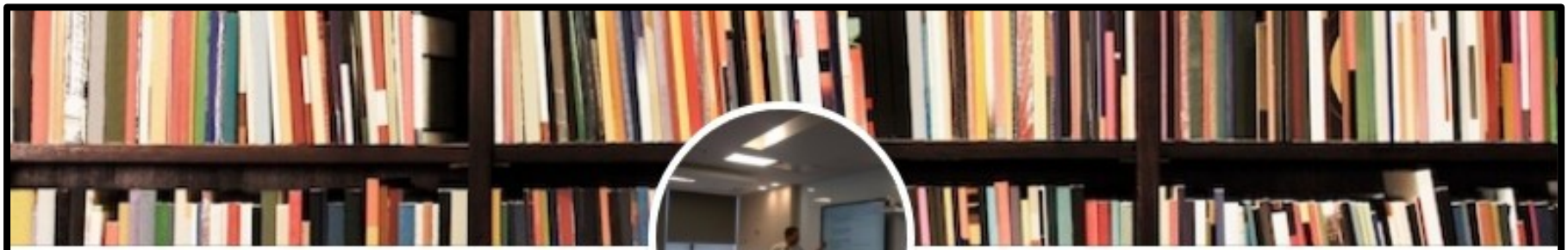
Data Structures with Java

Rajeev Gupta
Rgupta.mtech@gmail.com
Java Trainer & consultant



Rajeev Gupta

FreeLancer Corporate Java JEE/ Spring Trainer

freelance • Institution of Electronics and Telecommunication Engineers IETE

New Delhi Area, India • 500+

---

1. Expert trainer for Java 8, GOF Design patterns, OOAD, JEE 7,Spring 5, Hibernate 5 ,Spring boot, microservice, netflix oss, Spring cloud, angularjs, Spring MVC, Spring Data, Spring Security, EJB 3, JPA 2, JMS 2, Struts 1/2, Web service

2. Helping technology organizations by training their fresh and senior engineers in key technologies and processes.

3. Taught graduate and post graduate academic courses to students of professional degrees.

I am open for advance java training /consultancy/ content development/ guest lectures/online training for corporate / institutions on freelance basis

Data Structures with Java Part I

- Introduction to data structure and algorithm
- Basic searching and sorting algorithms
- Stacks and queues

Introduction to data structure and algorithm

What is data structure?

Data

A collection of facts from which conclusion may be drawn

e.g. Data: Temperature 35°C;Conclusion: It is hot.

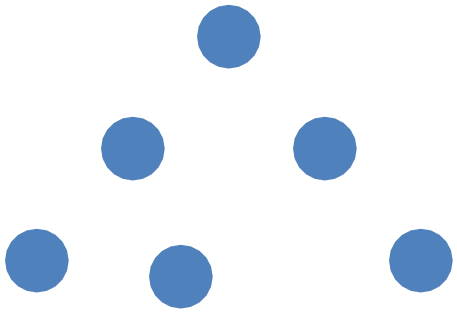
data structure

- A particular way of storing and organizing data in a computer so that it can be used efficiently and effectively.
- Data structure is the logical or mathematical model of a particular organization of data.
- A group of data elements grouped together under one name.
  - For example, an array of integers

Types of data structures

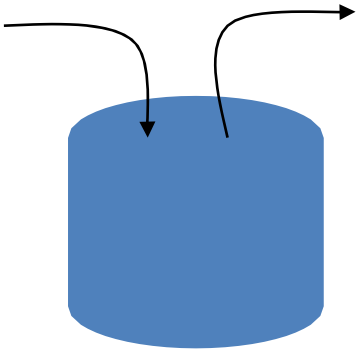
Array

Linked List



Tree

Queue



Stack

There are many, but we named a few. We'll learn these data structures in great detail!

The Need for Data Structures

- Goal: to **organize data**

- Criteria: to facilitate **efficient**
  - **storage** of data
  - **retrieval** of data
  - **manipulation** of data

- Design Issue:
  - **select and design** appropriate data types
    (This is the main motivation to learn and understand data
    structures)

Data Structure Operations

- **Traversing**
  - Accessing each data element exactly once so that certain items in
  - the data may be processed
- **Searching**
  - Finding the location of the data element (key) in the structure
- **Insertion**
  - Adding a new data element to the structure
- **Deletion**
  - Removing a data element from the structure
- **Sorting**
  - Arrange the data elements in a logical order (ascending/descending)
- **Merging**
  - Combining data elements from two or more data structures into one

What is algorithm?

- A finite set of instructions which accomplish a particular task
- A method or process to solve a problem
- Transforms input of a problem to output

Algorithm = Input + Process + Output

Algorithm development is an art – it needs practice, practice and only practice!

What is a good algorithm?

- It must be correct
- It must be finite (in terms of time and size)
- It must terminate
- It must be unambiguous
  - Which step is next?
- It must be space and time efficient

A program is an instance of an algorithm, written in some specific programming language

A simple algorithm

- Problem: Find maximum of `a, b, c`
- Algorithm
  - Input = `a, b, c`
  - Output = `max`
  - Process
    - `Let max = a`
    - `If b > max then`
      `max = b`
    - `If c > max then`
      `max = c`
    - `Display max`

Algorithm development: Basics

- Clearly identify:
  - what output is required?
  - what is the input?
- What steps are required to transform input into output
  - The most crucial bit
  - Needs problem solving skills
  - A problem can be solved in many different ways
  - Which solution, amongst the different possible solutions is optimal?

Algorithm Analysis: Motivation

- A problem can be solved in many different ways
  - Single problem, many algorithms

  Which of the several algorithms should I choose?

  - We use algorithm analysis to answer this question

- Writing a working program is not good enough
- The program may be inefficient!
- If the program runs on a large data set, then the running time becomes an issue

What is algorithm analysis?

- A methodology to predict the resources that the algorithm requires
  - Computer memory
  - Computational time
- We'll focus on computational time
  - It does not mean memory is not important
  - Generally, there is a trade-off between the two factors
    - Space-time trade-off is a common term

Counting Primitive Operations

- Total number of primitive operations executed
  - is the running time of an algorithms
  - is a function of the input size
- Example

```
Algorithm ArrayMax(A, n)                    # operations
currentMax ←A[0]                            2: (1 +1)
for i←1;i<n; i←i+1 do                       3n-1: (1 + n+2(n- 1))
      if A[i]>currentMax then               2(n − 1)
            currentMax ←A[i]                2(n − 1)
      endif  endfor
      return currentMax
                                            1
                                  Total:    7n − 2
```

Algorithm efficiency: growth rate

- An algorithm's time requirements can be expressed as a function of (problem) input size
- Problem size depends on the particular problem:

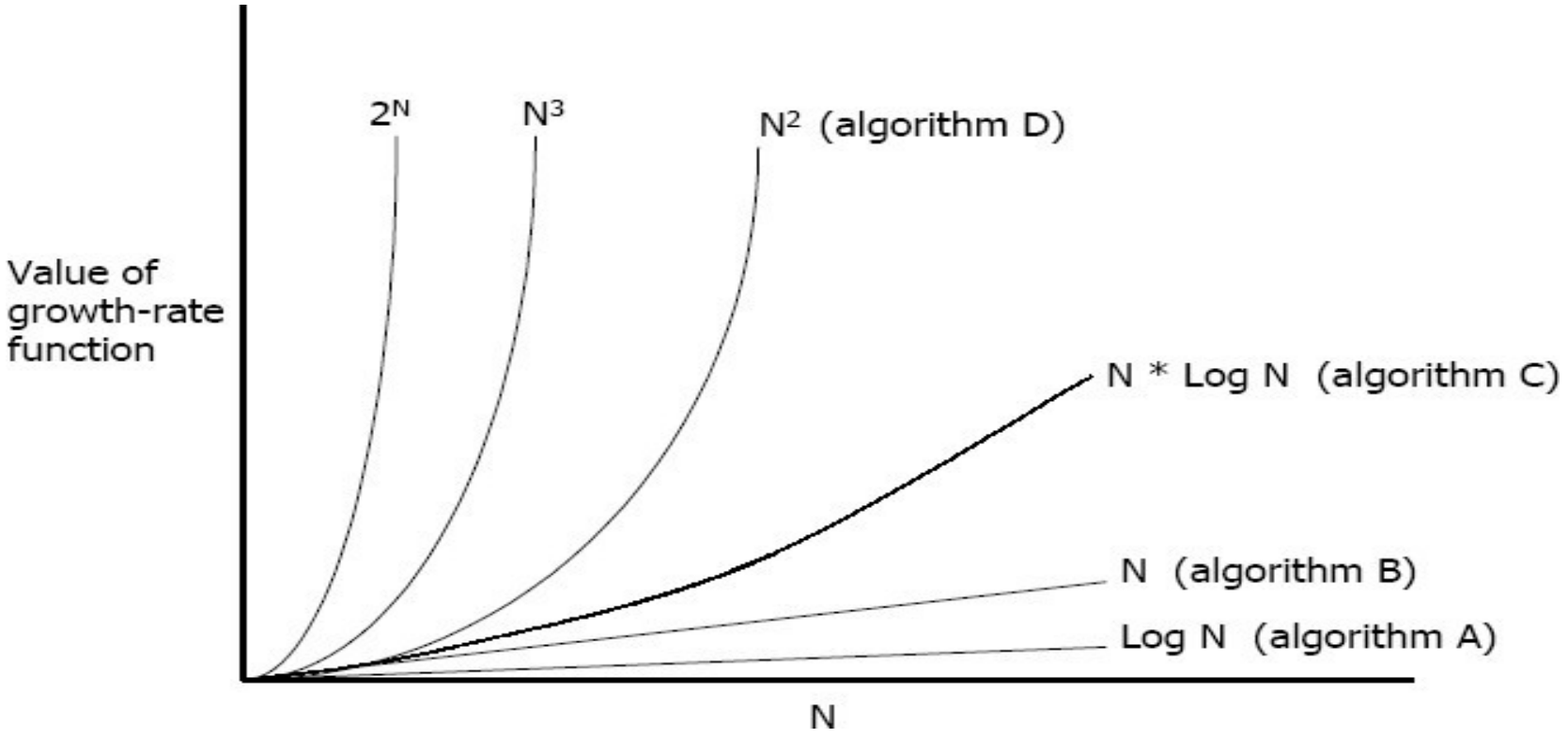
- How quickly the time of an algorithm grows as a function of problem size -- this is often called an algorithm's growthrate

For a search problem, the problem size is the number of elements in the search space

For a sorting problem, the problem size is the number of elements in the given list

Algorithm growth rate

Which algorithm is the most efficient? [The one with the growth rate Log N.]



Seven basic function

1. Constant function $f(n) = c$
2. Linear function $f(n) = n$
3. Quadratic function $f(n) = n^2$ $f(n)$
4. Cubic function $= n^3$ $f(n) =$
5. Log function $\log n$
6. Log linear function $f(n) = n \log n$
7. Exponential function $f(n) = b^n$

Algorithmic runtime

- Worst-case running time
  - measures the maximum number of primitive operations executed
  - The worst case can occur fairly often
    - e.g. in searching a database for a particular piece of information
- Best-case running time
  - measures the minimum number of primitive operations executed
    - Finding a value in a list, where the value is at the first position
    - Sorting a list of values, where values are already in desired order
- Average-case running time
  - the efficiency averaged on all possible inputs
  - maybe difficult to define what “average” means

Big-O and function growth rate

- We use a convention O-notation (also called Big-Oh) to represent different complexity classes
- The statement "*f(n)* is *O(g(n))*" means that the growth rate of *f(n)* is no more than the growth rate of *g(n)*
- *g(n)* is an upper bound on *f(n)*, i.e. maximum number of primitive operations
- We can use the big-O notation to rank functions according to their growth rate

Big-O: functions ranking

BETTER

▲

- $O(1)$ constant time
- $O(\log n)$ log time
- $O(n)$ linear time
- $O(n \log n)$ log linear time
- $O(n^2)$ quadratic time
- $O(n^3)$ cubic time
- $O(2^n)$ exponential time

▼

WORSE

Big O examples

- f(n) = 7n – 2; 7n - 2 is O(n)

  ? Find c > 0 and n0 ≥ 1 such that $7n-2 \leq c \bullet n$ for $n \geq n_0$

  This is true for c = 7 and n0 = 1

  at n = 1, $7-2 \leq 7$; at n = 2, $14 - 2 \leq 14$, and so on

- f(n) = 3 log n + 5; 3 log n + 5 is O(log n)

  ? Find c > 0 and $n_0 \geq 1$ such that $3 \log n + 5 \leq c \bullet \log n$ for $n \geq n_0$

  This is true for c = 8 and $n_0 = 2$

Arrays & Searching Algorithms

Array Operations

- **Indexing**: inspect or update an element using its index. Performance is very fast O(1)

```
randomNumber = numbers[5];  numbers[20000] = 100;
```

- **Insertion**: add an element at certain index
  - Start: very slow O(n) because of shift
  - End : very fast O(1) because no need to shift
- **Removal**: remove an element at certain index
  - Start: very slow O(n) because of shift
  - End : very fast O(1) because no need to shift
- **Search**: performance depends on algorithm

  1) Linear: slow O(n) 2) binary : O(log n)
- **Sort**: performance depends on algorithm

  1) Bubble: slow $O(n^2)$ 2) Selection: slow $O(n^2)$

  3) Insertion: slow $O(n^2)$ 4)Merge : O (n log n)

Searching Algorithms

Index:	0	1	2	3	4
Value:	20	40	10	30	60

Target = 30 (success or failure?)

Target = 45 (success or failure?)

Search strategy?

