COMP 110-001
Recursion, Searching, and Selection

Yi Hong

June 12, 2015

Announcements

- Homework 4 deadline extended to June 13th, by 11:59pm
- Final exam, comprehensive
  - Wednesday, June 17th, 8am – 11am
  - Review on Monday

Today

- Introduction to Recursion
- Introduction to Search & Selection
  - Not the focus of the final exam
  - But, you should be able to **understand** the code in the slides (and **know how to use the code in similar problems by making slight modifications**).

Recursion

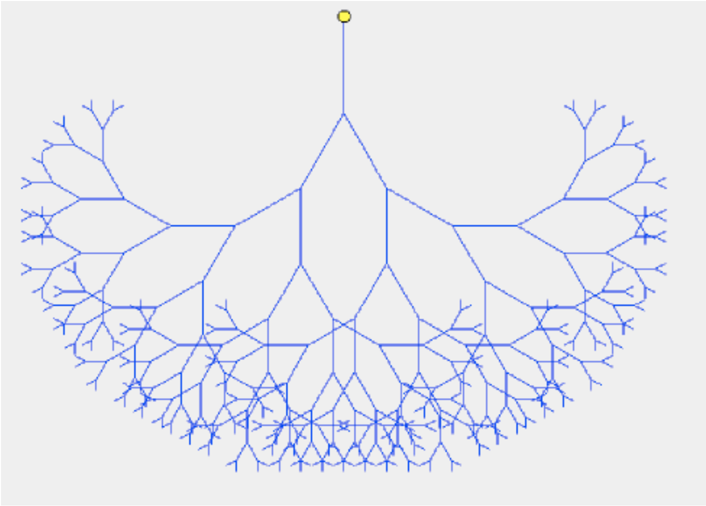
- Whenever an algorithm has one subtask that is a smaller version of the entire algorithm's task, it is said to be **recursive**
- **Recursion**: you write a method to solve a big task, and the method invokes itself to solve a smaller subtask
  - E.g., I want to eat 5 apples now. My subtask can be eating 4 apples, eating 3 apples, eating 2 apples, et…..
  - To eat 5 apples, I can do:
    - Eat 3 apples + Eat 2 apples
    - Eat 1 apple + Eat 4 apples

Eat 1 apple

Eat 1 apple

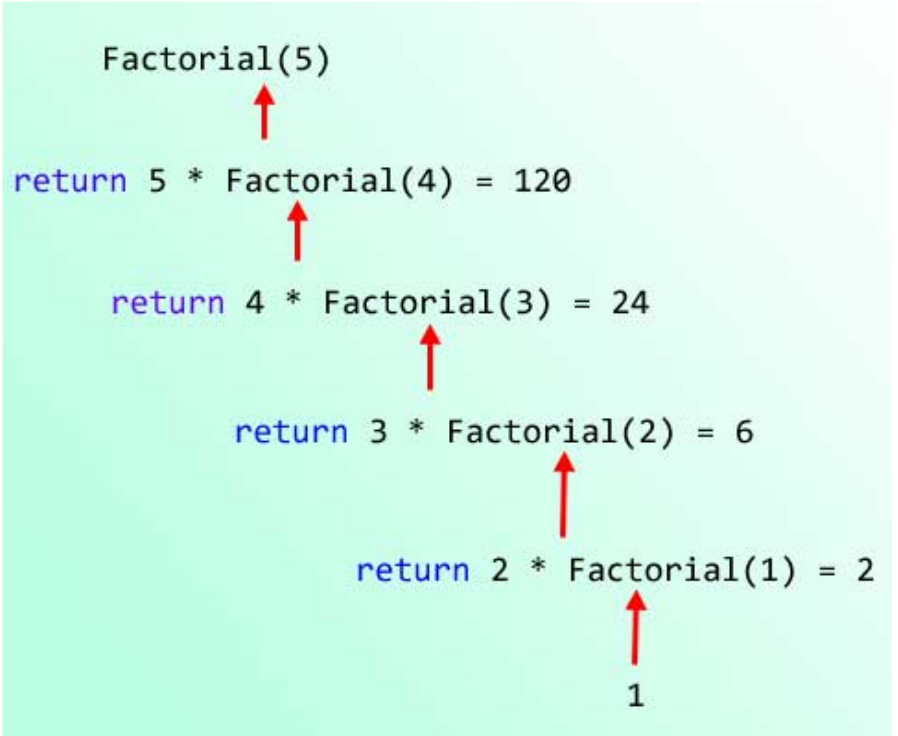
Recursion

- Eating 1 apple is the smallest task that I can have. I cannot divide it anymore.
- This is the base case in recursion.
- Recursion is to divide a big task into smaller tasks. Smaller tasks are then divided further. Until we reach base case.



