CSCI x490 Algorithms for Computational Biology

Lecture Note 1 (by Liming Cai)

January 19, 2016

Structure of the Course

Structure of the Course

- Part I. Introduction to Algorithms (Chapter 2)

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

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

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

Part I. Introduction to Algorithms

Chapter 2. Algorithms and Complexity

Part I. Introduction to Algorithms

Chapter 2. Algorithms and Complexity

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

$$Input(x) \Rightarrow Body \Rightarrow Output(y)$$

Part I. Introduction to Algorithms

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 $x = \{a_1, \cdots, a_n\}$;
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 $x = \{a_1, \cdots, a_n\}$;
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.

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 ith 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 ith 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 ith location of array a
Assignment: $a \leftarrow b$
Arithmetic: $a + b$, $a - b$, $a * b$, a/b, a^b

Chapter 2. Algorithms and Complexity

Notation conventions for algorithm writing (pseudo-code)

Memory: *variables, arrays, arguments, parameters*
Array Access: a_i: the ith location of array a
Assignment: $a \leftarrow b$
Arithmetic: $a + b$, $a - b$, $a * b$, 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: *variables, arrays, arguments, parameters*
Array Access: a_i: the ith location of array a
Assignment: $a \leftarrow b$
Arithmetic: $a + b$, $a - b$, $a * b$, a/b, a^b
Conditional:

```
if $condition$ is true
    $body$ 1
else
    $body$ 2
```

For Loop:

```
for $i \leftarrow low$ to $high$
    $body$
```

Chapter 2. Algorithms and Complexity

Notation conventions for algorithm writing (pseudo-code)

Memory: *variables, arrays, arguments, parameters*
Array Access: a_i: the ith location of array a
Assignment: $a \leftarrow b$
Arithmetic: $a + b$, $a - b$, $a * b$, a/b, a^b

```
Conditional: **if** $condition$ is true
                     $body$ 1
                 **else**
                     $body$ 2

For Loop: **for** $i \leftarrow low$ **to** $high$
                     $body$

While Loop: **while** $condition$ is true
                     $body$
```

Chapter 2. Algorithms and Complexity

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

Chapter 2. Algorithms and Complexity

Example: an iterative algorithm computing the nth 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** $(F_n)$
```

Chapter 2. Algorithms and Complexity

Example: an iterative algorithm computing the nth 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

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 $(T_1 + T_2)$;
```

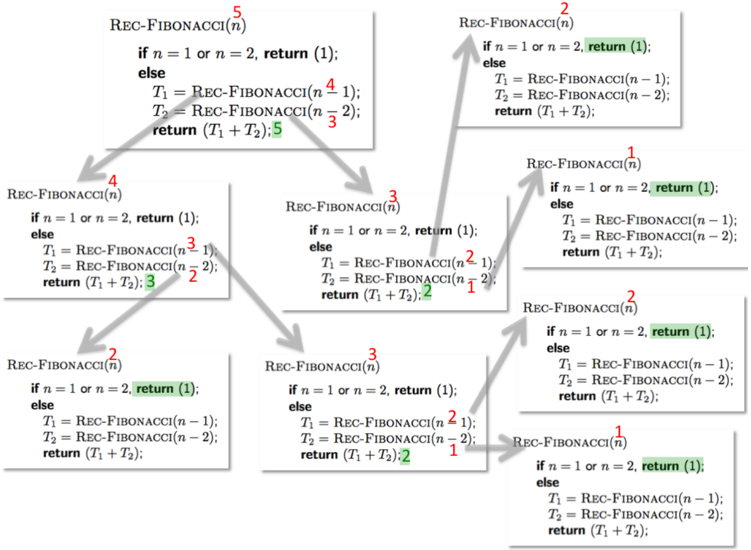
Chapter 2. Algorithms and Complexity

Recursive algorithms

```
REC-FIBONACCI($n$)
1. if $n = 1$ OR $n = 2$, return $(1)$
2. else
3.     $T_1 =$ REC-FIBONACCI($n - 1$);
4.     $T_2 =$ REC-FIBONACCI($n - 2$);
5.     return $(T_1 + T_2)$;
```

How is the algorithm executed?

Chapter 2. Algorithms and Complexity



```
REC-FIBONACCI(n)  5
    if $n = 1$ or $n = 2$, return (1);
    else
        $T_1$ = REC-FIBONACCI($n - 1$);  4
        $T_2$ = REC-FIBONACCI($n - 2$);  3
        return ($T_1 + T_2$); 5
```

```
REC-FIBONACCI(n)  4
    if $n = 1$ or $n = 2$, return (1);
    else
        $T_1$ = REC-FIBONACCI($n - 1$);  3
        $T_2$ = REC-FIBONACCI($n - 2$);  2
        return ($T_1 + T_2$); 3
```

```
REC-FIBONACCI(n)  2
    if $n = 1$ or $n = 2$, return (1);
    else
        $T_1$ = REC-FIBONACCI($n - 1$);
        $T_2$ = REC-FIBONACCI($n - 2$);
        return ($T_1 + T_2$);
```

```
REC-FIBONACCI(n)  3
    if $n = 1$ or $n = 2$, return (1);
    else
        $T_1$ = REC-FIBONACCI($n - 1$);  2
        $T_2$ = REC-FIBONACCI($n - 2$);  1
        return ($T_1 + T_2$); 2
```

```
REC-FIBONACCI(n)  3
    if $n = 1$ or $n = 2$, return (1);
    else
        $T_1$ = REC-FIBONACCI($n - 1$);  2
        $T_2$ = REC-FIBONACCI($n - 2$);  1
        return ($T_1 + T_2$); 2
```

