Lecture 1
Introduction to Data Structures

Instructor: Kevin K.F. YUEN, PhD.

Our Roadmap

◈ Programming basics

◈ How to solve a problem? by an algorithm?

◈ How to analyze the running time of an algorithm?

◈ What are data structures?

Programming

◈ This is not a programming course

◈ We will introduce some basic programming concepts (in the Java language)

  ◈ so that you can do practices on computers

- Building blocks in a Java program:
  - Variables, constants, operators (e.g., `x`, `y`, `3`, `5`, `+`, `=`)
  - → Expressions (e.g., `x+y`)
  - → Statements (e.g., `int x=3;`)
  - → Methods (e.g., `main`)
  - → Classes (e.g., `Test`)
- A *variable* can be used to store a value
  - Primitive type (e.g., `int`, `float`, `boolean`) or object type (e.g., `String`)
- The execution starts at the *main* method

Write a program in the file "Test.java"

```java
public class Test {
    public static void main(String args[]) {
        int x = 3;
        int y = 5;
        String msg = "hello";
        System.out.println( x+y );
        System.out.println( msg );
    }
}
```

Compile a program

javac Test.java

Run a program

java Test

8
hello

Java: Array

- *Array*: for storing multiple variables (of the same type)
    - Create an array *A* with *n* items

      ```
      int[] A = new int[n];
      ```
    - *A*[*i*] denotes the variable at position *i* of array *A*
        - In Java, the range of *i* is from 0 to $n - 1$
- Examples
    - `int[] A = new int[7];` // create an array
    - `x = A[2];`   // copy value *A*[2] to variable *x*
    - `A[3] = 5;`   // store value 5 to *A*[3]

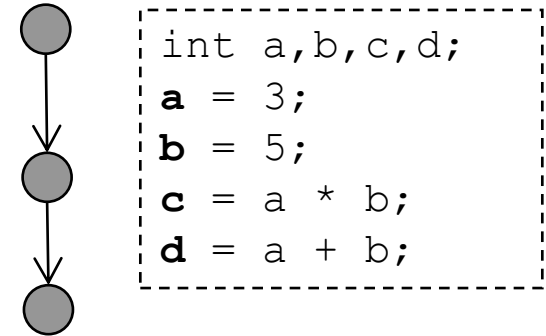| Position	0	1	2	3	4	5	6
Value	6	2	7	8	4	9	1

x: __   5

Control Flow

- **Sequence** (step-by-step)
  - A sequence of statements implies the execution ordering
  - Each statement is a 'step', e.g.,
    - Assignment statement: `c = a * b;`
    - Call statement: `System.out.println("hello");`
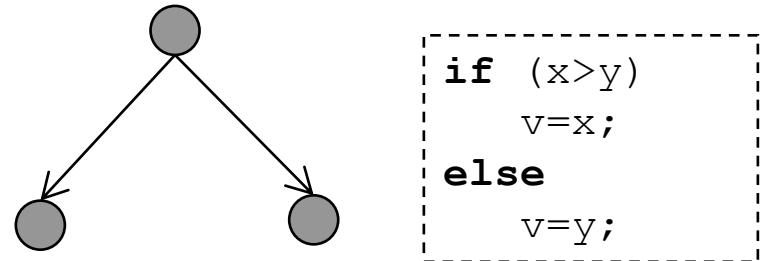
```
int a,b,c,d;
a = 3;
b = 5;
c = a * b;
d = a + b;
```

- **Selection** (choose a branch)
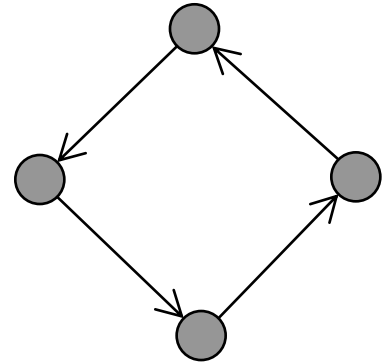 - *if* statement, *switch* statement
  - *Example* of *if* statement:
    - Evaluate the condition `x>y`
    - Condition is true → execute `v=x;`
    - Condition is false → execute `v=y;`

```
if (x>y)
    v=x;
else
    v=y;
```

Control Flow

◈ **Loop**: repeat the execution of the loop body (a sequence of statements) for multiple times

◈ Example: print "hello" 5 times

```
int i = 5;
while (i>0) {
    System.out.println("hello");
    i--;
}
```

◈ *while*-loop

  ◈ Evaluate <cond> **before** the loop body; the loop stops when <cond> is false

```
for (int i=0; i<5; i++) {
    System.out.println("hello");
}
```

◈ *for*-loop: counter-based

  ◈ Execute <init> once before the loop starts

  ◈ Evaluate <cond> **before** the loop body; the loop stops when <cond> is false

  ◈ Execute <update> **after** the loop body

7

Our Roadmap

⬥ Programming basics

⬥ How to solve a problem? by an algorithm?

