k) 
1. for j ← i to 1 
2. if a_j < k 
3. a_j ← a_j + 1 
4. return (a,j) 
5. return (a,0) 
```
The method of **branch-and-bound**:

While traversing a search tree, it is possible to skip a whole subtree rooted at certain vertex.

How?

At each vertex, we calculate a bound – the most optimistic score of any leaves within its subtree (which will be discussed later).

And using the following function to skip:

\[
\text{BYPASS}(a, i, L, k)
\]

1. **for** \( j \leftarrow i \) to 1
2. **if** \( a_j < k \)
3. \( a_j \leftarrow a_j + 1 \)
4. **return** \((a, j)\)
5. **return** \((a, 0)\)
4.8 Algorithms for Finding Motifs

First brute force algorithm for motif finding:

\[
\text{BruteForceMotifSearch}(\text{DNA}, t, n, l)
\]

1. \(\text{bestScore} \leftarrow 0\)
2. \(\text{for each } s = (s_1, ..., s_t) \text{ from } (1, ..., 1) \text{ to } (n-l+1, ..., n-l+1)\)
3. \(\text{if } \text{Score}(s, \text{DNA}) > \text{bestScore}\)
4. \(\text{bestScore} \leftarrow \text{Score}(s, \text{DNA})\)
5. \(\text{bestMotif} \leftarrow s\)
6. \(\text{return } \text{bestMotif}\)

Line 2 enumerates all tuples \((1, ..., 1)\) to \((n-l+1, ..., n-l+1)\);
4.8 Algorithms for Finding Motifs

First brute force algorithm for motif finding:

\textsc{BruteForceMotifSearch}(DNA, t, n, l)
1. \hspace{1em} bestScore \leftarrow 0
2. \hspace{1em} for each \hspace{1em} s = (s_1, ..., s_t) \hspace{1em} from \hspace{1em} (1, ..., 1) \hspace{1em} to \hspace{1em} (n - l + 1, ..., n - l + 1)
3. \hspace{1em} \hspace{1em} if \hspace{1em} Score(s, DNA) > bestScore
4. \hspace{1em} \hspace{1em} \hspace{1em} bestScore \leftarrow Score(s, DNA)
5. \hspace{1em} \hspace{1em} \hspace{1em} bestMotif \leftarrow s
6. \hspace{1em} return bestMotif
4.8 Algorithms for Finding Motifs

First brute force algorithm for motif finding:

\texttt{BruteForceMotifSearch}(DNA, t, n, l)
1. $bestScore \leftarrow 0$
2. for each $s = (s_1, ..., s_t)$ from $(1, ..., 1)$ to $(n - l + 1, ..., n - l + 1)$
3. \hspace{1em} if $Score(s, DNA) > bestScore$
4. \hspace{2em} $bestScore \leftarrow Score(s, DNA)$
5. \hspace{2em} $bestMotif \leftarrow s$
6. \hspace{1em} return $bestMotif$

Line 2 enumerates all tuples $(1, \ldots, 1)$ to $(n - l + 1, \ldots, n - l + 1)$;
Chapter 4. Exhaustive Search

Using subroutine NEXTLEAF to enumerate tuples;
Chapter 4. Exhaustive Search

Using subroutine \texttt{NextLeaf} to enumerate tuples;

\textbf{BruteForceMotifSearchAgain}(DNA, t, n, l)
1. \hspace{1em} \(s \leftarrow (1, \ldots, 1)\)
2. \hspace{1em} \(bestScore \leftarrow \text{Score}(s, DNA)\)
3. \hspace{1em} \textbf{while} forever
4. \hspace{2.5em} \(s \leftarrow \text{NextLeaf}(s, t, n - l + 1)\)
5. \hspace{2.5em} \textbf{if} \ \text{Score}(s, DNA) > bestScore
6. \hspace{3em} \(bestScore \leftarrow \text{Score}(s, DNA)\)
7. \hspace{3em} \(bestMotif \leftarrow (s_1, ..., s_t)\)
8. \hspace{1em} \textbf{if} \ \(s = (1, \ldots, 1)\)
9. \hspace{1em} \textbf{return} \ bestMotif
Chapter 4. Exhaustive Search

Using subroutine \texttt{NextLeaf} to enumerate tuples;

\texttt{BruteForceMotifSearchAgain(DNA, t, n, l)}
\begin{enumerate}
\item $s \leftarrow (1, \ldots, 1)$
\item $\text{bestScore} \leftarrow \text{Score}(s, DNA)$
\item \textbf{while} forever