List Size = N = 5

Min index = 0

Max index = 4 (N - 1)

Sequential Search

- Search in a sequential order
- Termination condition
  - Target is found (success)
  - List of elements is exhausted (failure)

```
public class LinearSearch {

    public static void main(String[] args) {
        int arr[]= {3,5,6,-3,78,22};
        int pos=-1;
        int val=60;
        boolean isFound=false;
        for(int i=0;i<arr.length; i++) {
            if(arr[i]==val) {
                isFound=true;
                pos=i;
                break;
            }
        }
        if(isFound)
            System.out.println("found: "+ pos);
        else
            System.out.println("not found");
    }
}
```

Binary Search

- Search through a sorted list of items
  - Sorted list is a pre-condition for Binary Search!
- Repeatedly divides the search space (list) into two
- Divide-and-conquer approach

```
5  public class BS {
6      public static void main(String[] args) {
7          int arr[]= {3,5,6,-3,78,22};
8          Arrays.sort(arr);// first sort   //-3 3 5 6 22 78
9          BS.binarySearch(arr, 0, arr.length, 6);
10
11     }
12     public static void binarySearch(int arr[], int first, int last, int key) {
13         int mid = (first + last) / 2;
14         while (first <= last) {
15             if (arr[mid] < key) {
16                 first = mid + 1;
17             } else if (arr[mid] == key) {
18                 System.out.println("Element is found at index: " + mid);
19                 break;
20             } else {
21                 last = mid - 1;
22             }
23             mid = (first + last) / 2;
24         }
25         if (first > last) {
26             System.out.println("Element is not found!");
27         }
28     }
29 }
30
```

Search Algorithms:Time Complexity

- Time complexity of Sequential Search algorithm:
  - Best-case : O(1) comparison
    - target is found immediately at the first location
  - Worst-case: O(n) comparisons
    - Target is not found
  - Average-case: O(n) comparisons
    - Target is found somewhere in the middle
- Time complexity of Binary Search algorithm:

  $O(\log(n))$ ➔ This is worst-case

Sorting Algorithms

1. Bubble Sort
2. Selection Sort
3. Insertion Sort
4. Quick Sort
5. Merge Sort

Bubble Sort Algorithm: Informal

- Repeatedly compare the elements at consecutive locations in a given list, and do the following until the elements are in required order:
  - If elements are not in the required order, swap them (change their position)
  - Otherwise do nothing

Bubble Sort Algorithm in Java

```java
public class Bubble {
    public static void main(String[] args) {
        int a[]= {38,52,9,18, 6,62,13};
        int temp;
        for(int i=0;i<a.length; i++)
        {
            for(int j=0;j<a.length-1; j++)
            {
                if(a[j]> a[j+1])
                {
                    temp=a[j];
                    a[j]=a[j+1];
                    a[j+1]=temp;
                }
            }
        }
        for(int val: a) {
            System.out.print(val+" ");
        }
    }
}
```

Time complexity of the Bubble Sort algorithm is $O(n^2)$.

Selection Sort

- Suppose we want to sort an array in ascending order:
  - Locate the smallest element in the array; swap it with element at index 0
  - Then, locate the next smallest element in the array; swap it with element at index 1.
  - Continue until all elements are arranged in order

Selection Sort Algorithm in Java

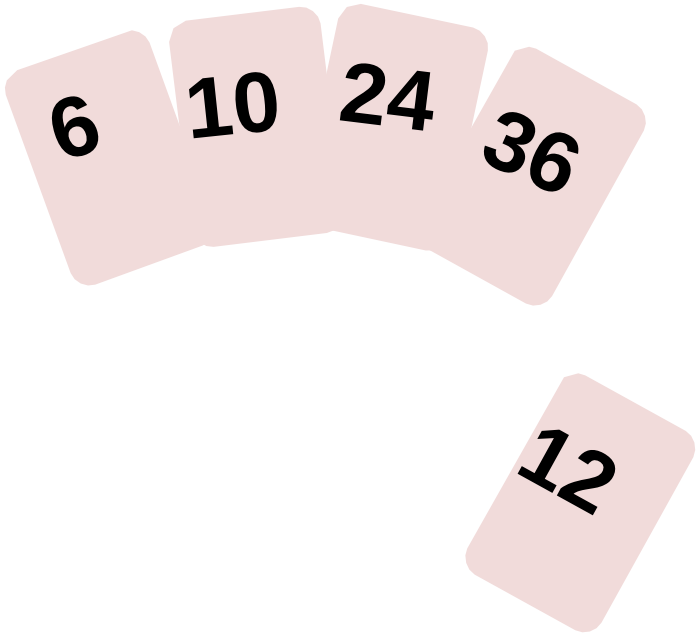
```
public static void main(String[] args) {
    int a[]= {38,52,9,18, 6,62,13};
    int min;
    int temp;
    for(int i=0;i<a.length; i++)
    {
        min=i;
        for(int j=i+1;j<a.length; j++)
        {
            if(a[j]<a[min])
                min=j;
        }
        temp=a[i];
        a[i]=a[min];
        a[min]=temp;

    }
    for(int val: a) {
        System.out.print(val+" ");
    }
}
}
```

- Time complexity of the Selection Sort algorithm is $O(n^2)$.

Bubble Sort vs. Selection Sort

- Selection Sort is more efficient than Bubble Sort, because of fewer exchanges in the former
- Both Bubble Sort and Selection Sort belong to the same (quadratic) complexity class ($O(n^2)$)

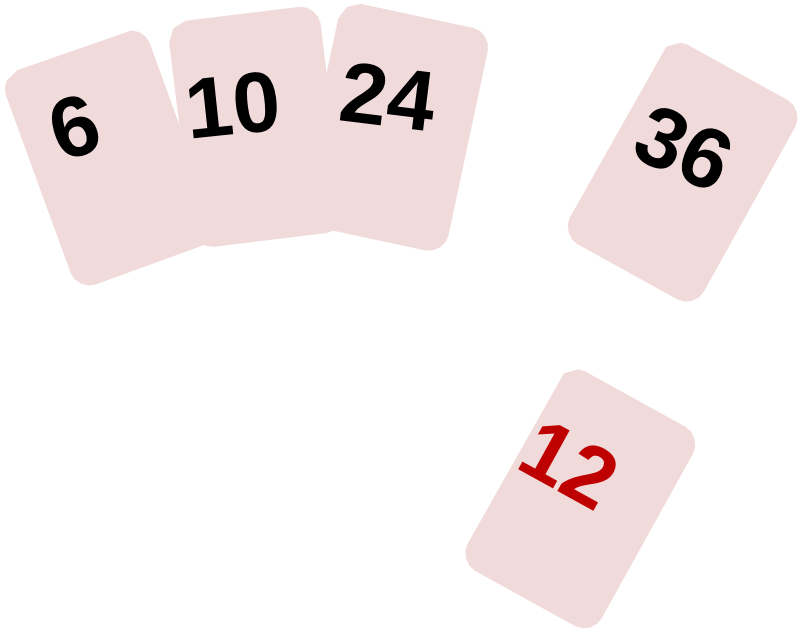
Insertion Sort



Works like someone who inserts one more card at a time into a hand of cards that are already sorted

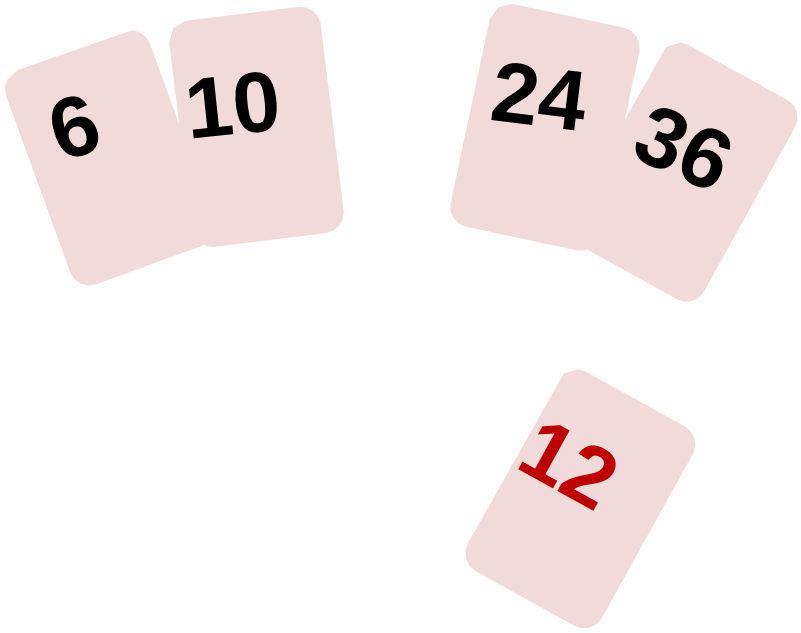
To insert 12, we need to make room for it …