⬥ How to analyze the running time of an algorithm?

⬥ What are data structures?

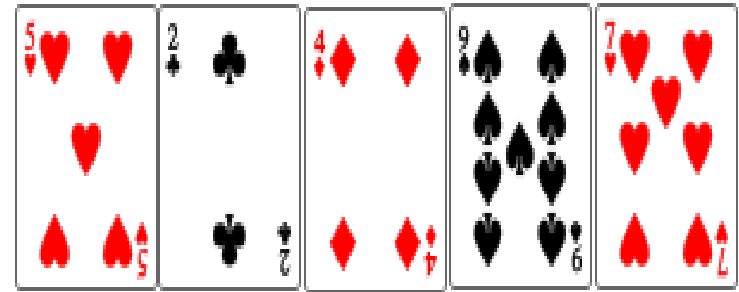# How to solve a problem?

* *A problem for human (in the real world)*:
  ==Sort== poker cards on hand
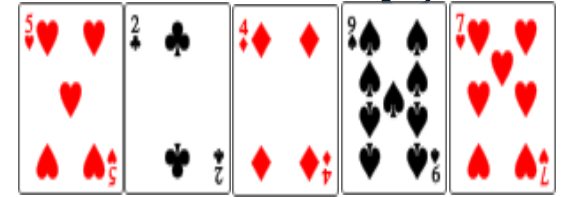
* Input:
  * A list of cards

* Procedure:
  * 1. **Put** all cards on the left hand ✋
  * 2. **Pick** the smallest card from the left hand ✋
  * 3. **Move** that card to the right hand ✋
  * 4. Repeat Steps 2-3 until the left hand is empty ✋

* How to solve a problem by using a program?
  * Use variables, operators, statements, ……

Algorithms for Problem Solving

Input: a sequence *A* of *n* integers
Output: a sequence *A* of *n* integers
such that ……

Define a computational problem

Selection-Sort (…)
1. for integer $i \leftarrow 0$ to $n-2$
2. $k \leftarrow i$
3.   for integer $j \leftarrow i+1$ to $n-1$
4.   if $A[k] > A[j]$ then
            ……

Design an algorithm

void Selection-Sort (…)
int $i, j, k, temp$;
int $n = A.length$;
for ($i=0$; $i<n-1$; $i++$) {
    …
    for ($j=i+1$; $j<n$; $j++$)
        …
    …
}

Convert it into a program

Real-world input

5 | 2 | 4 | 9 | 7

Input
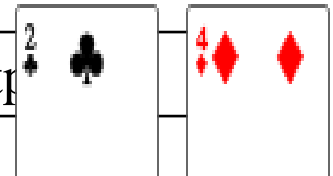
Run a program

Output

2 | 4 | 5 | 7 | 9

Real-world output

Computational problem

◈ *Example*: the **sorting** problem

   ◈ Input: a *sequence A* of *n* integers

$$5 \mid 2 \mid 4 \mid 9 \mid 7$$

   ◈ Output: a sequence *A'* of *n* integers such that

$$2 \mid 4 \mid 5 \mid 7 \mid 9$$

      ◆ *A'* is an ***increasing*** sequence, and
      ◆ *A* and *A'* contain the same integers.