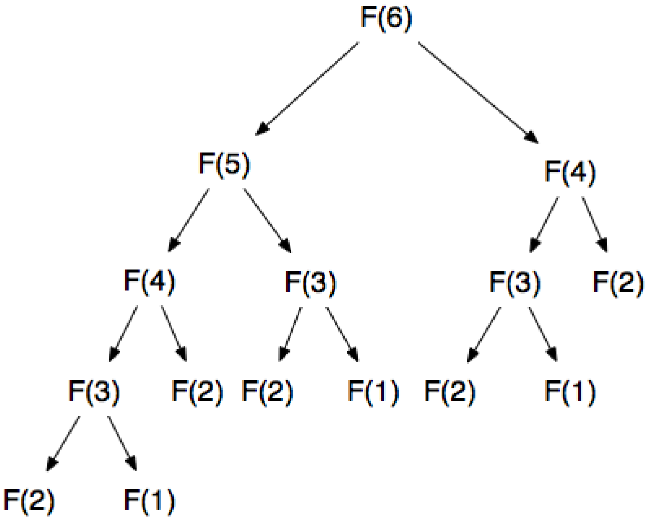
```
REC-FIBONACCI(n)  2
    if $n = 1$ or $n = 2$, return (1);
    else
        $T_1$ = REC-FIBONACCI($n - 1$);
        $T_2$ = REC-FIBONACCI($n - 2$);
        return ($T_1 + T_2$);
```

```
REC-FIBONACCI(n)  1
    if $n = 1$ or $n = 2$, return (1);
    else
        $T_1$ = REC-FIBONACCI($n - 1$);
        $T_2$ = REC-FIBONACCI($n - 2$);
        return ($T_1 + T_2$);
```

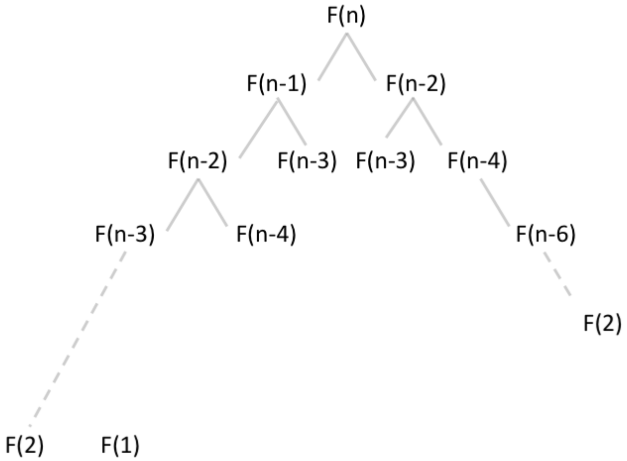
```
REC-FIBONACCI(n)  2
    if $n = 1$ or $n = 2$, return (1);
    else
        $T_1$ = REC-FIBONACCI($n - 1$);
        $T_2$ = REC-FIBONACCI($n - 2$);
        return ($T_1 + T_2$);
```

```
REC-FIBONACCI(n)  1
    if $n = 1$ or $n = 2$, return (1);
    else
        $T_1$ = REC-FIBONACCI($n - 1$);
        $T_2$ = REC-FIBONACCI($n - 2$);
        return ($T_1 + T_2$);
```

Chapter 2. Algorithms and Complexity



Chapter 2. Algorithms and Complexity



Chapter 2. Algorithms and Complexity

Advantages with recursive algorithms

Chapter 2. Algorithms and Complexity

Advantages with recursive algorithms

Example: *recursive StringCopy*

Chapter 2. Algorithms and Complexity

Advantages with recursive algorithms

Example: *recursive StringCopy*

Key ingredients for admitting recursive algorithms

More examples

Chapter 2. Algorithms and Complexity

Advantages with recursive algorithms

Example: *recursive StringCopy*

Key ingredients for admitting recursive algorithms

More examples

Sum(n), linear search,

Chapter 2. Algorithms and Complexity

Advantages with recursive algorithms

Example: *recursive StringCopy*

Key ingredients for admitting recursive algorithms

More examples
Sum(n), linear search,
summation over numbers in a 'triangle', etc.

Chapter 2. Algorithms and Complexity

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

Chapter 2. Algorithms and Complexity

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

TowersOfHanoi(First, Second, Third, n)

Chapter 2. Algorithms and Complexity

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
```

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)
```

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)
```

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

Chapter 2. Algorithms and Complexity

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

Chapter 2. Algorithms and Complexity

Algorithm Efficiency: how is it defined?

Chapter 2. Algorithms and Complexity

Algorithm Efficiency: how is it defined?

Counting the number of basic operations used

Chapter 2. Algorithms and Complexity

Algorithm Efficiency: how is it defined?

Counting the number of basic operations used

Example

```
SelectionSort($a, n$)
1 for $i \leftarrow 1$ to $n-1$
2.    $j \leftarrow i$        {starting of inner loop, assume $a_i$ to be
                              the smallest elements in $a_i, \dots, a_n$
                              $j$ memorizes the index of the smallest element }
3.    for $k \leftarrow i+1$ to $n$    {search for the smallest element}
4.       if $a_k < a_j$
5.          $j \leftarrow k$
6.    Swap $a_i$ and $a_j$
7 return array $a$
```

Chapter 2. Algorithms and Complexity

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 $a$
```

Chapter 2. Algorithms and Complexity

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:

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$(n-2)$)
```

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!

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 (RecFib($n-1$) + RecFib($n-2$))
```

Deriving and solving recurrences!