\item $s \leftarrow \text{NextLeaf}(s, t, n - l + 1)$
\item \textbf{if} $\text{Score}(s, DNA) > \text{bestScore}$
\item $\text{bestScore} \leftarrow \text{Score}(s, DNA)$
\item $\text{bestMotif} \leftarrow (s_1, \ldots, s_t)$
\item \textbf{if} $s = (1, \ldots, 1)$
\item \textbf{return} $\text{bestMotif}$
\end{enumerate}

There are $(n - l + 1)^t$ such tuples;
Chapter 4. Exhaustive Search

Using subroutine NextLeaf to enumerate tuples;

BruteForceMotifSearchAgain(DNA, t, n, l)
1. \( s \leftarrow (1, \ldots, 1) \)
2. \( \text{bestScore} \leftarrow \text{Score}(s, DNA) \)
3. while forever
4. \( s \leftarrow \text{NextLeaf}(s, t, n - l + 1) \)
5. if \( \text{Score}(s, DNA) > \text{bestScore} \)
6. \( \text{bestScore} \leftarrow \text{Score}(s, DNA) \)
7. \( \text{bestMotif} \leftarrow (s_1, \ldots, s_t) \)
8. if \( s = (1, \ldots, 1) \)
9. return \( \text{bestMotif} \)

There are \((n - l + 1)^t\) such tuples;

Computing \(\text{Score}(s, DNA)\) takes \(O(l \times t)\) steps;
Chapter 4. Exhaustive Search

Using subroutine \texttt{NextLeaf} to enumerate tuples;

\texttt{BruteForceMotifSearchAgain}(DNA, t, n, l)

1. \( s \leftarrow \langle 1, \ldots, 1 \rangle \)
2. \( \text{bestScore} \leftarrow \text{Score}(s, DNA) \)
3. \textbf{while} forever
4. \( s \leftarrow \text{NextLeaf}(s, t, n - l + 1) \)
5. \textbf{if} \( \text{Score}(s, DNA) > \text{bestScore} \)
6. \( \text{bestScore} \leftarrow \text{Score}(s, DNA) \)
7. \( \text{bestMotif} \leftarrow \langle s_1, \ldots, s_t \rangle \)
8. \textbf{if} \( s = \langle 1, \ldots, 1 \rangle \)
9. \textbf{return} \text{bestMotif}

There are \((n - l + 1)^t\) such tuples;

Computing \(\text{Score}(s, DNA)\) takes \(O(l \times t)\) steps;

So the complexity is \(O(lt(n - l + 1)^t)\);
Chapter 4. Exhaustive Search

Using subroutine `NextVertex`: 
Chapter 4. Exhaustive Search

Using subroutine \textsc{NextVertex}:

\textsc{SimpleMotifSearch}(DNA, t, n, l)
1. \hspace{1em} s \leftarrow (1, \ldots, 1)
2. \hspace{1em} bestScore \leftarrow 0
3. \hspace{1em} i \leftarrow 1
4. \hspace{1em} \textbf{while} \ i > 0
5. \hspace{2em} \textbf{if} \ i < t
6. \hspace{3em} (s, i) \leftarrow \textsc{NextVertex}(s, i, t, n - l + 1)
7. \hspace{2em} \textbf{else}
8. \hspace{3em} \textbf{if} \ Score(s, DNA) > bestScore
9. \hspace{4em} bestScore \leftarrow Score(s, DNA)
10. \hspace{4em} bestMotif \leftarrow s
11. \hspace{3em} (s, i) \leftarrow \textsc{NextVertex}(s, i, t, n - l + 1)
12. \hspace{1em} \textbf{return} \ bestMotif

Still without branch-and-bound heuristics
Chapter 4. Exhaustive Search

Using subroutine `NextVertex`:

\[ \text{SimpleMotifSearch}(DNA, t, n, l) \]

1. \( s \leftarrow (1, ..., 1) \)
2. \( \text{bestScore} \leftarrow 0 \)
3. \( i \leftarrow 1 \)
4. \( \textbf{while } i > 0 \)
5. \( \quad \textbf{if } i < t \)
6. \( \quad \quad (s, i) \leftarrow \text{NextVertex}(s, i, t, n - l + 1) \)
7. \( \quad \textbf{else} \)
8. \( \quad \quad \textbf{if } \text{Score}(s, DNA) > \text{bestScore} \)
9. \( \quad \quad \quad \text{bestScore} \leftarrow \text{Score}(s, DNA) \)
10. \( \quad \quad \quad \text{bestMotif} \leftarrow s \)
11. \( \quad \quad (s, i) \leftarrow \text{NextVertex}(s, i, t, n - l + 1) \)
12. \( \textbf{return } \text{bestMotif} \)