Algorithms

- *Algorithm*: a well defined <u>sequence of steps</u> for solving a <u>computational problem</u>
  - It always produces the *correct output*
  - It uses *well-defined* steps or operations
  - It finishes in *finite time*
- It is written for human readers
  - Like programming, we can use variables and data structures, control flow, call statement
  - We can also use natural language provided that it is clear

Selection-Sort (Array $A[0..n–1]$)
1. for integer $i \leftarrow 0$ to $n–2$
2. $\quad k \leftarrow i$
3. $\quad$ for integer $j \leftarrow i+1$ to $n–1$
4. $\quad\quad$ if $A[k] > A[j]$ then
5. $\quad\quad\quad k \leftarrow j$
6. $\quad$ swap $A[i]$ and $A[k]$

Algorithm vs. Program

◈ An algorithm can be implemented as a program in any language (e.g., C, C++, Java, Python)

◈ An algorithm is more *readable* than a program

Algorithm

Selection-Sort (Array $A[0..n{-}1]$)
1. for integer $i \leftarrow 0$ to $n{-}2$
2. $k \leftarrow i$
3.   for integer $j \leftarrow i{+}1$ to $n{-}1$
4.   if $A[k] > A[j]$ then
5. $k \leftarrow j$
6.   swap $A[i]$ and $A[k]$

Java program

```
// K.K.F. Yuen
void SelectionSort(int[] A){
int i, j, k, temp;
int n = A.length;
    for (i = 0; i<n-1; i ++){
        k = i ;
        for ( j=i+1 ; j<n; j++ ){
        if  ( A[k] > A[j] )
            k = j ;
        }
        temp = A[i] ;
        A[i] = A[k] ;
        A[k] = temp ;
    }
}
```

Or i <= n-2

A trivial mistake is NOT trivial

Error result

```java
void SelectionSort(int[] A){
    int i,j,k,temp;
    int n = A.length;
    for (i = 0; i<n-1; i++){
        k = i;
        for (j = i+1; j<n; j++){
            if(A[k] > A[j])
                k = j;
            temp = A[i];
            A[i] = A[k];
            A[k] =temp;
        }
    }
}
```

```java
// K.K.F. Yuen
void SelectionSort(int[] A){
int i, j, k, temp;
int n = A.length;
    for (i = 0; i<n-1; i ++){
        k = i ;
        for ( j=i+1 ; j<n; j++ ){
        if  ( A[k] > A[j] )
            k = j ;
        }
        temp = A[i] ;
        A[i] = A[k] ;
        A[k] = temp ;
    }
}
```
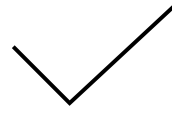
```java
void SelectionSort(int[] A){
int i, j, k, temp;
int n = A.length;
    for (i = 0; i<n-1; i ++){
        k = i ;
        for ( j=i+1 ; j<n; j++ )
        if  ( A[k] > A[j] )
            k = j ;
        temp = A[i] ;
        A[i] = A[k] ;
        A[k] = temp ;
    }
}
```

Selection-Sort (Array $A[0..n-1]$)
1. for integer $i \leftarrow 0$ to $n-2$
2. $k \leftarrow i$
3.   for integer $j \leftarrow i+1$ to $n-1$
4.   if $A[k] > A[j]$ then
5. $k \leftarrow j$
6.   swap $A[i]$ and $A[k]$

Java

```java
void SelectionSort(int[] A){
int i, j, k, temp;
int n = A.length;
    for (i = 0; i<n-1; i ++){
        k = i ;
        for ( j=i+1 ; j<n; j++ ){
        if  ( A[k] > A[j] )
            k = j ;
        }
        temp = A[i] ;
        A[i] = A[k] ;
        A[k] = temp ;
    }
}
```

R

```r
SelectionSort  <- function(A)
{
    n = length(A)
    for (i in 1:(n-1)){
     k = i
        for (j in (i+1):(n))
        {
            if (A[k] > A[j]){
                k <- j}
        }
    # swap A[i] and A[k]
    temp = A[i]
    A[i] = A[k]
    A[k] = temp}
    A
}
```

Python

```python
def selectionSort2(A):
    for i in range(0,len(A)-1):
        k = i
        for j in range(i+1,len(A)):
            if(A[k] > A[j]):
                k = j;
        # swap A[i] and A[k]
        A[i],A[k] = A[k],A[i]
    return A
```

Algorithms: Running Steps

◈ Run an algorithm manually on a sample input, then draw *running steps*

  ◈ E.g., show the content of important variables in each iteration

  ◈ Useful for understanding how the algorithm works

Selection-Sort (Array $A[0..n–1]$)
1. for integer $i \leftarrow$ 0 to $n–2$
2. $k \leftarrow i$
3.   for integer $j \leftarrow i+1$ to $n–1$
4.   if $A[k] > A[j]$ then
5. $k \leftarrow j$
6.   swap $A[i]$ and $A[k]$

input   5 | 2 | 4 | 9 | 7

$i = 1$ 2 | 5 | 4 | 9 | 7

$i = 2$ 2 | 4 | 5 | 9 | 7

$i = 3$ 2 | 4 | 5 | 9 | 7

$i = 4$ 2 | 4 | 5 | 7 | 9

$i = 5$ 2 | 4 | 5 | 7 | 9

Algorithmic Analysis

5 | 2 | 4 | 9 | 7

8 | 1 | 6 | 4 | 7

1 | 3 | 5 | 7 | 9

. . . . . .