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

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

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

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

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

- 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

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

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

- 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 $t(n) = 1, \quad$ when $n = 1, 2$

Chapter 2. Algorithms and Complexity

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

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

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

Chapter 2. Algorithms and Complexity

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

where $t(n) = 1, \quad$ 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 $t(n) = 1, \quad$ 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 $t(n) = 1,\quad$ 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 $t(n) = 1, \quad$ 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 $t(n) = 1, \quad$ 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

  $\alpha^n = c + \alpha^{n-1} + \alpha^{n-2}$
  $\alpha^2 = \alpha + 1$

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

  $\alpha^n = c + \alpha^{n-1} + \alpha^{n-2}$
  $\alpha^2 = \alpha + 1$ i.e., $\alpha^2 - \alpha - 1 = 0$

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:

  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

  $\alpha^n = c + \alpha^{n-1} + \alpha^{n-2}$
  $\alpha^2 = \alpha + 1$ i.e., $\alpha^2 - \alpha - 1 = 0$
  $\alpha = (1 + \sqrt{5})/2 \approx 1.618$

Chapter 2. Algorithms and Complexity

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

where $t(n) = 1, \quad$ 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

  $\alpha^n = c + \alpha^{n-1} + \alpha^{n-2}$
  $\alpha^2 = \alpha + 1$ i.e., $\alpha^2 - \alpha - 1 = 0$
  $\alpha = (1 + \sqrt{5})/2 \approx 1.618$
  So $t(n) \approx 1.618^n$

Chapter 2. Algorithms and Complexity

Analysis of recursive algorithms (cont')

Analysis of recursive algorithms (cont')

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

```
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}
```

Analysis of recursive algorithms (cont')

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

```
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 $\textsc{BinarySearch}$ (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 $\textsc{BinarySearch}\ (L, i, j, key)$ where $n = j - i + 1$, the length of the list to be searched.

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

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.

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

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.

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

then

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.

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

then

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

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.

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

then

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

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.

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

then

$$\begin{aligned} t(n) &= c + t(\lfloor \tfrac{n}{2} \rfloor) \\ t(n) &= c + c + t(\lfloor \tfrac{n}{2^2} \rfloor) \\ t(n) &= c + c + c + t(\lfloor \tfrac{n}{2^3} \rfloor) \end{aligned}$$

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.

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

then

$$t(n) = c + t(\lfloor \tfrac{n}{2} \rfloor)$$
$$t(n) = c + c + t(\lfloor \tfrac{n}{2^2} \rfloor)$$
$$t(n) = c + c + c + t(\lfloor \tfrac{n}{2^3} \rfloor)$$
$$\ldots$$

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.

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

then

$$
\begin{aligned}
&t(n) = c + t(\lfloor \tfrac{n}{2} \rfloor) \\
&t(n) = c + c + t(\lfloor \tfrac{n}{2^2} \rfloor) \\
&t(n) = c + c + c + t(\lfloor \tfrac{n}{2^3} \rfloor) \\
&\dots \\
&t(n) = c + c + c + \cdots + c + t(\lfloor \tfrac{n}{2^k} \rfloor)
\end{aligned}
$$

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.

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

then

$$\begin{aligned}
&t(n) = c + t(\lfloor \tfrac{n}{2} \rfloor) \\
&t(n) = c + c + t(\lfloor \tfrac{n}{2^2} \rfloor) \\
&t(n) = c + c + c + t(\lfloor \tfrac{n}{2^3} \rfloor) \\
&\dots \\
&t(n) = c + c + c + \dots + c + t(\lfloor \tfrac{n}{2^k} \rfloor) \quad \text{where } \tfrac{n}{2^k} = 1
\end{aligned}$$

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.

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

then

$$\begin{aligned}
&t(n) = c + t(\lfloor \tfrac{n}{2} \rfloor)\\
&t(n) = c + c + t(\lfloor \tfrac{n}{2^2} \rfloor)\\
&t(n) = c + c + c + t(\lfloor \tfrac{n}{2^3} \rfloor)\\
&\dots\\
&t(n) = c + c + c + \cdots + c + t(\lfloor \tfrac{n}{2^k} \rfloor) \quad \text{where } \tfrac{n}{2^k} = 1\\
&t(n) = c + c + c + \cdots + c + c + t(0)
\end{aligned}$$

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.

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

then

$$
\begin{aligned}
&t(n) = c + t(\lfloor \tfrac{n}{2} \rfloor)\\
&t(n) = c + c + t(\lfloor \tfrac{n}{2^2} \rfloor)\\
&t(n) = c + c + c + t(\lfloor \tfrac{n}{2^3} \rfloor)\\
&\dots\\
&t(n) = c + c + c + \cdots + c + t(\lfloor \tfrac{n}{2^k} \rfloor) \quad \text{where } \tfrac{n}{2^k} = 1\\
&t(n) = c + c + c + \cdots + c + c + t(0) = k \times c + c,\\
&\quad \text{where } k = \log_2 n.
\end{aligned}
$$

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.

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

then

$t(n) = c + t(\lfloor \frac{n}{2} \rfloor)$
$t(n) = c + c + t(\lfloor \frac{n}{2^2} \rfloor)$
$t(n) = c + c + c + t(\lfloor \frac{n}{2^3} \rfloor)$
$\dots$
$t(n) = c + c + c + \dots + c + t(\lfloor \frac{n}{2^k} \rfloor)$ where $\frac{n}{2^k} = 1$
$t(n) = c + c + c + \dots + c + c + t(0) = k \times c + c,$
where $k = \log_2 n$.
So $t(n) = c(\log_2 n + 1)$

Chapter 2. Algorithms and Complexity

Big-O notation for complexity

Chapter 2. Algorithms and Complexity

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.

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.

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.

ı.e, tractable problems vs intractable problems

Chapter 2. Algorithms and Complexity

Algorithm design techniques (included in this class)

Chapter 2. Algorithms and Complexity

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

Chapter 2. Algorithms and Complexity

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.

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

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

Chapter 4. Exhaustive Search

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

CGGGGCT$\underline{\texttt{ATGCAACT}}$GGGTCGTCACATTCCCCTTTCGATA
TTTGAGGGTGCCCAATAA$\underline{\texttt{ATGCAACT}}$CCAAAGCGGACAAA
GG$\underline{\texttt{ATGCAACT}}$GATGCCGTTTGACGACCTAAATCAACGGCC
AAGG$\underline{\texttt{ATGCAACT}}$CCAGGAGCGCCTTTGCTGGTTCTACCTG
AATTTTCTAAAAAGATTATAATGTCGGTCC$\underline{\texttt{ATGCAACT}}$TC
CTGCTGTACAACTGAGATCATGCTGC$\underline{\texttt{ATGCAACT}}$TTCAAC
TACATGATCTTTTG$\underline{\texttt{ATGCAACT}}$TGGATGATGAGGGAATGC

motifs are underscored.

Chapter 4. Exhaustive Search

Upstream sequences of genes

CGGGGCT$\underline{\texttt{ATGCAACT}}$GGGTCGTCACATTCCCCTTTCGATA
TTTGAGGGTGCCCAATAA$\underline{\texttt{ATGCAACT}}$CCAAAGCGGACAAA
GG$\underline{\texttt{ATGCAACT}}$GATGCCGTTTGACGACCTAAATCAACGGCC
AAGG$\underline{\texttt{ATGCAACT}}$CCAGGAGCGCCTTTGCTGGTTCTACCTG
AATTTTCTAAAAAGATTATAATGTCGGTCC$\underline{\texttt{ATGCAACT}}$TC
CTGCTGTACAACTGAGATCATGCTGC$\underline{\texttt{ATGCAACT}}$TTCAAC
TACATGATCTTTTG$\underline{\texttt{ATGCAACT}}$TGGATGATGAGGGAATGC motifs are underscored.