Still without branch-and-bound heuristics
Chapter 4. Exhaustive Search

With a branch-and-bound heuristics:

1. \( s \leftarrow (1, \ldots, 1) \)
2. \( \text{bestScore} \leftarrow 0 \)
3. \( i \leftarrow 1 \)
4. while \( i > 0 \)
5. if \( i < t \)
6. \( \text{optimisticScore} \leftarrow \text{Score}(s, i, \text{DNA}) + (t - i) \cdot l \)
7. if \( \text{optimisticScore} < \text{bestScore} \)
8. \( (s, i) \leftarrow \text{ByPass}(s, i, t, n - l + 1) \)
9. else
10. \( (s, i) \leftarrow \text{NextVertex}(s, i, t, n - l + 1) \)
11. else
12. if \( \text{Score}(s, \text{DNA}) > \text{bestScore} \)
13. \( \text{bestScore} \leftarrow \text{Score}(s, \text{DNA}) \)
14. \( \text{bestMotif} \leftarrow s \)
15. \( (s, i) \leftarrow \text{NextVertex}(s, i, t, n - l + 1) \)
16. return \( \text{bestMotif} \)
Chapter 4. Exhaustive Search

With a branch-and-bound heuristics:

\textbf{BranchAndBoundMotifSearch}(DNA, t, n, l)
1. \( s \leftarrow (1, \ldots, 1) \)
2. \( \text{bestScore} \leftarrow 0 \)
3. \( i \leftarrow 1 \)
4. \textbf{while} \( i > 0 \)
5. \hspace{1em} \textbf{if} \( i < t \)
6. \hspace{2em} \text{optimisticScore} \leftarrow \text{Score}(s, i, DNA) + (t - i) \cdot l \)
7. \hspace{2em} \textbf{if} \ \text{optimisticScore} < \text{bestScore} \)
8. \hspace{3em} (s, i) \leftarrow \text{BYPASS}(s, i, t, n - l + 1) \)
9. \hspace{2em} \textbf{else} \)
10. \hspace{3em} (s, i) \leftarrow \text{NEXTVERTEX}(s, i, t, n - l + 1) \)
11. \hspace{1em} \textbf{else} \)
12. \hspace{2em} \textbf{if} \ \text{Score}(s, DNA) > \text{bestScore} \)
13. \hspace{3em} \text{bestScore} \leftarrow \text{Score}(s, DNA) \)
14. \hspace{3em} \text{bestMotif} \leftarrow s \)
15. \hspace{3em} (s, i) \leftarrow \text{NEXTVERTEX}(s, i, t, n - l + 1) \)
16. \textbf{return} \text{bestMotif} \)
Chapter 4. Exhaustive Search

4.9 Finding a median string

BruteForceMedianSearch(DNA, t, n, l)

1. bestWord ← AAA ...
   AAA
2. bestDistance ← ∞
3. for each l-mer word ← AAA ...
   AAA to TTT ...
   TTT
4. if TotalDistance(word, DNA) < bestDistance
5. bestDistance ← TotalDistance(word, DNA)
6. bestWord ← word
7. return bestWord
4.9 Finding a median string

BruteForceMedianSearch(DNA, t, n, l)
1. \[\text{bestWord} \leftarrow \text{AAA...AAA}\]
2. \[\text{bestDistance} \leftarrow \infty\]
3. \[\text{for each } l\text{-mer } \text{word} \leftarrow \text{AAA...AAA to TTT...TTT}\]
4. \[\text{if } \text{TotalDistance(} \text{word}, \text{DNA} \text{)} < \text{bestDistance}\]
5. \[\text{bestDistance} \leftarrow \text{TotalDistance(} \text{word}, \text{DNA} \text{)}\]
6. \[\text{bestWord} \leftarrow \text{word}\]
7. \[\text{return } \text{bestWord}\]
Using subroutine \texttt{NextVertex}: 

\begin{verbatim}
using subroutine NextVertex:
1. \texttt{s} ← (1,...,1)
2. \texttt{bestDistance} ← \infty
3. \texttt{i} ← 1
4. while \texttt{i} > 0
5. \texttt{if} \texttt{i} < \texttt{l}
6. \texttt{(s,i)} ← NextVertex(\texttt{s},i,\texttt{l},4)
7. \texttt{else}
8. \texttt{word} ← nucleotide string from (s1,...,sl)
9. \texttt{if TotalDistance(word,DNA)} < \texttt{bestDistance}
10. \texttt{bestDistance} ← TotalDistance(word,DNA)
11. \texttt{bestWord} ← word
12. \texttt{(s,i)} ← NextVertex(\texttt{s},i,\texttt{l},4)
13. return \texttt{bestWord}.