Recursion

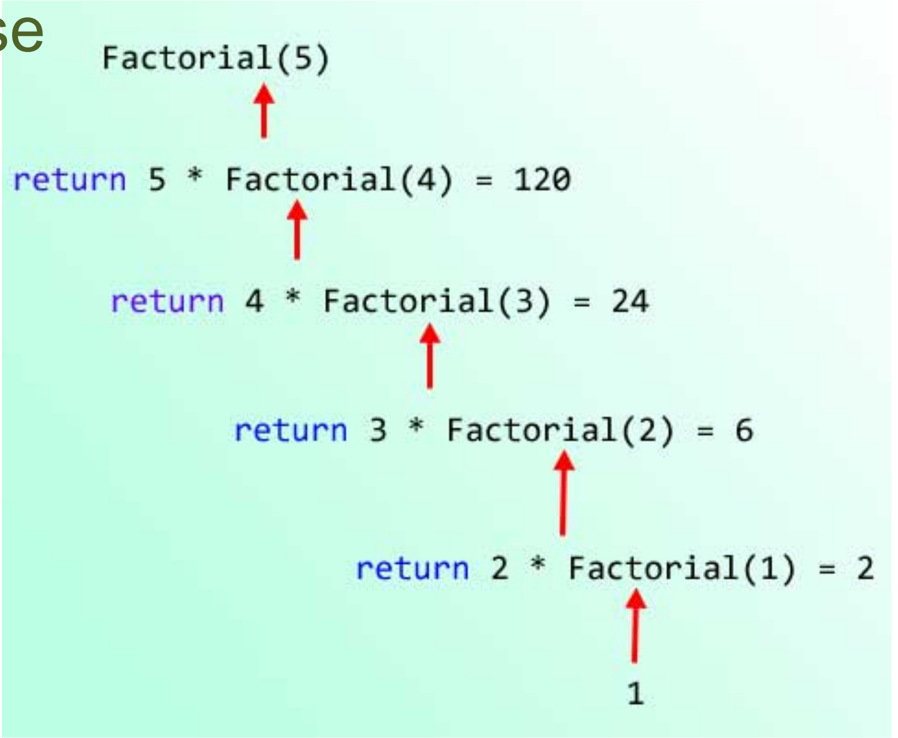
- Let's start with a simple example: calculating factorial
  - Factorial(n) = n * (n-1) * (n-2) * ….. * 3 * 2 * 1
- How do you solve a task with smaller task(s)?
  - Factorial(n) = n * Factorial(n-1)



Recursion

- Translate this into Java code

```
public static int factorial( int n )
{
 if (n==1) return 1; // base case
 else
     return n * factorial(n-1);
}
```



Recursion

- The recursion form can be more natural in many problems (than using loops)
- Some problems can be hard to formulate using naïve looping (but such problems are beyond the scope of this course)
- Let's see more recursion examples:
  - Digits to Words from textbook

Recursion: Digits to Words

- Define a method that takes a single integer as an argument and displays the digits of that integer as words.
  - For example, if the argument is the number 223, the method should display:

    two two three

- Base case?
- Recursive rule?

Recursion: Digits to Words

- Base case: only 1 digit
  - print word for 1 digit

- Recursive rule:

  Print words for n digits -->

  (print words for first n-1 digits) + (print word for last digit)

Recursion: Digits to Words

```
public static void displayAsWords( int number )
{
    if (number < 10) // base case
        System.out.print(getWordFromDigit(number) + " ");
    else      //number has two or more digits
    {
        displayAsWords(number / 10);
        System.out.print(getWordFromDigit(number % 10) + " ");
    }
}
```

You should be able to write out: getWordFromDigit(int num)