```
CGGGGCTATGCAACTGGGTCGTCACATTCCCCTTTCGATA
TTTGAGGGTGCCCAATAAATGCAACTCCAAAGCGGACAAA
GGATGCAACTGATGCCGTTTGACGACCTAAATCAACGGCC
AAGGATGCAACTCCAGGAGCGCCTTTGCTGGTTCTACCTG
AATTTTCTAAAAAGATTATAATGTCGGTCCATGCAACTTC
CTGCTGTACAACTGAGATCATGCTGCATGCAACTTTCAAC
TACATGATCTTTTGATGCAACTTGGATGATGAGGGAATGC
```

underlines are removed.

Chapter 4. Exhaustive Search

Upstream sequences of genes

CGGGGCT$\underline{\texttt{ATGCAACT}}$GGGTCGTCACATTCCCCTTTCGATA
TTTGAGGGTGCCCAATAA$\underline{\texttt{ATGCAACT}}$CCAAAGCGGACAAA
GG$\underline{\texttt{ATGCAACT}}$GATGCCGTTTGACGACCTAAATCAACGGCC
AAGG$\underline{\texttt{ATGCAACT}}$CCAGGAGCGCCTTTGCTGGTTCTACCTG
AATTTTCTAAAAAGATTATAATGTCGGTCC$\underline{\texttt{ATGCAACT}}$TC
CTGCTGTACAACTGAGATCATGCTGC$\underline{\texttt{ATGCAACT}}$TTCAAC
TACATGATCTTTTG$\underline{\texttt{ATGCAACT}}$TGGATGATGAGGGAATGC motifs are underscored.

CGGGGCTATGCAACTGGGTCGTCACATTCCCCTTTCGATA
TTTGAGGGTGCCCAATAAATGCAACTCCAAAGCGGACAAA
GGATGCAACTGATGCCGTTTGACGACCTAAATCAACGGCC
AAGGATGCAACTCCAGGAGCGCCTTTGCTGGTTCTACCTG
AATTTTCTAAAAAGATTATAATGTCGGTCCATGCAACTTC
CTGCTGTACAACTGAGATCATGCTGCATGCAACTTTCAAC
TACATGATCTTTTGATGCAACTTGGATGATGAGGGAATGC underlines are removed.

All motifs are the same ATGCAACT.

Chapter 4. Exhaustive Search

two mutations in every motif.

$\texttt{CGGGGCT}\underline{\texttt{ATcCAgCT}}\texttt{GGGTCGTCACATTCCCCTTTCGATA}$

$\texttt{TTTGAGGGTGCCCAATAA}\underline{\texttt{ggGCAACT}}\texttt{CCAAAGCGGACAAA}$

$\texttt{GG}\underline{\texttt{ATGgAtCT}}\texttt{GATGCCGTTTGACGACCTAAATCAACGGCC}$

$\texttt{AAGG}\underline{\texttt{AaGCAACc}}\texttt{CCAGGAGCGCCTTTGCTGGTTCTACCTG}$

$\texttt{AATTTTCTAAAAAGATTATAATGTCGGTCC}\underline{\texttt{tTGgAACT}}\texttt{TC}$

$\texttt{CTGCTGTACAACTGAGATCATGCTGC}\underline{\texttt{ATGCAatT}}\texttt{TCAAC}$

$\texttt{TACATGATCTTTTG}\underline{\texttt{ATGgcACT}}\texttt{TGGATGATGAGGGAATGC}$

Chapter 4. Exhaustive Search

two mutations in every motif.