\end{verbatim}
Chapter 4. Exhaustive Search

Using subroutine \texttt{NextVertex}:

\texttt{SimpleMedianSearch}(DNA, t, n, l)

1. \( s \leftarrow (1, ... 1) \)
2. \( \text{bestDistance} \leftarrow \infty \)
3. \( i \leftarrow 1 \)
4. \textbf{while} \( i > 0 \)
5. \hspace{1em} \textbf{if} \( i < l \)
6. \hspace{2em} \( (s, i) \leftarrow \text{NextVertex} (s, i, l, 4) \)
7. \hspace{1em} \textbf{else}
8. \hspace{2em} \texttt{word} \leftarrow \text{nucleotide string from} \ (s_1, ..., s_l)
9. \hspace{2em} \textbf{if} \ \texttt{TotalDistance} (\texttt{word}, DNA) < \text{bestDistance}
10. \hspace{3em} \text{bestDistance} \leftarrow \text{TotalDistance} (\texttt{word}, DNA)
11. \hspace{3em} \text{bestWord} \leftarrow \texttt{word}
12. \hspace{2em} \( (s, i) \leftarrow \text{NextVertex} (s, i, l, 4) \)
13. \textbf{return} \ bestWord.
Chapter 4. Exhaustive Search

With a branch-and-bound strategy:
Chapter 4. Exhaustive Search

With a branch-and-bound strategy:

**BranchAndBoundMedianSearch**(DNA, t, n, l)
1. \( s \leftarrow (1, \ldots, 1) \)
2. \( \text{bestDistance} \leftarrow \infty \)
3. \( i \leftarrow 1 \)
4. **while** \( i > 0 \)
5. \hspace{1em} **if** \( i < l \)
6. \hspace{2em} \( \text{prefix} \leftarrow \text{nucleotide string from } (s_1, \ldots, s_i) \)
7. \hspace{2em} \( \text{optimisticDistance} \leftarrow \text{TotalDistance}(\text{prefix}, \text{DNA}) \)
8. \hspace{2em} **if** \( \text{optimisticDistance} > \text{bestDistance} \)
9. \hspace{3em} \( (s, i) \leftarrow \text{Bypass}(s, i, l, 4) \)
10. \hspace{2em} **else**
11. \hspace{3em} \( (s, i) \leftarrow \text{NextVertex } (s, i, l, 4) \)
12. \hspace{1em} **else**
13. \hspace{2em} \( \text{word} \leftarrow \text{nucleotide string from } (s_1, \ldots, s_l) \)
14. \hspace{2em} **if** \( \text{TotalDistance} (\text{word}, \text{DNA}) < \text{bestDistance} \)
15. \hspace{3em} \( \text{bestDistance} \leftarrow \text{TotalDistance} (\text{word}, \text{DNA}) \)
16. \hspace{3em} \( \text{bestWord} \leftarrow \text{word} \)
17. \hspace{2em} \( (s, i) \leftarrow \text{NextVertex } (s, i, l, 4) \)
18. **return** \( \text{bestWord} \).
Chapter 4. Exhaustive Search

How much time does it need to compute $\text{TotalDistance} (\text{word}, \text{DNA})$ and $\text{TotalDistance} (\text{prefix}, \text{DNA})$?
Chapter 4. Exhaustive Search

How much time does it need to compute

\textsc{TotalDistance}(\textit{word, DNA})?
Chapter 4. Exhaustive Search

How much time does it need to compute

\[ \text{TotalDistance}(word, DNA) \]

and

\[ \text{TotalDistance}(prefix, DNA) \]
4.9 $\frac{1}{2}$ Profile-based Motif Search

Extending the motif finding question: Once a motif profile is established, how to find a motif from a new DNA sequence which fits the profile "well"? I.e., consider the following problem:

INPUT: a DNA sequence $D$, and motif profile $P$;
OUTPUT: some position $s$ in $D$ such that $\text{Score}(s, P)$ achieves the optimal.

where $\text{Score}(s, P)$ is computed with the motif starting at position $s$ against the profile $P$. 
Chapter 4. Exhaustive Search

4.9 $\frac{1}{2}$ Profile-based Motif Search

Extending the motif finding question:

Once a motif profile is established, how to find a motif from a new DNA sequence which fits the profile “well”?
4.9 $\frac{1}{2}$ Profile-based Motif Search

Extending the motif finding question:

Once a motif profile is established, how to find a motif from a new DNA sequence which fits the profile “well”?

I.e., consider the following problem:

INPUT: a DNA sequence $D$, and motif profile $P$;
OUTPUT: some position $s$ in $D$ such that $Score(s, P)$ achieves the optimal.