- ◈ There are many possible **input instances**

- ◈ Is your algorithm always **correct**?
    - ◈ Not enough even if you have tested your algorithm on many instances
        - ◆ Will your algorithm fail on some other instance?
    - ◈ Need a **correctness proof** to show that:
        - ◆ "For every possible input instance, the algorithm produces the correct output"

Running time

Worst-case

Best-case

instance ID

- ◈ **How fast** is your algorithm?
    - ◈ Consider all instances of the same size n and their running time
    - ◈ Estimate the worst-case running time as a function of the input size n
    - ◈ What's the trend of the worst-case running time?
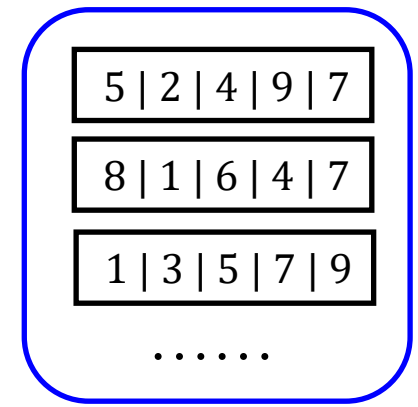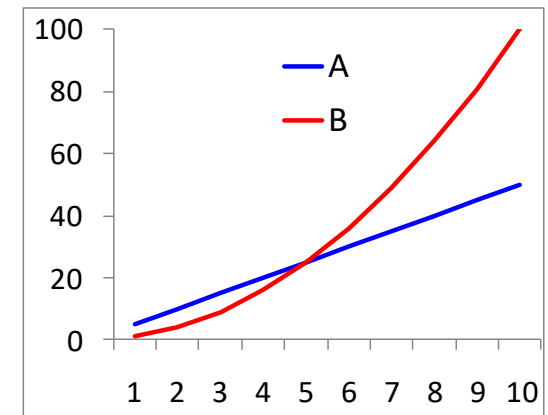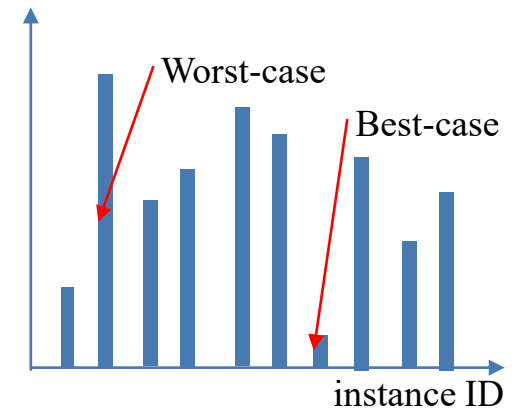
100
80
60
40
20
0

— A
— B

1 2 3 4 5 6 7 8 9 10

Our Roadmap

◈ Programming basics

◈ How to solve a problem? by an algorithm?

◈ How to analyze the running time of an algorithm?

◈ What are data structures?

Why do we analyze the running time of an algorithm?

- The same problem (e.g., sorting) can be solved by several different algorithms

- It is time consuming to convert all algorithms to programs

- We just want to choose the "fastest" algorithm and then convert it into a program

- Understand the scalability of a program.

  E.g., if the input size doubles, how much would its running time increase?

How to analyze the running time of an algorithm?