```
CGGGGCTATcCAgCTGGGTCGTCACATTCCCCTTTCGATA
TTTGAGGGTGCCCAATAAggGCAACTCCAAAGCGGACAAA
GGATGgAtCTGATGCCGTTTGACGACCTAAATCAACGGCC
AAGGAaGCAACcCCAGGAGCGCCTTTGCTGGTTCTACCTG
AATTTTCTAAAAAGATTATAATGTCGGTCCtTGgAACTTC
CTGCTGTACAACTGAGATCATGCTGCATGCAatTTTCAAC
TACATGATCTTTTGATGgcACTTGGATGATGAGGGAATGC
```

```
CGGGGCTATCCAGCTGGGTCGTCACATTCCCCTTTCGATA
TTTGAGGGTGCCCAATAAGGGCAACTCCAAAGCGGACAAA
GGATGGATCTGATGCCGTTTGACGACCTAAATCAACGGCC
AAGGAAGCAACCCCAGGAGCGCCTTTGCTGGTTCTACCTG
AATTTTCTAAAAAGATTATAATGTCGGTCCTTGGAACTTC
CTGCTGTACAACTGAGATCATGCTGCATGCAATTTTCAAC
TACATGATCTTTTGATGGCACTTGGATGATGAGGGAATGC
```

underscores are removed.

Chapter 4. Exhaustive Search

two mutations in every motif.

CGGGGCT$\underline{\texttt{ATcCAgCT}}$GGGTCGTCACATTCCCCTTTCGATA
TTTGAGGGTGCCCAATAA$\underline{\texttt{ggGCAACT}}$CCAAAGCGGACAAA
GG$\underline{\texttt{ATGgAtCT}}$GATGCCGTTTGACGACCTAAATCAACGGCC
AAGG$\underline{\texttt{AaGCAACc}}$CCAGGAGCGCCTTTGCTGGTTCTACCTG
AATTTTCTAAAAAGATTATAATGTCGGTCC$\underline{\texttt{tTGgAACT}}$TC
CTGCTGTACAACTGAGATCATGCTGC$\underline{\texttt{ATGCAatT}}$TTCAAC
TACATGATCTTTTG$\underline{\texttt{ATGgcACT}}$TGGATGATGAGGGAATGC

```
CGGGGCTATCCAGCTGGGTCGTCACATTCCCCTTTCGATA
TTTGAGGGTGCCCAATAAGGGCAACTCCAAAGCGGACAAA
GGATGGATCTGATGCCGTTTGACGACCTAAATCAACGGCC
AAGGAAGCAACCCCAGGAGCGCCTTTGCTGGTTCTACCTG
AATTTTCTAAAAAGATTATAATGTCGGTCCTTGGAACTTC
CTGCTGTACAACTGAGATCATGCTGCATGCAATTTTCAAC
TACATGATCTTTTGATGGCACTTGGATGATGAGGGAATGC
```

underscores are removed.

How do identify these motifs?

Chapter 4. Exhaustive Search

4.5 Profiles for motifs

```
                CGGGGCTatccagctGGGTCGTCACATTCCCCTTTCGATA
     TTTGAGGGTGCCCAATAAgggcaactCCAAAGCGGACAAA
                     GGatggatctGATGCCGTTTGACGACCTAAATCAACGGCC
                   AAGGaagcaaccCCAGGAGCGCCTTTGCTGGTTCTACCTG
CTAAAAGATTATAATGTCGGTCCttggaactTC
       CTGTACATCATGCTGCatgccattTTCAAC
         TACATGATCTTTTGatggcactTGGATGATGAGGGAATGC
```

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

Chapter 4. Exhaustive Search

4.5 Profiles for motifs

```
                CGGGGCTatccagctGGGTCGTCACATTCCCCTTTCGATA
     TTTGAGGGTGCCCAATAAgggcaactCCAAAGCGGACAAA
                     GGatggatctGATGCCGTTTGACGACCTAAATCAACGGCC
                   AAGGaagcaaccCCAGGAGCGCCTTTGCTGGTTCTACCTG
CTAAAAGATTATAATGTCGGTCCttggaactTC
       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 $P(s)$

```
                CGGGGCTatccagctGGGTCGTCACATTCCCCTTTCGATA
     TTTGAGGGTGCCCAATAAgggcaactCCAAAGCGGACAAA
                     GGatggatctGATGCCGTTTGACGACCTAAATCAACGGCC
                   AAGGaagcaaccCCAGGAGCGCCTTTGCTGGTTCTACCTG
CTAAAAGATTATAATGTCGGTCCttggaactTC
       CTGTACATCATGCTGCatgccattTTCAAC
         TACATGATCTTTTGatggcactTGGATGATGAGGGAATGC
                       ________
```

Chapter 4. Exhaustive Search

Then profile $P(s)$

```
                CGGGGCTatccagctGGGTCGTCACATTCCCCTTTCGATA
     TTTGAGGGTGCCCAATAAgggcaactCCAAAGCGGACAAA
                     GGatggatctGATGCCGTTTGACGACCTAAATCAACGGCC
                   AAGGaagcaaccCCAGGAGCGCCTTTGCTGGTTCTACCTG
CTAAAAGATTATAATGTCGGTCCttggaactTC
       CTGTACATCATGCTGCatgccattTTCAAC
         TACATGATCTTTTGatggcactTGGATGATGAGGGAATGC
                       ________
```

Consensus of 7 motifs (of length 8)

nucleotide/position	1	2	3	4	5	6	7	8
A	5	1	0	0	5	5	0	0
C	0	0	1	4	2	0	6	1
G	1	1	6	3	0	1	0	0
T	1	5	0	0	0	1	1	6
Consensus	A	T	G	C	A	A	C	T