4.9 1/2 Profile-based Motif Search

Extending the motif finding question:

Once a motif profile is established, how to find a motif from a new DNA sequence which fits the profile “well”?  

I.e., consider the following problem:

**INPUT**: a DNA sequence $D$, and motif profile $P$;  
**OUTPUT**: some position $s$ in $D$ such that $Score(s, P)$ achieves the optimal.  

where $Score(s, P)$ is computed with the motif starting at position $s$ against the profile $P$.  

Two components are needed for the profile-based motif search:
Chapter 4. Exhaustive Search

Two components are needed for the profile-based motif search:

(1) scanning algorithm
Chapter 4. Exhaustive Search

Two components are needed for the profile-based motif search:

1. scanning algorithm
   enumerating all positions on the DNA sequence
Two components are needed for the profile-based motif search:

(1) scanning algorithm
   enumerating all positions on the DNA sequence
(2) scoring method
Chapter 4. Exhaustive Search

Two components are needed for the profile-based motif search:

(1) scanning algorithm
   enumerating all positions on the DNA sequence

(2) scoring method
   computing score $Score(sP)$, how?
Chapter 4. Exhaustive Search

<table>
<thead>
<tr>
<th>nucleotide/position</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>G</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>Consensus</td>
<td>A</td>
<td>T</td>
<td>G</td>
<td>C</td>
<td>A</td>
<td>A</td>
<td>C</td>
<td>T</td>
</tr>
</tbody>
</table>
## Chapter 4. Exhaustive Search

### Table

<table>
<thead>
<tr>
<th>nucleotide/position</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>G</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>6</td>
</tr>
</tbody>
</table>

| Consensus           | A | T | G | C | A | A | C | T |

motif at position $s$: G T G G A A C T
## Chapter 4. Exhaustive Search

<table>
<thead>
<tr>
<th>nucleotide/position</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>G</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>Consensus</td>
<td>A</td>
<td>T</td>
<td>G</td>
<td>C</td>
<td>A</td>
<td>A</td>
<td>C</td>
<td>T</td>
</tr>
</tbody>
</table>

motif at position $s$: $G\ T\ G\ G\ A\ A\ C\ T$

* + + * + + + +
## Chapter 4. Exhaustive Search

<table>
<thead>
<tr>
<th>nucleotide/position</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>G</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>6</td>
</tr>
</tbody>
</table>

**Consensus**

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A T G C A A C T</td>
</tr>
</tbody>
</table>

**motif at position** \(s\):

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>G T G G A A C T</td>
</tr>
</tbody>
</table>

\[ * + + * + + + + + \]

One method is to use **Hamming distance**, where

\[ Score(s, P) = 2 \]
Chapter 4. Exhaustive Search

<table>
<thead>
<tr>
<th>nucleotide/position</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>G</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>Consensus</td>
<td>A</td>
<td>T</td>
<td>G</td>
<td>C</td>
<td>A</td>
<td>A</td>
<td>C</td>
<td>T</td>
</tr>
</tbody>
</table>

Motif at position $s$: G T G G A A C T

* + + * + + + + +

One method is to use Hamming distance,

$$Score(s, P) = 2$$

disadvantage?
Chapter 4. Exhaustive Search

(2) Statistical method

The profile gives probability $p_i(x)$ in the $i$th column, for $x \in \{A, C, G, T\}$, and $i = 1, 2, \ldots, 8$.

Score $(s, P) = p_1(G) \times p_2(T) \times \cdots \times p_8(T)$

But one question remains: how high a probability is for a motif to be considered acceptable?
(2) Statistical method

The profile gives probability \( p_i(x) \) in the \( i \)th column, for \( x \in \{A, C, G, T\} \), and \( i = 1, 2, \ldots, 8 \).

\[
Score(s, P) = p_1(G) \times p_2(T) \times \cdots \times p_8(T)
\]
(2) Statistical method

The profile gives probability $p_i(x)$ in the $i$th column, for $x \in \{A, C, G, T\}$,
and $i = 1, 2, \ldots, 8$.

$$Score(s, P) = p_1(G) \times p_2(T) \times \cdots \times p_8(T)$$

But one question remains:
(2) Statistical method

The profile gives probability $p_i(x)$ in the $i$th column, for $x \in \{A, C, G, T\}$, and $i = 1, 2, \ldots, 8$.

$$\text{Score}(s, P) = p_1(G) \times p_2(T) \times \cdots \times p_8(T)$$

But one question remains:

how high a probability is for a motif to be considered acceptable?