- Assumptions
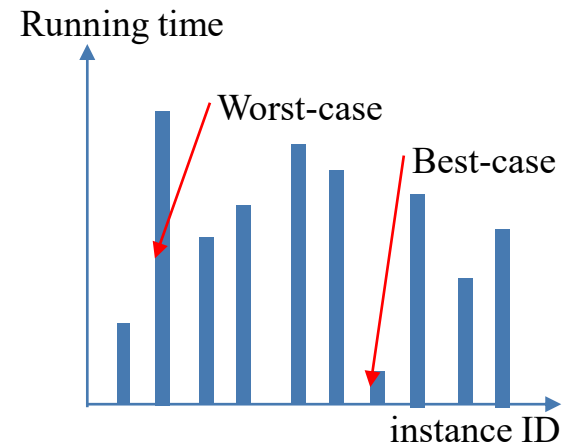 - #1. Each basic step takes constant time
    - E.g., add two integers: c_{add}
      compare two numbers: $c_{compare}$
      copy a value to a variable: c_{copy}
  - #2. The memory is large enough to store all input/intermediate/output data
- Find the (worst-case) frequency and cost of executing each line, then sum up the total cost

Running time

Worst-case

Best-case

instance ID

How to estimate running time?

Search (Array $A[0..n-1]$, Key k)

	cost	frequency
1. for integer $i \leftarrow 0$ to $n-1$	c_1	at most n
2. if $A[i] = k$	c_2	at most n
3. return i	c_3	at most 1
4. return -1	c_4	at most 1

- What is the frequency of executing Line 1?
  - It depends on which element $A[i]$ is equal to k
  - Line 1 is executed **at most** n times
- What is the frequency of executing Line 2?
- ……
- <u>Worst-case</u> running time of the algorithm

$= c_1 * n + c_2 * n + c_3 * 1 + c_4 * 1$

$= (c_1 + c_2) n + c_3 + c_4$

> - This expression contains many constants
> - How to simplify it?

Asymptotic Notation

◈ How to describe the trend of running time in a concise way?

◈ Asymptotic **upper-bound**, **O**-notation

   ◈ We can write $f(n) = O(g(n))$ if

   we can find positive constants c, n_0 such that

   $0 \leq f(n) \leq c\, g(n)$ for all $n \geq n_0$

◈ Is $(5n^2 + 3n) = O(n^2)$?

   $5n^2 + 3n$
   $\leq 5n^2 + 3n^2$ [true for all $n \geq 1$]
   $\leq 8n^2$

   ◈ YES, we choose $c=8$ and $n_0=1$.

   [*Note*: other proofs/choices are also possible]

◈ Is $(5n^2 + 3n) = O(n^3)$?

   ◈ YES, *but we prefer to use a tighter expression*

$cg(n)$

$f(n)$

n_0 $f(n) = O(g(n))$
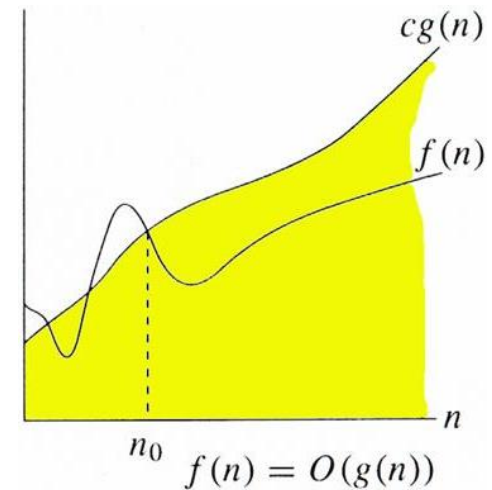
Proof

◈ $f(n) = (5n^2 + 3n)$ and $g(n) = n^2$.

◈ To find $(5n^2 + 3n) = O(n^2)$, show $f(n) \leq c\, g(n)$ for all $n \geq n_0$.

◈ We need to find c and n_0.

◈ Since the coefficient of the highest order of n is 5, the minimal c is 6 if there is a valid n_0.

◈ Find the tight n, e.g., n_0,

such that $(5n^2 + 3n) \leq 6n^2$, i.e., $3n \leq n^2$.

◈ For n $\geq n_0 = 3$, we have $3n \leq n^2$.

So, the order of $(5n^2 + 3n)$ is $O(n^2)$

```
> n = n0 = 3
>   c = 6
> 5*n^2 + 3*n
[1] 54
> 6*n^2
[1] 54
> 5*n^2 + 3*n <= 6*n^2
[1] TRUE
> n = n0 = 2
> 5*n^2 + 3*n <= 6*n^2
[1] FALSE
> n = n0 = 4
> 5*n^2 + 3*n <= 6*n^2
[1] TRUE
```

Simplification Rules of O-Notation

◈ Ignore constant factors

    ◈ E.g., $\quad 3 = O(1)$

    ◈ E.g., $\quad 3n = O(n)$

◈ Combine fixed number of terms with same complexity

    $c_1\, n + c_2\, n = (c_1 + c_2)\, n$

    ◈ E.g., $\quad O(n) + O(n) = O(n)$

    ◈ E.g., $\quad O(n) + O(n) + O(n) = O(n)$

◈ Polynomial

    ◈ Just use the highest-degree term.   Why?