Chapter 4. Exhaustive Search

4.6 The motif finding problem

Given profile $P(s)$

nucleotide/position	1	2	3	4	5	6	7	8
A	5	1	0	0	5	5	0	0
C	0	0	1	4	2	0	6	1
G	1	1	6	3	0	1	0	0
T	1	5	0	0	0	1	1	6
Consensus	`A`	`T`	`G`	`C`	`A`	`A`	`C`	`T`

Chapter 4. Exhaustive Search

4.6 The motif finding problem

Given profile $P(s)$

nucleotide/position	1	2	3	4	5	6	7	8
A	5	1	0	0	5	5	0	0
C	0	0	1	4	2	0	6	1
G	1	1	6	3	0	1	0	0
T	1	5	0	0	0	1	1	6
Consensus	A	T	G	C	A	A	C	T

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

Chapter 4. Exhaustive Search

4.6 The motif finding problem

Given profile $P(s)$

nucleotide/position	1	2	3	4	5	6	7	8
A	5	1	0	0	5	5	0	0
C	0	0	1	4	2	0	6	1
G	1	1	6	3	0	1	0	0
T	1	5	0	0	0	1	1	6
Consensus	`A`	`T`	`G`	`C`	`A`	`A`	`C`	`T`

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

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

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 $P(s)$

nucleotide/position	1	2	3	4	5	6	7	8
A	5	1	0	0	5	5	0	0
C	0	0	1	4	2	0	6	1
G	1	1	6	3	0	1	0	0
T	1	5	0	0	0	1	1	6
Consensus	A	T	G	C	A	A	C	T

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

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

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 $\frac{lt}{4}$

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.*

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 $Score(s, DNA)$.

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 $Score(s, DNA)$.

A related problem is finding a *median string*.

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.

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., $d_H(\texttt{ATTGTC}, \texttt{ACTCTC}) = 2$

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., $d_H(\texttt{ATTGTC}, \texttt{ACTCTC}) = 2$

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 ith sequence.

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., $d_H(\texttt{ATTGTC}, \texttt{ACTCTC}) = 2$

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 ith sequence.

And define

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

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

Chapter 4. Exhaustive Search

Define

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

where the minimization is taken over all s's.

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.

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*

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!

Chapter 4. Exhaustive Search

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

Chapter 4. Exhaustive Search

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

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

Chapter 4. Exhaustive Search

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,

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

Chapter 4. Exhaustive Search

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?

Chapter 4. Exhaustive Search

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

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 $Score(s, DNA)$:

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

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

Left is the goal of MEDIAN FINDING PROBLEM; and right is the goal of MOTIF FINDING PROBLEM.

Chapter 4. Exhaustive Search

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^l 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?

Chapter 4. Exhaustive Search

4.7 Search trees

Chapter 4. Exhaustive Search

4.7 Search trees

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

4.7 Search trees

```
NextLeaf($a, L, k$)
1. **for** $i \leftarrow L$ **to** $1$
2.     **if** $a_i < k$
3.         $a_i \leftarrow a_i + 1$
4.         **return** $a$
5.     $a_i \leftarrow 1$
6. **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;

Chapter 4. Exhaustive Search

4.7 Search trees

```
NEXTLEAF($a, L, k$)
1.  for $i \leftarrow L$ to $1$
2.      if $a_i < k$
3.          $a_i \leftarrow a_i + 1$
4.          return $a$
5.      $a_i \leftarrow 1$
6.  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 NEXTLEAF do?

Chapter 4. Exhaustive Search

```
NextLeaf$(a, L, k)$ { with comments }
1.  for $i \leftarrow L$ to $1$ { from low to high digits }
2.      if $a_i < k$
3.          $a_i \leftarrow a_i + 1$ {increment the first digit not yet reaching $k$}
4.          return $a$
5.      $a_i \leftarrow 1$ {set the digit back to 1 if having reached $k$,
                        carried to the next higher digit}
6.  return $a$
```

Chapter 4. Exhaustive Search

```
NextLeaf($a, L, k$) { with comments }
1.  for $i \leftarrow L$ to $1$ { from low to high digits }
2.      if $a_i < k$
3.          $a_i \leftarrow a_i + 1$ {increment the first digit not yet reaching $k$}
4.          return $a$
5.      $a_i \leftarrow 1$ {set the digit back to 1 if having reached $k$,
                    carried to the next higher digit}
6.  return $a$
```

Search tree:

- each node can have k children;
- there are L levels of nodes;
- leaves are L-mers;
- 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

```
AllLeaves($L, k$)
1.  $a \leftarrow (1, \ldots, 1)$
2.  $continue \leftarrow$ TRUE
3.  while continue
4.     print $a$
5.     NextLeaf($a, L, k$)
6.     if $a = (1, \ldots, 1)$
7.        $continue \leftarrow$ FALSE
8.  return
```