Insertion Sort



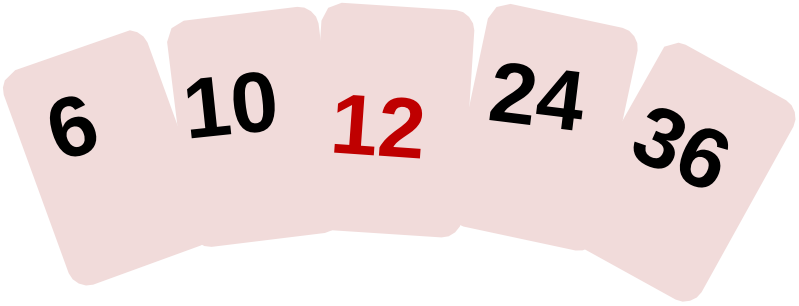
… by shifting first 36 towards right…

Insertion Sort



… and then shifting 24 towards right

Insertion Sort



Once room is available, insert the element (12 in this case)

Insertion Sort: Informal

- We divide the list into two parts: Sorted and Unsorted parts

  - Initially

    the sorted part contains the first element (at index 0)

    the unsorted part contains the elements from index 1 to N-1

  - Then, we move element from index 1 to an appropriate position in the sorted part, keeping order intact
  - Then, we move element from index 2 to an appropriate position in the sorted part, keeping order intact

    ...
  - Finally, we move element from index N-1 to an appropriate position in the sorted part, keeping order intact

Insertion Sort Algorithm in Java

```
public static void main(String[] args) {
    int a[]= {38,52,9,18, 6,62,13};
    int min;
    int temp;
    for(int i=0;i<a.length; i++)
    {
        min=i;
        for(int j=i+1;j<a.length; j++)
        {
            if(a[j]<a[min])
                min=j;
        }
        temp=a[i];
        a[i]=a[min];
        a[min]=temp;

    }
    for(int val: a) {
        System.out.print(val+" ");
    }
}
```

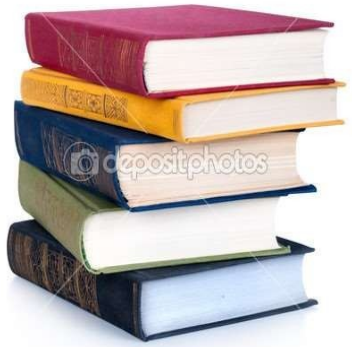
- Time complexity of the Insertion Sort algorithm is $O(n^2)$.

Stack and Queues

What is stack?

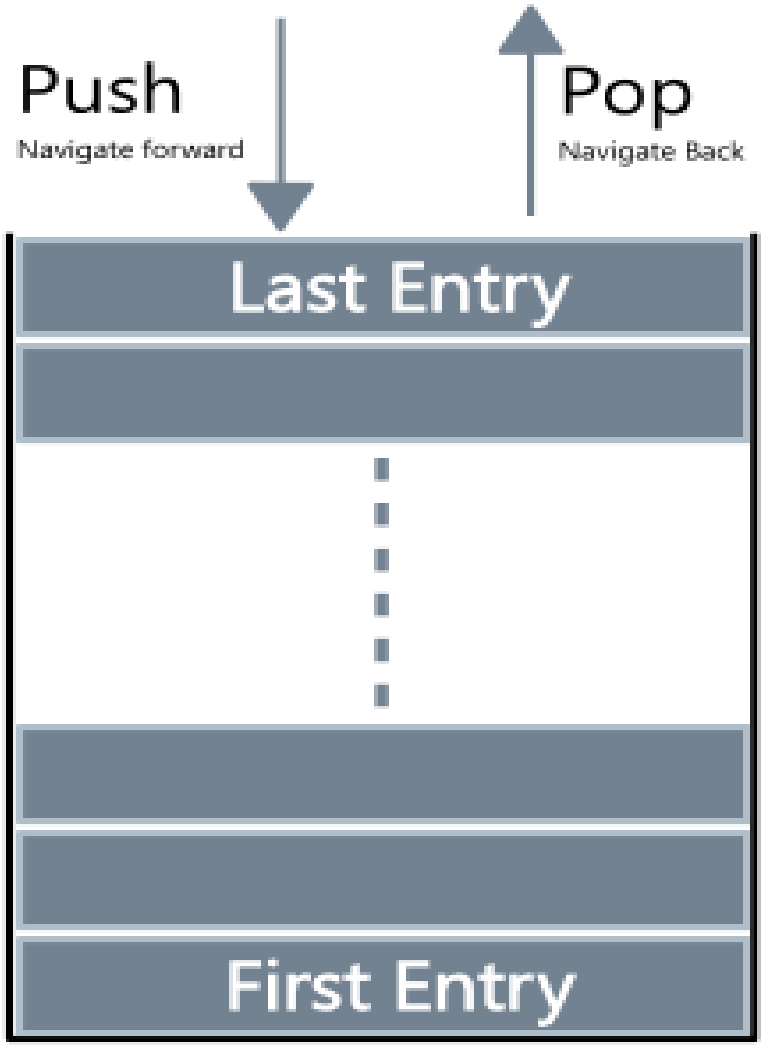
A stack is called a last-in-first-out (LIFO) collection. This means that the last thing we added (pushed) is the first thing that gets pulled (popped) off.

A stack is a sequence of items that are accessible at only one end of the sequence.



Operations that can be performed on STACK:

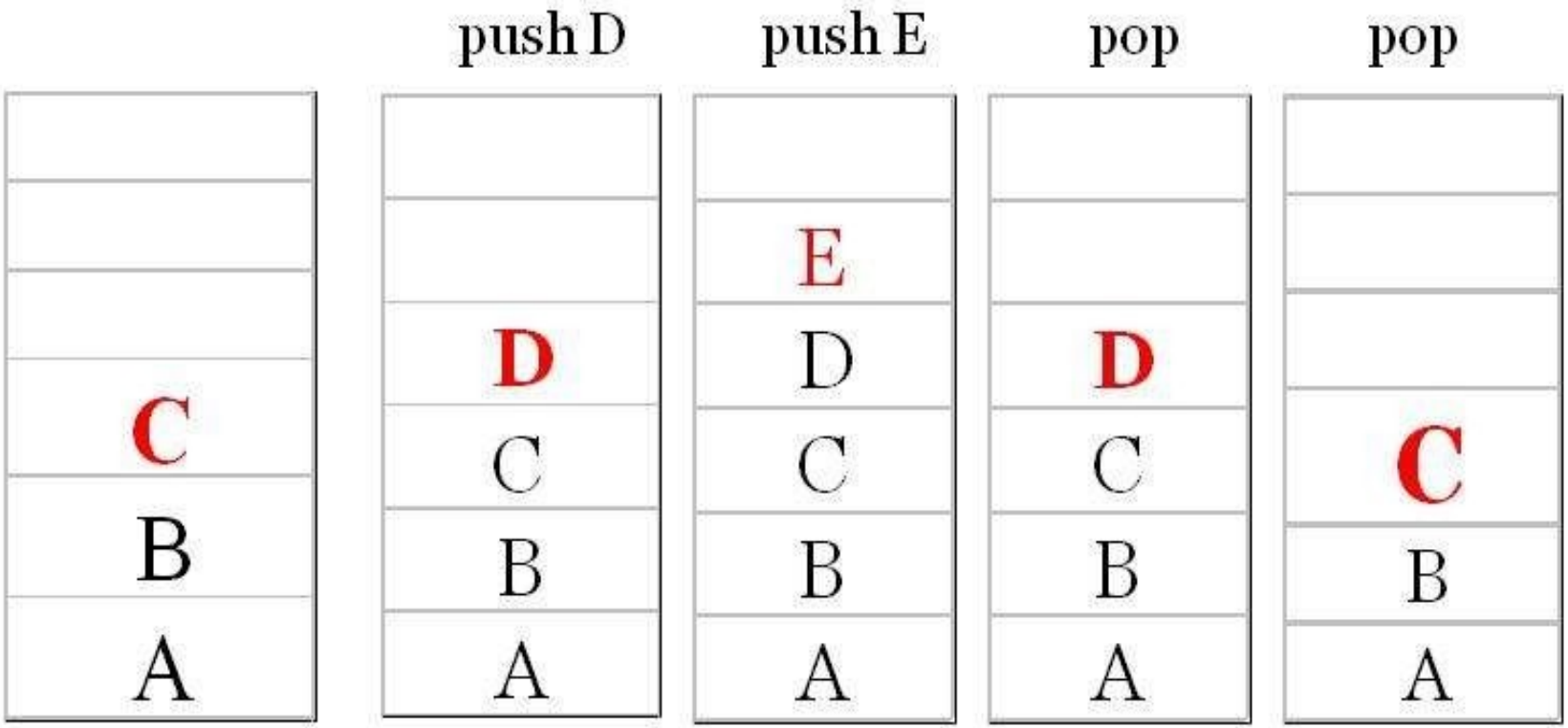
- PUSH.
- POP.



PUSH : It is used to insert items into the stack.

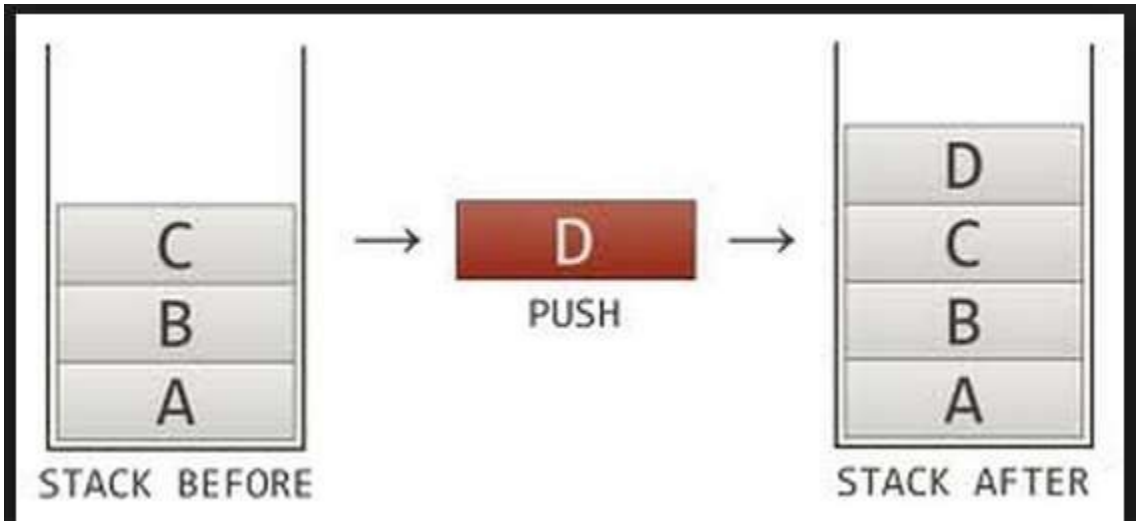
POP: It is used to delete items from stack.

TOP: It represents the current location of data in stack.



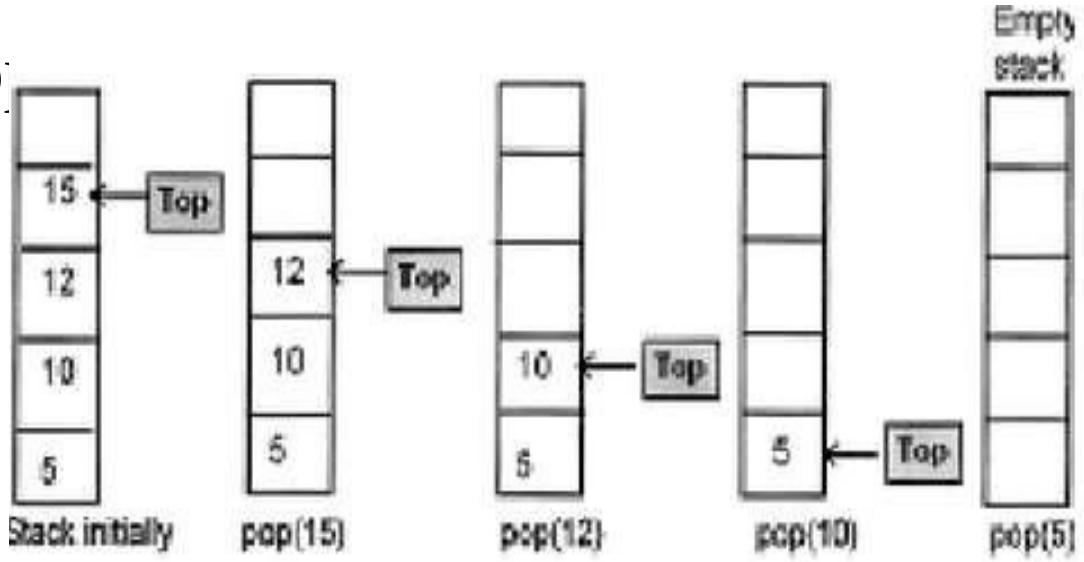
ALGORITHM OF INSERTION IN STACK: (PUSH)

1. Insertion(a,top,item,max)
2. If top=max then
   print ‘STACK OVERFLOW’
   exit else
3. top=top+1 end if
4. a[top]=item
5. Exit



ALGORITHM OF DELETION IN STACK: (POP)

1. Deletion(a,top,item)
2. If top=0 then
   print ‘STACK UNDERFLOW’
   exit else item=a[to
3. top=top-1
4. Exit



DISPLAY IN STACK:

1.Display(top,i,a[i])

2.If top=0 then

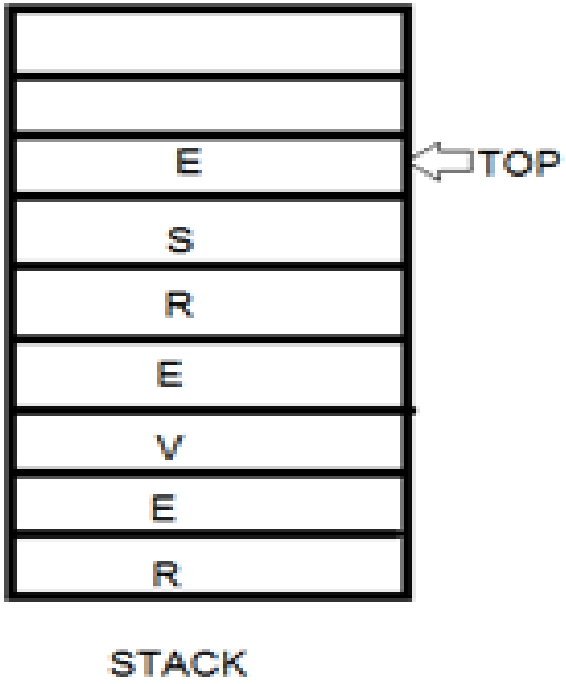
Print ‘STACK EMPTY’

Exit Else

3.For i=top to 0

Print a[i] End for

4.exit



APPLICATIONS OF STACKS ARE:

I. Reversing Strings:

- A simple application of stack is reversing strings.