    ◈ E.g., $\quad 5n^2 + 3n = O(n^2)$

    ◈ E.g., $\quad 2n^3 + 5n^2 + 3n = O(n^3)$

Asymptotic running time of some algorithms

Complexity		*Algorithm*
O(1)	Constant time	Compare two numbers
O(log n)	Logarithmic	Binary search (on a sorted array)
O(n)	Linear time	Linear search (on an unsorted array)
O(n log n)		Merge-sort
O(n^2)	Quadratic	Selection-sort

◈ The trend matters

  ◈ We care about the performance at *large input size*

  ◈ Constant factors (e.g., *c*) do not affect *the order of growth*
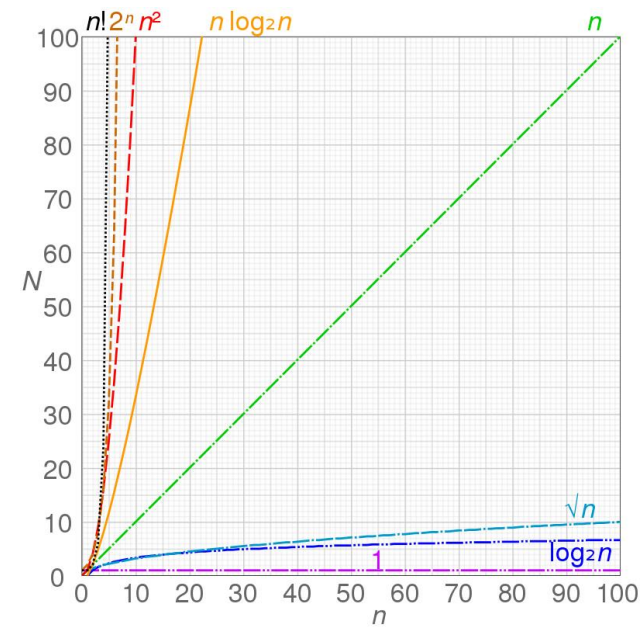
Asymptotic running time: Example 1

Search (Array $A[0..n-1]$, Key k)

   1. for integer $i \leftarrow 0$ to $n-1$

   2.   if $A[i] = k$

   3.   return i

   4. return -1

| 5 | 2 | 4 | 9 | 7 |

◈ Let's analyze the asymptotic running time of this algorithm

   ◈ Line 1:   at most n basic steps $= \mathrm{O}(n)$

   ◈ Line 2:   at most $1*n$ basic steps $= \mathrm{O}(n)$

   ◈ Lines 3 and 4: $\mathrm{O}(1)$

◈ Running time of algorithm $\mathrm{T}(n)$

$= \mathrm{O}(n) + \mathrm{O}(n) + \mathrm{O}(1)$

$= \mathrm{O}(n)$

Asymptotic running time: Example 2

Selection-Sort (Array $A[0..n–1]$)
1. for integer $i \leftarrow 0$ to $n–2$
2. $k \leftarrow i$
3.   for integer $j \leftarrow i+1$ to $n–1$
4.   if $A[k] > A[j]$ then
5. $k \leftarrow j$
6.   swap $A[i]$ and $A[k]$

$$5 \mid 2 \mid 4 \mid 9 \mid 7$$

- Line 1: $n – 1$ basic steps $= O(n)$
- Line 2: $1*(n – 1)$ basic steps $= O(n)$
- Lines 3 to 5:
  - How many times can we execute Line 3 at most?
  - For each given value of i, what is the possible range of j?

