# Running time analysis

Read: Chapter 1.2

# Algorithm

- How to express an algorithm?
  - English
  - Programming language (e.g., Java, C/C++, Perl, etc.)
  - Pseudocode
    - A mixture of English and Programming language

# Search problem

- From banner system, search for student record
- From Bank of America system, find your account

- Assume:
  - All the cards (with numbers) in a box
- Problem:
  - Find a card with number x from the box (desired element is x)

# Sequential search algorithm

```
//Initialization
current_card =  the first card in the box
variable found = false;

//loop: search the box
while (there are still cards in the box and x is not found){
    if (the current_card’s number equals to x)
        found = true;
    else
        current_card = the next card in the box;
}
If(found = true)
    Return current_card;
Else
    Report “The element x is not found”;
```

# How to analyze this algorithm?

- Correctness
- Resource
- Running time
  - Actual elapsed time
  - Number of operations
- One operation: each time the algorithm retrieves one card from the box

# Analysis

- Given a box of cards with 10 numbers
  - 3, 5, 7, 1, 10, 5, 21, 9, 8, 34

- Search 100, 3, 1, 21?
  - How many operations do you have?

# Worse-case Time

- Given a box of cards with 10 numbers
  - 3, 5, 7, 1, 10, 5, 21, 9, 8, 34
- Search 100

- When the desired element is not in the box
- For a box with *10* cards, the worse-case requires 10 operations

- Count the maximum number of operations

# Best-case Time

- Given a box of cards with 10 numbers
  - 3, 5, 7, 1, 10, 5, 21, 9, 8, 34
- Search 3

- When the desired element is found in the beginning
- For a box with *10* cards, the best-case requires 1 operation

# Average-case Time

- Given a box of cards with 10 numbers
  - 3, 5, 7, 1, 10, 5, 21, 9, 8, 34
- Search 1, 21, ….

- Average-case running time: average the different running times for ALL different inputs
- (1+2+3+…+10)/10 = 5.5
- For a box with *10* cards, the average-case time requires 5.5 operations

# Big-O Notation

- Exact number of operations is not needed
  - Rough number is ok
- Number of operations: expressed in size of input data
  - n, (n+1)/2, 1
- Big-O
  - largest term in a formula, term with the largest exponent on n

# Typical running time

- Quadratic time: $O(n^2)$
  - $n^2 + n + 2$
- Linear time: $O(n)$
  - $3n + 10$
- Logarithmic time: $O(logn)$
  - $50logn$
- Multiplicative constants are ignored in big-O notation.
  - 2n +10, 3n + 10000
  - Lose some information
- When time analysis is expressed with big-O, the result is called the order of the algorithm.

# Examples

| n | $\text{Log}_{10}\text{n}$ | n/2 | $n^2$ | $2^n$ |
|---|---|---|---|---|
| 10 | 1 | 5 | 100 | 1K |
| 100 | 2 | 50 | 10,000 | ? |
| 1000 | 3 | 500 | 1,000,000 | ? |
| 10000 | 4 | 5000 | 100,000,000 | ? |

| Name | Value |
|---|---|
| Kilobyte | $2^{10}$ |
| Megabyte | $2^{20}$ |
| Gigabyte | $2^{30}$ |
| Terabyte | $2^{40}$ |
| Petabyte | $2^{50}$ |

# Java code implementation

```
public static boolean search(double[] data, double target){
    int i=0;
    boolean found = false;
    while(i<data.length && !found){
        if(data[i]==target){
            found = true;
        }else{
            i++;
        }
    }
    return found;
}
```

# Java code implementation

```
public static boolean search(double[] data, double target){
    int i=0;
    boolean found = false;
    while(i<data.length && !found){
        if(data[i]==target){
            found = true;
        }else{
            i++;
        }
    }
    return found;
}
```

Two assignment operations

# Java code implementation

```
public static boolean search(double[] data, double target){
    int i=0;
    boolean found = false;
    while(i<data.length && !found){
        if(data[i]==target){
            found = true;
        }else{
            i++;
        }
    }
    return found;
}
```

Two assignment operations

Each iteration (k operations)
about 3-5 operations

# Java code implementation

```java
public static boolean search(double[] data, double target){
    int i=0;
    boolean found = false;
    while(i<data.length && !found){
        if(data[i]==target){
            found = true;
        }else{
            i++;
        }
    }
    return found;
}
```

Two assignment operations

Each iteration (k operations)
about 3-5 operations

One operation

Total number of operations: n*k+ 3

# Analysis

- Step 1: define what n is. E.g., Let n be data.length.
- Step 2: analyze the number of operations in each step.
  - Special attention should be paid to loops.
  - Need to identify three major components
    - 1) How many iterations are in the loop?
    - 2) In each iterations, how many operations are needed?
    - 3) Overall, how many operations for a loop
- Step 3: calculate the overall cost, which is a function of n.
- Step 4: derive the Big-O complexity from the function of n.

# Frequent Linear Pattern

- *O(n)* time: A loop that does a fixed amount of operations *n* times
- For each student in the class, find another student that matches his/her searching criteria.
  - *O($n^2$)*