Recursion: Digits to Words

- displayAsWords(987);

```
{//Code for invocation of displayAsWords(987)
    if (987 < 10)
            System.out.print(getWordFromDigit(987) + " ");
    else//987 has two or more digits
    {
            displayAsWords(987 / 10);
            System.out.print(getWordFromDigit(987 % 10) + " ");
    }
}
```

```
{//Code for invocation of displayAsWords(98)
    if (98 < 10)
        System.out.print(getWordFromDigit(98) + " ");
    else//98 has two or more digits
    {
        displayAsWords(98 / 10);
        System.out.print(getWordFromDigit(98 % 10) + " ");
    }
}
```

```
{//Code for invocation of displayAsWords(9)
    if (9 < 10)
            System.out.print(getWordFromDigit(9) + " ");
    else//9 has two or more digits
    {
            displayAsWords(9 / 10);
            System.out.print(getWordFromDigit(9 % 10) + " ");
    }
}
```

Search

- Given a list of numbers (in an array), how do you search for a number?
  - Return index if the number is found in the array
  - Return -1 if the number is not found

Sequential (Linear) Search

- Basic idea
  - For each item in the list:
    - if that item has the desired value, stop the search and return the item's location.
  - Return *Not Found*.
- Can you do better than this (by making it faster)?
- The general answer is no
  - No assumptions made on array (unsorted)
  - In worst case, have to examine each array element at least once