27

Asymptotic running time: Example 2

- ◈ Worst-case analysis of Line 3:
  - ◈ Values of j in that loop: 1 to n–1, 2 to n–1, …, n–1 to n–1
  - ◈ Number of iterations: $n - 1, n - 2, n - 3, …, 1$

$$\Sigma_{i=1..n-1}\ (n-i) = (n - 1) + (n - 2)+ …+ 1$$
$$= (n-1)*n/2 = O(n^2)$$

- ◈ Running time of algorithm T(n) :

Formula:

$$\sum_{i=1}^{n} i = 1 + 2+…+n = \frac{n(n + 1)}{2}$$

  - ◈ Line 1. $O(n)$
  - ◈ Lines 2, 6.   same as Line 1
  - ◈ Line 3. $O(n^2)$
  - ◈ Lines 4, 5.   same as Line 3
  - ◈ Thus, we have: $\mathbf{T(n)} = 3*O(n) + 3*O(n^2) = O(n^2)$

Our Roadmap

◈ Programming basics

◈ How to solve a problem? by an algorithm?

◈ How to analyze the running time of an algorithm?

◈ What are data structures?

Data Structures

◈ What are the data structures for human?

  ◈ Used in libraries, books, clinics, companies, ……

Oriental lampshades, 105-107
Patterns, how to make, 87, 135, 137
Piping, 120
Pleating, 99-104
Pricing your work, 152
Relining lampshades, 118
Rewiring lamps, 80-82
Roses, 126
Ruffles, how to make, 122-123
Scallops, 33, 85
Shampooing lampshades, 151
Shapes of lampshades, 31-41
Silhouettes of lamps, 21-30
Slipcovers for lamps, 108
Smocking, 96, 99
Spiders, different kinds, 31-32
Sunburst pleating, 100-102

◈ How about the data structures for computers?

Data Objects

- Examples of data objects
  - <u>Employees</u> in payroll application
  - <u>Tasks</u> in the scheduler of OS
  - <u>Shopping cart items</u> in online shopping
  - In any application that you can think of ……
- A data object has attributes
  - E.g., the attributes of shopping cart items are:
    `product_id, product_name, price, ……`
- The search key depends on the application, e.g.,
  - `product_id` for a shopping cart application

Data Structures

- Let *S* be a set of items, and *x* be a search key
  - A key is a number, e.g., product id

set

- Useful operations on a set *S*
  - Search(*S*, *x*): search whether *x* appears in *S*
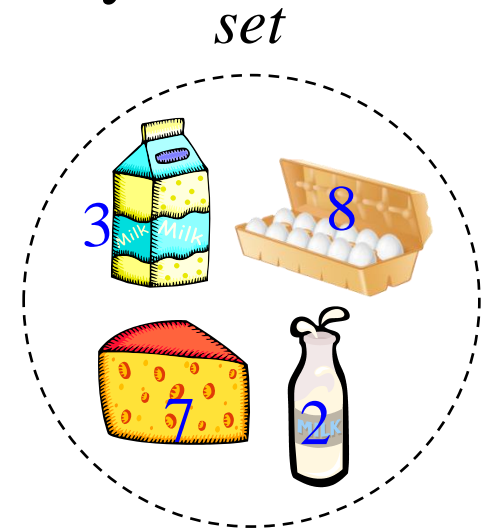 - Insert(*S*, *x*): insert item *x* into *S*
  - Delete(*S*, *x*): remove item *x* from *S*

search key

- *Data structure*:
  - A way of organizing data objects for efficient usage
  - Building blocks for designing algorithms

The problems in Lectures 2-5

- How to solve these problems quickly?
  - *Sort* a set of items
  - *Search* an item from a set of items
  - *Insert* / *Delete* an item from a set of items
- Question 1
  - Selection-Sort takes O(n^2) time
  - Is there a faster sorting algorithm?
- Question 2
  - Is an array good enough for searching, insertion and deletion?

Is Array Good Enough?

◈ Search a key in an unsorted array (of length n)

◈ Linear search algorithm

  ◈ Running time: $O(n)$

◈ Example #1: search k=21

  ◈ Check the case i=0

  ◈ Return **0**

◈ Example #2: search k=17

  ◈ Check the cases i=0,1,2,3,4,5,6

  ◈ Return **6**

◈ How to search faster?

$$21 \mid 12 \mid 8 \mid 3 \mid 35 \mid 1 \mid 17$$

Search (Array $A[0..n\text{-}1]$, Key k)

1. for integer $i \leftarrow 0$ to $n–1$

2.   if $A[i] = k$

3.   return i

4. return $–1$

Is Array Good Enough?