To reverse a string , the characters of string are pushed onto the stack one by one as the string is read from left to right.

- Once all the characters

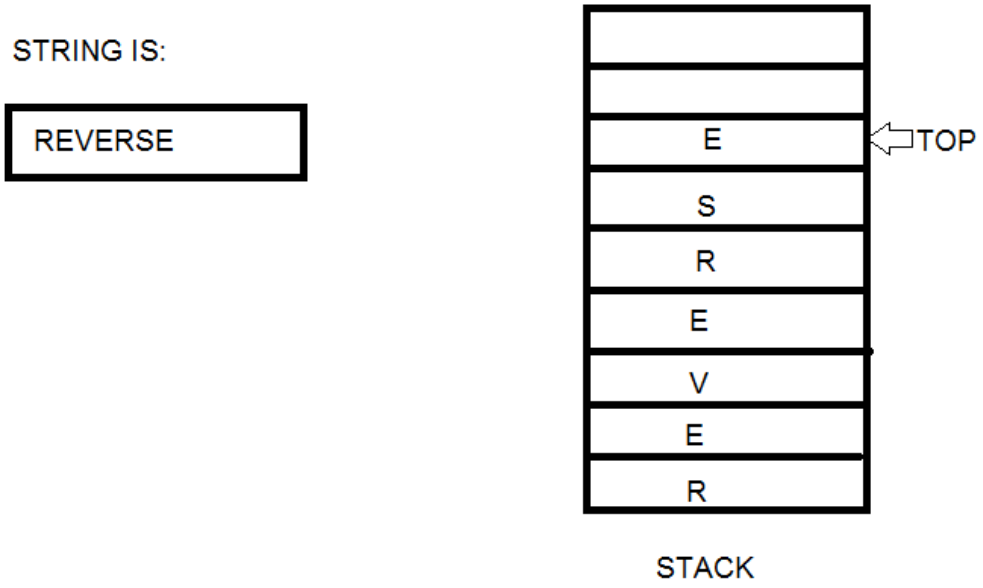
of string are pushed onto stack, they are

popped one by one. Since the character last pushed in comes out first, subsequent pop operation results in the reversal of the string.

For example:

To reverse the string ‘REVERSE’ the string is read from left to right and its characters are pushed . LIKE:

onto a stack.



II. Checking the validity of an expression containing nested parenthesis:

- Stacks are also used to check whether a given arithmetic expressions containing nested parenthesis is properly parenthesized.
- The program for checking the validity of an expression verifies that for each left parenthesis braces or bracket ,there is a corresponding closing symbol and symbols are appropriately nested.

For example:

VALID INPUTS	INVALID INPUTS
{ } ({ [ ] }) { [ ]( ) } [{ ({ } [ ] ({ })}]	{ (} ([(( )]) { } [ ]) [{) } ( ] }]

III. Evaluating arithmetic expressions:

INFIX notation:

The general way of writing arithmetic expressions is known as infix notation. e.g, (a+b)

PREFIX notation: e.g, +AB

POSTFIX notation: e.g: AB+

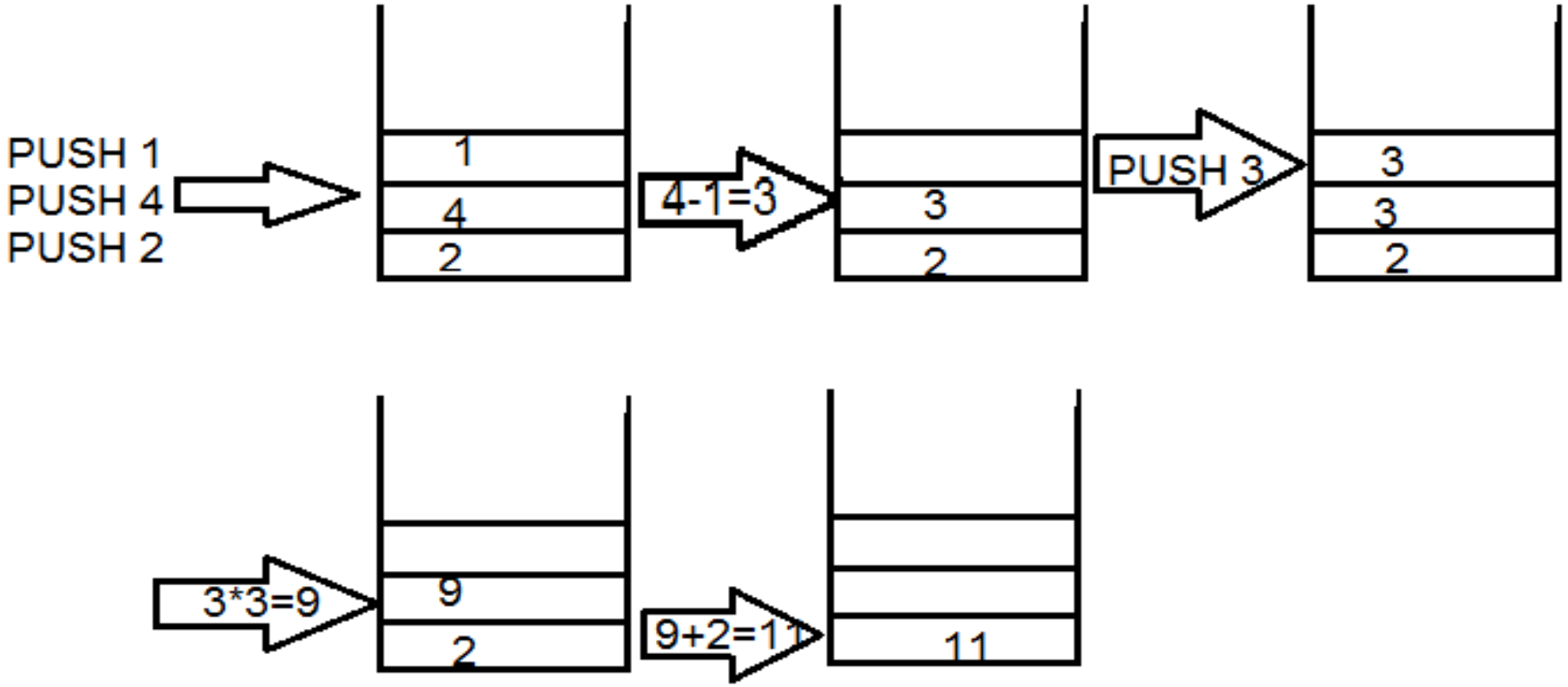
Conversion of INFIX to POSTFIX conversion:

Example: 2+(4-1)*3 step1

2+41-*3 step2

2+41-3* step3

241-3*+ step4



CONVERSION OF INFIX INTO POSTFIX

2+(4-1)*3 into 241-3*+

CURRENT SYMBOL	ACTION PERFORMED	STACK STATUS	POSTFIX EXPRESSION
(	PUSH C	C	2
2			2
+	PUSH +	(+	2
(	PUSH (	(+(	24
4			24
-	PUSH -	(+(-	241
1	POP		241-
)		**(+**	**241-**
*	PUSH *	(+*	241-
3			241-3
	POP *		241-3*
	POP +		241-3*+
)			

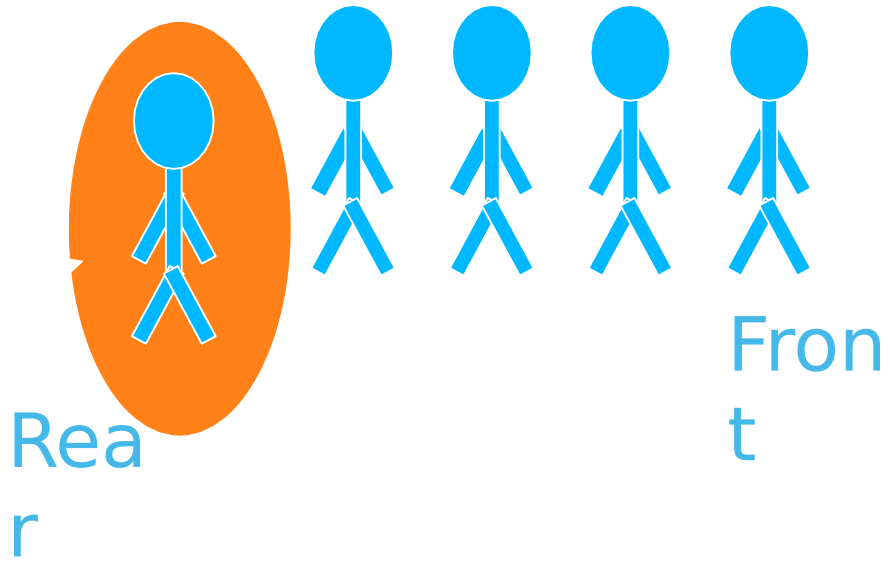
Queues

- Queue is an ADT data structure similar to stack, except that the first item to be inserted is the first one to be removed.
- This mechanism is called First-In-First-Out (FIFO).

  Placing an item in a queue is called “insertion or enqueue”, which is done at the end of the queue called “rear”.
- Removing an item from a queue is called “deletion or dequeue”, which is done at the other end of the queue called “front”.
- Some of the applications are : printer queue, keystroke queue, etc.

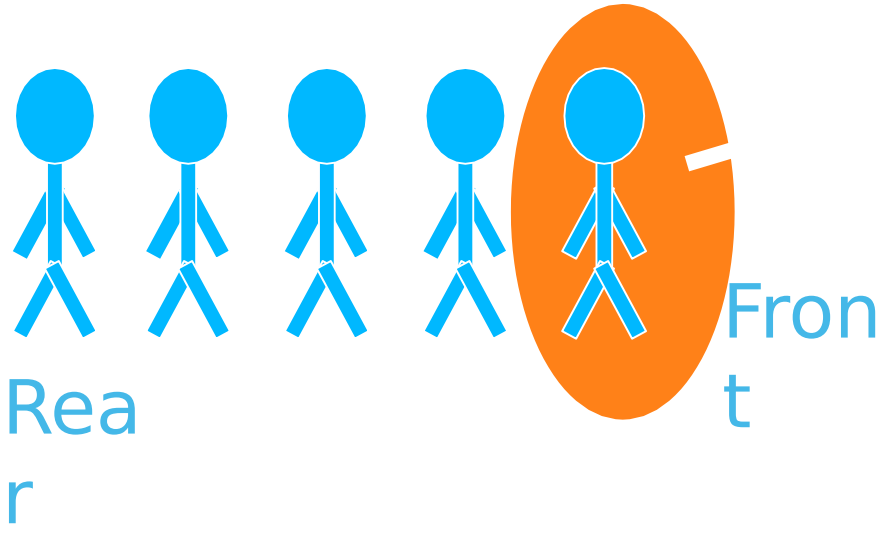
The Queue Operation

Placing an item in a queue is called “insertion or enqueue”, which is done at the end of the queue called “rear”.



The Queue Operation

Removing an item from a queue is called “deletion or dequeue”, which is done at the other end of the queue called “front”.



Algorithm Qinsert (ITEM)

1. If (rear = maxsize-1)
   print (“queue overflow”) and return
2. Else

   rear = rear + 1 Queue [rear] =

   item

Algorithm QDELETE()

1. If (front =rear)
   print “queue empty” and return
2. Else
   Front = front + 1 item =
   queue [front]; Return item

Queue Application

- Real life examples
  - ✓Waiting in line
  - ✓Waiting on hold for tech support
- Applications related to Computer Science
  - ✓Round robin scheduling
  - ✓Job scheduling (FIFO Scheduling)
  - ✓Key board buffer

3 states of the queuet

1. Queue is empty
   FRONT=REAR
2. Queue is full **REAR=N**
3. Queue contains element >=1
   FRONT<REAR
   NO. OF ELEMENT=REAR-FRONT+1