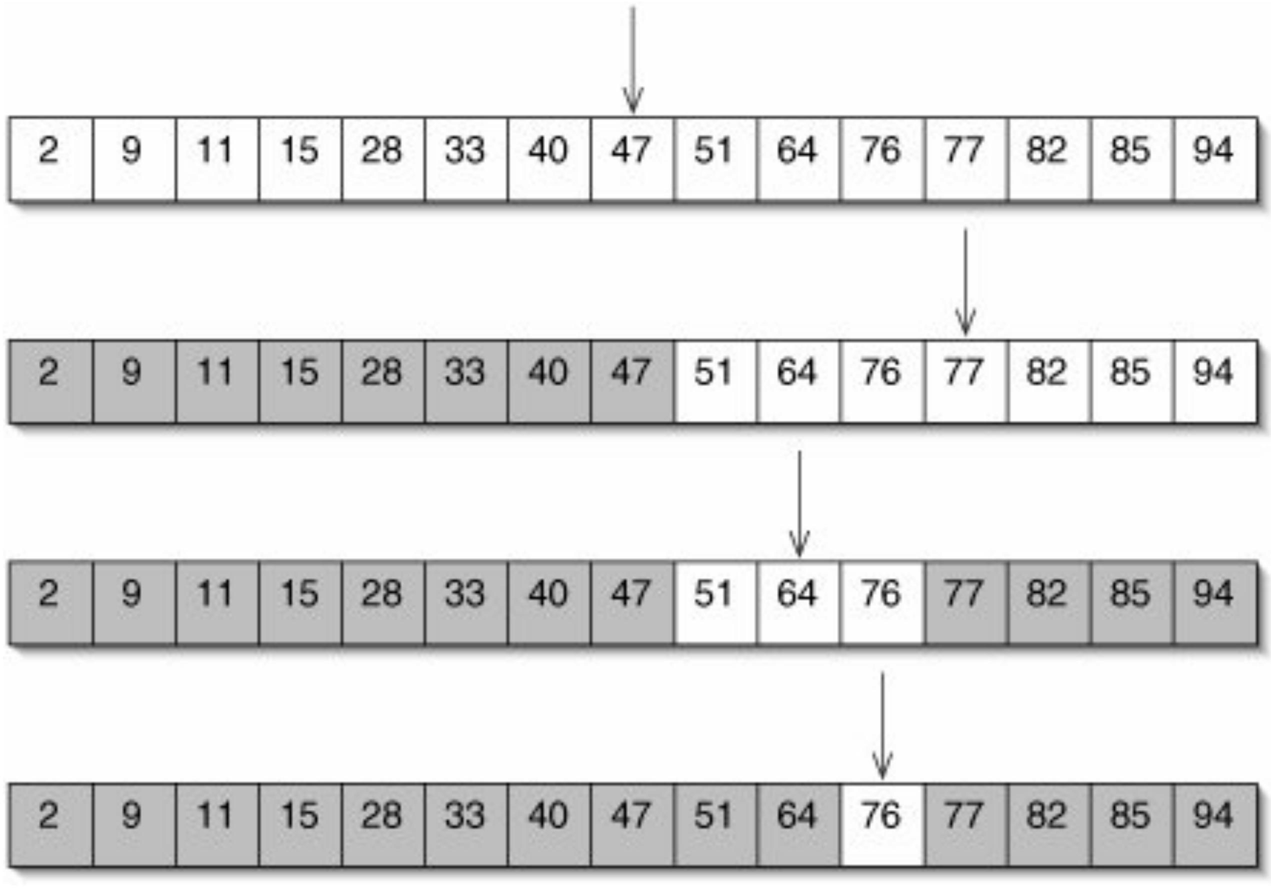
Search

- How about sorted array? (numbers are in ascending or descending order)

- Can you make the linear search faster?

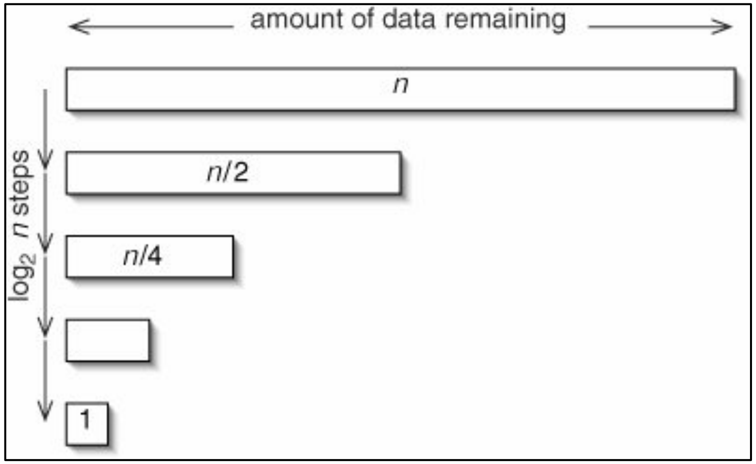
Search

- Let's see an example: searching for 76



Search

- Given **n** numbers:
  - In linear search, you need to explore one possible choice in each iteration
    - Worst case, **n** comparisons needed
  - With the new search algorithm (which only works for sorted array), we can reduce half of the search space in each iteration!
    - How many comparisons do I need in the worst case?



Binary Search

```
int binary_search(int A[], int key, int imin, int imax) {
    // test if search range is empty
    if (imax < imin) {
        return KEY_NOT_FOUND;  // set is empty
    } else {
        // calculate midpoint to cut set in half
        int imid = midpoint(imin, imax);
        // three-way comparison
        if (A[imid] > key) // key is in lower subset
            return binary_search(A, key, imin, imid-1);
        else if (A[imid] < key) // key is in upper subset
            return binary_search(A, key, imid+1, imax);
        else // key has been found
            return imid;
    }
}
```

Search Algorithms

- A lot of search algorithms, here we just covered two simplest cases:
  - Linear search in a list (array) of numbers
  - Binary search in sorted array
- More with different data structures:
  - Search in graphs and trees (computer science concepts, not the usual graph/tree)
    - E.g., search for a move in chess game
    - Search for relations/patterns in social network communication graph

Selection

- One selection problem:
  - Find the smallest / largest number in a given list (array)
  - No assumption made on the list (so it is not sorted)
- We have solved this in lab 4
  - Loop through each element, keep the largest/ smallest
- Let's relax the problem a bit

Selection

- Find the k-th smallest (or largest) element in a list of numbers

- How to solve this problem?
  - Go through each element, for each element, check its position in list
    - How many operations in the worst case?
  - Sort array first. Then get the k-th element
    - How many operations in the worst case

Selection

Quickselect (quick in practice, but not in the worst case)

- To find **k**-th smallest number in **n** numbers:
  - Randomly pick a number from the list, call it **p**
  - Partition the array into two parts:
    - Numbers that are < p **(m numbers)**
    - Numbers that are > p **(n – m – 1 numbers)**
  - If **m==k-1**, **p** is the **k**-th smallest
  - If **m > k**, find the **k**-th smallest in the m numbers
  - If **m < k**, find the **(k-m-1)**-th smallest in the **(n-m-1)** numbers
- On average, this requires ~n*constant operations
- But in the worst case, it is ~n^2*constant

Next Class

- Introduction to sorting