Chapter 4. Exhaustive Search

Enumerate all L-mers for a k-letter alphabet

```
AllLeaves($L, k$)
1.  $a \leftarrow (1, \ldots, 1)$
2.  $continue \leftarrow$ TRUE
3.  while continue
4.      print $a$
5.      NextLeaf($a, L, k$)
6.      if $a$ = (1,. . . , 1)
7.          $continue \leftarrow$ FALSE
8.  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.

Chapter 4. Exhaustive Search

How big is such a search tree?

- Number of leaves is k^L for a k-letter alphabet.
- Number of internal nodes is $(k^L - 1)/(k-1)$.

Chapter 4. Exhaustive Search

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

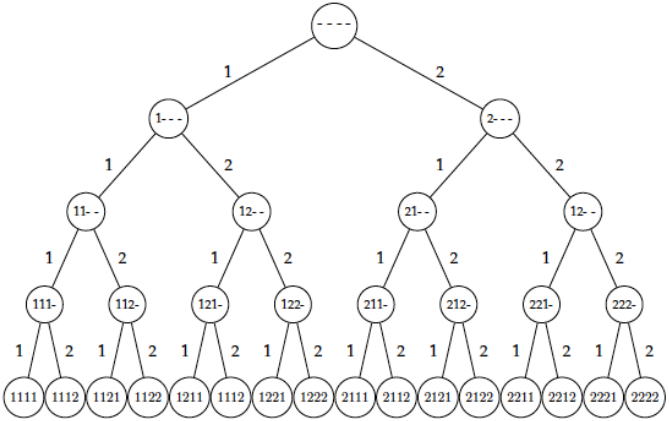


Figure 4.6 All 4-mers in the two-letter alphabet $\{1, 2\}$ can be represented as leaves in a tree.

Chapter 4. Exhaustive Search

An alternative search program (going through internal nodes):

```
NextVertex($a, i, L, k$)
1.  if $i < L$             { not yet at the bottom level, go one level }
2.      $a_{i+1} \leftarrow 1$       { deeper, follow the leftmost branch }
3.      return $(a, i+1)$
4.  else                   { do as NextLeaf }
5.      for $j \leftarrow L$ to $1$     { when this starts, $j = L$, bottom level }
6.          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.              return$(a, j)$
9.  return $(a, 0)$
```

Chapter 4. Exhaustive Search

An alternative search program (going through internal nodes):

```
NextVertex($a, i, L, k$)
1.  if $i < L$                  { not yet at the bottom level, go one level }
2.      $a_{i+1} \leftarrow 1$          { deeper, follow the leftmost branch }
3.      return $(a, i+1)$
4.  else                        { do as NextLeaf }
5.      for $j \leftarrow L$ to $1$    { when this starts, $j = L$, bottom level }
6.          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.              return$(a, j)$
9.  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.  if $i < L$                 { not yet at the bottom level, go one level }
2.      $a_{i+1} \leftarrow 1$          { deeper, follow the leftmost branch }
3.      return $(a, i+1)$
4.  else                       { do as NextLeaf }
5.      for $j \leftarrow L$ to $1$        { when this starts, $j = L$, bottom level }
6.          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.              return$(a, j)$
9.  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**:

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.

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:

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 $j \leftarrow i$ to 1
2.       if $a_j < k$
3.          $a_j \leftarrow a_j + 1$
4.          return$(a, j)$
5.    return $(a, 0)$
```

Chapter 4. Exhaustive Search

4.8 Algorithms for Finding Motifs

First brute force algorithm for motif finding:

Chapter 4. Exhaustive Search

4.8 Algorithms for Finding Motifs

First brute force algorithm for motif finding:

```
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.      if $Score(s, DNA) > bestScore$
4.        $bestScore \leftarrow Score(s, DNA)$
5.        $bestMotif \leftarrow s$
6.    return $bestMotif$
```

4.8 Algorithms for Finding Motifs

First brute force algorithm for motif finding:

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

Line 2 enumerates of all tuples $(1, \ldots, 1)$ to $(n-l+1, ..., n-l+1)$;

Chapter 4. Exhaustive Search

Using subroutine NEXTLEAF to enumerate tuples;

Chapter 4. Exhaustive Search

Using subroutine NextLeaf to enumerate tuples;

```
BruteForceMotifSearchAgain($DNA, t, n, l$)
1.  $s \leftarrow (1, ..., 1)$
2.  $bestScore \leftarrow Score(s, DNA)$
3.  while forever
4.     $s \leftarrow$ NextLeaf($s, t, n - l + 1$)
5.     if $Score(s, DNA) > bestScore$
6.        $bestScore \leftarrow Score(s, DNA)$
7.        $bestMotif \leftarrow (s1, ..., s_t)$
8.     if $s = (1, ..., 1)$
9.        return $bestMotif$
```

Chapter 4. Exhaustive Search

Using subroutine NextLeaf to enumerate tuples;

```
BruteForceMotifSearchAgain($DNA, t, n, l$)
1.  $s \leftarrow (1, ..., 1)$
2.  $bestScore \leftarrow Score(s, DNA)$
3.  while forever
4.      $s \leftarrow$ NextLeaf($s, t, n - l + 1$)
5.      if $Score(s, DNA) > bestScore$
6.          $bestScore \leftarrow Score(s, DNA)$
7.          $bestMotif \leftarrow (s1, ..., s_t)$
8.      if $s = (1, ..., 1)$
9.          return $bestMotif$
```

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

Chapter 4. Exhaustive Search

Using subroutine NextLeaf to enumerate tuples;

```
BruteForceMotifSearchAgain($DNA, t, n, l$)
1.  $s \leftarrow (1, ..., 1)$
2.  $bestScore \leftarrow Score(s, DNA)$
3.  while forever
4.     $s \leftarrow$ NextLeaf$(s, t, n - l + 1)$
5.     if $Score(s, DNA) > bestScore$
6.        $bestScore \leftarrow Score(s, DNA)$
7.        $bestMotif \leftarrow (s1, ..., s_t)$
8.     if $s = (1, ..., 1)$
9.        return $bestMotif$
```

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

Computing $Score(s, DNA)$ takes $O(l \times t)$ steps;

Chapter 4. Exhaustive Search

Using subroutine NextLeaf to enumerate tuples;