- Search in a **sorted** array (of length n)
- Binary search algorithm
  - Running time: O(log n)
- *Idea*:
  - Compare k with the middle item $A[mid]$
  - Is k in the left sub-array or in the right sub-array?

$$\boxed{1 \mid 3 \mid 8 \mid 12 \mid 17 \mid 23 \mid 35}$$

BinarySearch (Array A, Integers *low*, *high*, Key k)

1. $mid \leftarrow \lfloor (low+high)/2 \rfloor$
2. if $A[mid] = k$
3.   return mid
4. else if $low = high$
5.   return -1
6. if $k < A[mid]$
7.   return BinarySearch(A, *low*, *mid*–1, k)
8. else
9.   return BinarySearch(A, *mid*+1, *high*, k)

Initial call: BinarySearch (A, 0, n–1, k)

Example: find the key "35" from this array A

$$1 \mid 3 \mid 8 \mid 12 \mid 17 \mid 23 \mid 35$$

BinarySearch (Array A, Integers low, $high$, Key k)
1. $mid \leftarrow \lfloor (low+high)/2 \rfloor$
2. if $A[mid] = k$
3.   return mid
4. else if $low = high$
5.   return -1
6. if $k < A[mid]$
7.   return BinarySearch(A, low, $mid-1$, k)
8. else
9.   return BinarySearch(A, $mid+1$, $high$, k)

Initial call: BinarySearch (A, 0, $n-1$, k)

BinarySearch (A, 0, 6, **35**),
$mid = 3$

$$1 \mid 3 \mid 8 \mid 12 \mid 17 \mid 23 \mid 35$$

BinarySearch (A, 4, 6, **35**),
$mid = 5$

$$1 \mid 3 \mid 8 \mid 12 \mid 17 \mid 23 \mid 35$$

BinarySearch (A, 6, 6, **35**),
$mid = 6$

$$1 \mid 3 \mid 8 \mid 12 \mid 17 \mid 23 \mid 35$$

Key found, return 6

36

```java
int i = 0;  // for understanding recursive function only
public int binarySearch(int[] A, int low, int high, int k){
    //int mid = (int) Math.floor( ((double) low + (double) high) / 2.0);
    i = i + 1;
    System.out.println("call " + i); // tracking
    int mid = (low + high) / 2;
    System.out.println("mid position is  " + mid); // tracking
    System.out.println("value  is  " + A[mid]);  // tracking
    if(A[mid] == k) return mid;
    else if (low == high) return -1;
    if(k < A[mid]) return binarySearch(A, low, mid-1, k);
    else return binarySearch(A, mid+1, high, k);
    }
```

```
int[] A = {1,3,8,12,17,23,35};
binarySearch(A, 0, A.length - 1, 35);
```

```
call 1
mid position is  3
value  is  12
call 2
mid position is  5
value  is  23
call 3
mid position is  6
value  is  35

6
```

Is Array Good Enough?

◈ How about insertion/deletion?

◈ Can an unsorted array support fast insertion/deletion?

| 21 | 12 | 8 | 3 | 35 | 1 | 17 |

◈ Can a sorted array support fast insertion/deletion?

| 1 | 3 | 8 | 12 | 17 | 23 | 35 |

Data Structures in Lectures 2-5

- Learn the **properties** of data structures
  - E.g., stack's LIFO property, queue's FIFO property, growable structure like linked list
- Learn the **operations** of data structures
  - They support different operations, and with different running time
- You may play with them in *https://visualgo.net/en*
- What's the most suitable data structure to solve a particular problem?
  - It depends on the properties and operations most frequently used in your algorithm

Stack (*Lecture 2*)
Queue (*Lecture 2*)
Linked List (*Lecture 2*)
Tree (*Lecture 3*)
Heap (*Lecture 4*)
......

Java Programmers

◈ At least know how to use the data structures and algorithms in the `java.util` library
(*https://docs.oracle.com/javase/8/docs/api/java/util/package-summary.html*)

  ◈ `Stack, Queue, LinkedList, PriorityQueue, ……`
  ◈ `Arrays.sort,  Arrays.binarySearch, ……`

◈ We hope you can also understand the "magic" behind them

Summary

◈ Basic concepts in Java, algorithm, asymptotic running time, data structure

◈ Please read Chapters 1, 4.2.1-4.2.6 in the book "*Data Structures and Algorithms in Java*"

◈ *Next lecture*:

linear data structures (stack, queue, linked list)