```
BruteForceMotifSearchAgain($DNA, t, n, l$)
1.  $s \leftarrow (1, ..., 1)$
2.  $bestScore \leftarrow Score(s, DNA)$
3.  while forever
4.      $s \leftarrow$ NextLeaf($s, t, n - l + 1$)
5.      if $Score(s, DNA) > bestScore$
6.          $bestScore \leftarrow Score(s, DNA)$
7.          $bestMotif \leftarrow (s1, ..., s_t)$
8.      if $s = (1, ..., 1)$
9.          return $bestMotif$
```

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

Computing $Score(s, DNA)$ 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 NextVertex:

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

Chapter 4. Exhaustive Search

Using subroutine NextVertex:

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

Still without branch-and-bound heuristics

Chapter 4. Exhaustive Search

With a branch-and-bound heuristics:

Chapter 4. Exhaustive Search

With a branch-and-bound heuristics:

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

Chapter 4. Exhaustive Search

4.9 Finding a median string

Chapter 4. Exhaustive Search

4.9 Finding a median string

```
BruteForceMedianSearch($DNA, t, n, l$)
1.  $bestWord \leftarrow$ AAA...AAA
2.  $bestDistance \leftarrow \infty$
3.  for each $l$-mer $word \leftarrow$ AAA...AAA to TTT...TTT
4.      if TotalDistance($word, DNA$) $< bestDistance$
5.          $bestDistance \leftarrow$ TotalDistance($word, DNA$)
6.          $bestWord \leftarrow word$
7.  return $bestWord$
```

Chapter 4. Exhaustive Search

Using subroutine NEXTVERTEX:

Using subroutine NEXTVERTEX:

```
SIMPLEMEDIANSEARCH($DNA, t, n, l$)
1.    $s \leftarrow (1, ...1)$
2.    $bestDistance \leftarrow \infty$
3.    $i \leftarrow 1$
4.   while $i > 0$
5.       if $i < l$
6.          $(s, i) \leftarrow$ NEXTVERTEX $(s, i, l, 4)$
7.       else
8.          $word \leftarrow$ nucleotide string from $(s_1, ..., s_l)$
9.          if TOTALDISTANCE $(word, DNA) < bestDistance$
10.             $bestDistance \leftarrow$ TOTALDISTANCE $(word, DNA)$
11.             $bestWord \leftarrow word$
12.          $(s, i) \leftarrow$ NEXTVERTEX $(s, i, l, 4)$
13.    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.   $s \leftarrow (1, ...1)$
2.   $bestDistance \leftarrow \infty$
3.   $i \leftarrow 1$
4.   while $i > 0$
5.     if $i < l$
6.       $prefix \leftarrow$ nucleotide string from $(s_1, ...s_i)$
7.       $optimisticDistance \leftarrow$ TotalDistance($prefix, DNA$)
8.       if $optimisticDistance > bestDistance$
9.         $(s, i) \leftarrow$ByPass$(s, i, l, 4)$
10.      else
11.        $(s, i) \leftarrow$ NextVertex $(s, i, l, 4)$
12.    else
13.      $word \leftarrow$ nucleotide string from $(s_1, ..., s_l)$
14.      if TotalDistance $(word, DNA) < bestDistance$
15.        $bestDistance \leftarrow$ TotalDistance $(word, DNA)$
16.        $bestWord \leftarrow word$
17.      $(s, i) \leftarrow$ NextVertex $(s, i, l, 4)$
18.  return $bestWord$.
```

Chapter 4. Exhaustive Search

Chapter 4. Exhaustive Search

How much time does it need to compute

$\textsc{TotalDistance}(word, DNA)$?

Chapter 4. Exhaustive Search

How much time does it need to compute

$\textsc{TotalDistance}(word, DNA)$?

and

$\textsc{TotalDistance}(prefix, DNA)$?

Chapter 4. Exhaustive Search

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

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" ?

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” ?

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.

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” ?

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.

Chapter 4. Exhaustive Search

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

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

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

nucleotide/position	1	2	3	4	5	6	7	8
A	5	1	0	0	5	5	0	0
C	0	0	1	4	2	0	6	1
G	1	1	6	3	0	1	0	0
T	1	5	0	0	0	1	1	6
Consensus	A	T	G	C	A	A	C	T

Chapter 4. Exhaustive Search

nucleotide/position	1	2	3	4	5	6	7	8
A	5	1	0	0	5	5	0	0
C	0	0	1	4	2	0	6	1
G	1	1	6	3	0	1	0	0
T	1	5	0	0	0	1	1	6
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

nucleotide/position	1	2	3	4	5	6	7	8
A	5	1	0	0	5	5	0	0
C	0	0	1	4	2	0	6	1
G	1	1	6	3	0	1	0	0
T	1	5	0	0	0	1	1	6
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

nucleotide/position	1	2	3	4	5	6	7	8
A	5	1	0	0	5	5	0	0
C	0	0	1	4	2	0	6	1
G	1	1	6	3	0	1	0	0
T	1	5	0	0	0	1	1	6
Consensus	A	T	G	C	A	A	C	T

motif at position s:	G	T	G	G	A	A	C	T
	*	+	+	*	+	+	+	+

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

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

Chapter 4. Exhaustive Search

nucleotide/position	1	2	3	4	5	6	7	8
A	5	1	0	0	5	5	0	0
C	0	0	1	4	2	0	6	1
G	1	1	6	3	0	1	0	0
T	1	5	0	0	0	1	1	6
Consensus	A	T	G	C	A	A	C	T

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 ith column, for $x \in \{A, C, G, T\}$,
and $i = 1, 2, \ldots, 8$.

Chapter 4. Exhaustive Search

(2) Statistical method

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

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

Chapter 4. Exhaustive Search

(2) Statistical method

The profile gives probability $p_i(x)$ in the ith 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:

Chapter 4. Exhaustive Search

(2) Statistical method

The profile gives probability $p_i(x)$ in the ith 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?