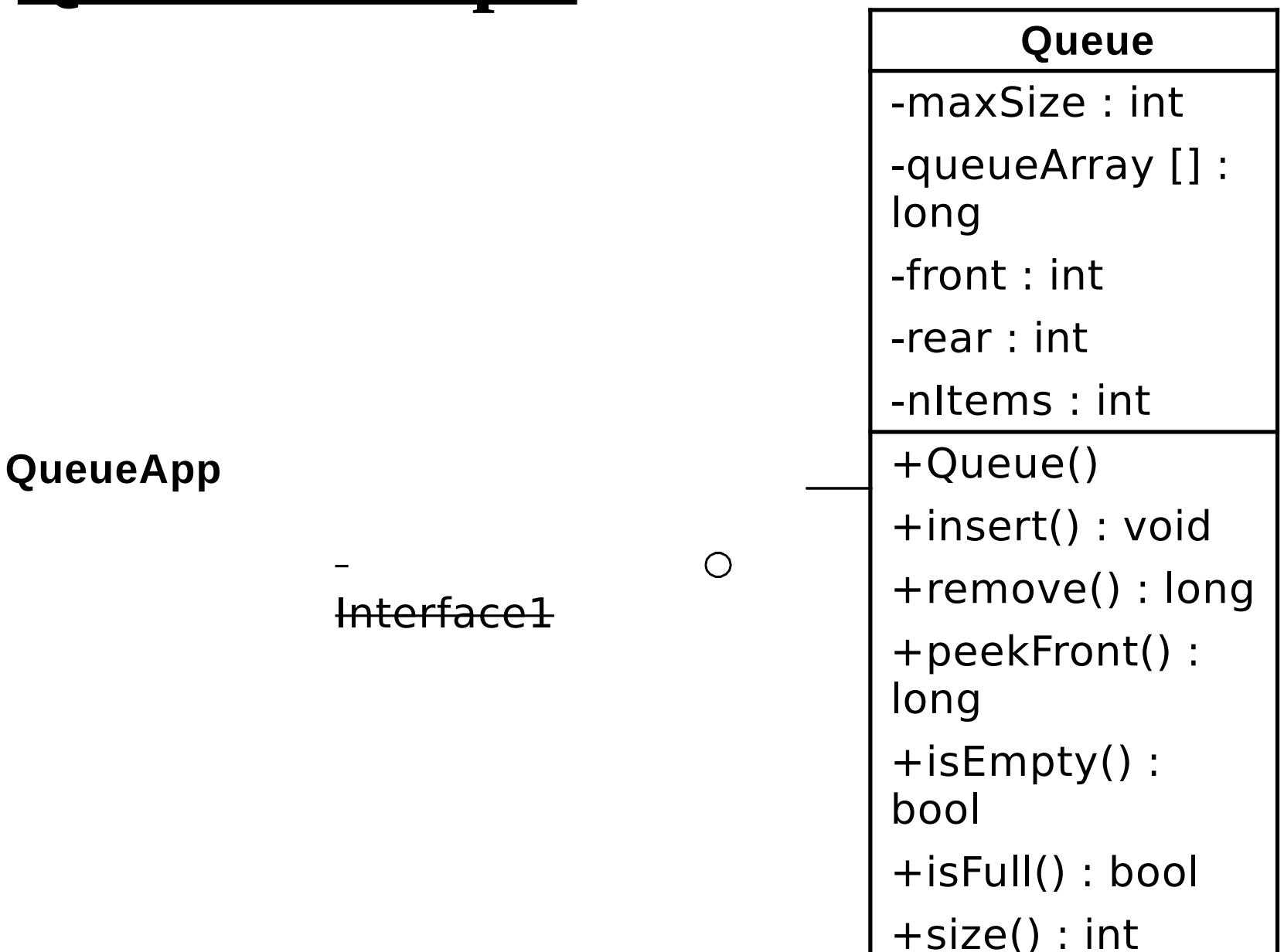
Representation of Queues

1. Using an array
   Using linked list

Circular Queue

- To solve this problem, queues implement wrapping around. Such queues are called Circular Queues.
- Both the front and the rear pointers wrap around to the beginning of the array.
- It is also called as “Ring buffer”.
- Items can inserted and deleted from a queue in O(1) time.

Queue Example



VariousQueue

- Normal queue (FIFO)
- Circular Queue (Normal Queue)
- Double-ended Queue (Deque)
- Priority Queue

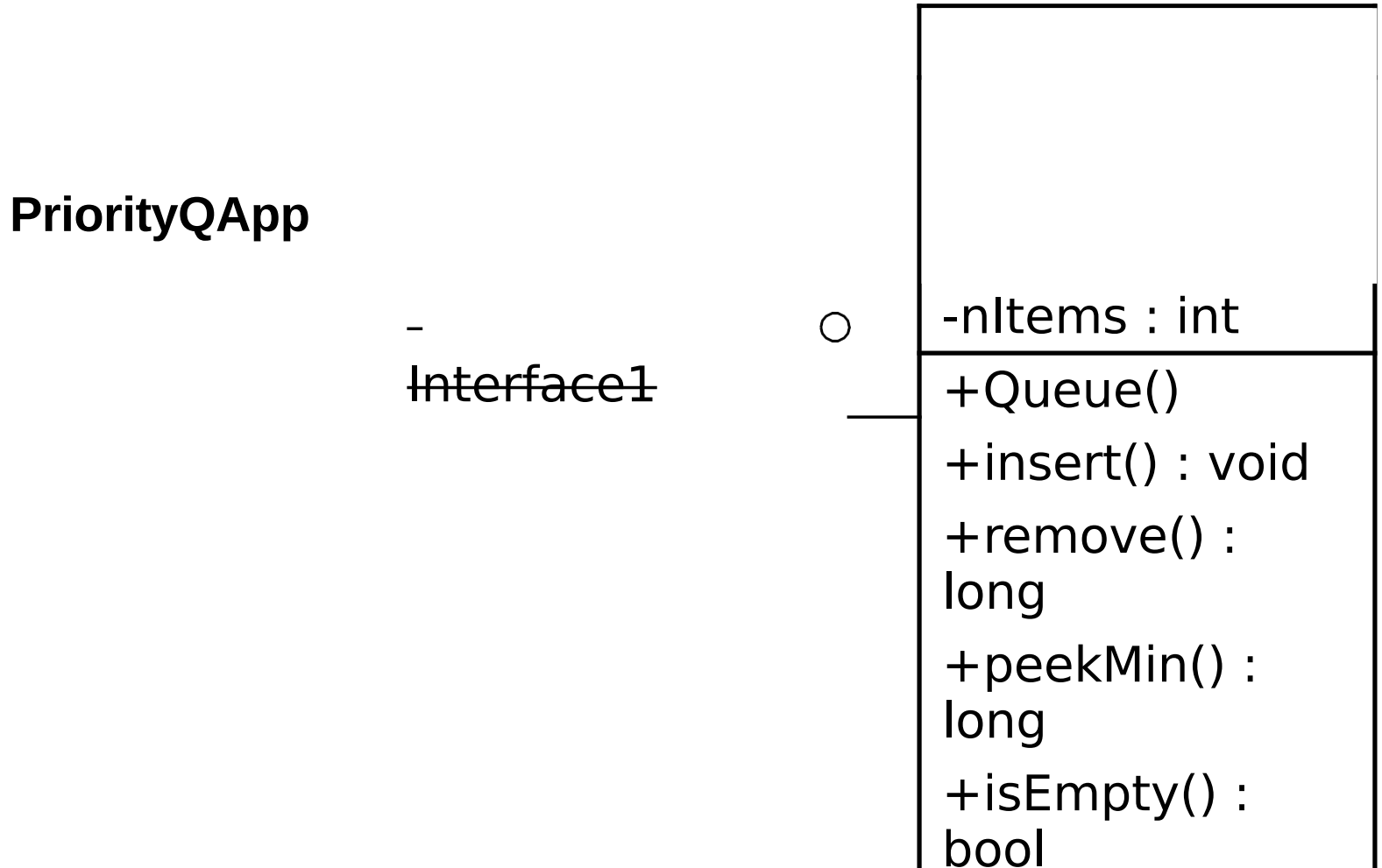
Deque

- It is a double-ended queue.
- Items can be inserted and deleted from either ends.
- More versatile data structure than stack or queue.
- E.g. policy-based application (e.g. low priority go to the end, high go to the front)
- In a case where you want to sort the queue once in a while, What sorting algorithm will you use?

Priority Queue

- More specialized data structure.
- Similar to Queue, having front and rear.
- Items are removed from the front.
- Items are ordered by key value so that the item with the lowest key (or highest) is always at the front.
- Items are inserted in proper position to maintain the order.
- Let’s discuss complexity

Priority Queue Example



Priority Queue

- Used in multitasking operating system.
- They are generally represented using “heap” data structure.
- Insertion runs in O(n) time, deletion in O(1) time.
- C:\Documents and Settings\box\My Documents\CS\CSC\220\ReaderPrograms\ReaderFiles\Chap04\PriorityQ\priorityQ.java

Parsing Arithmetic Expressions

- 2 +
- 2 + 4 *
  5
- ((2 + 4) * 7) + 3* (9 – 5))
  +

- 2 3 +
- 2 4 5 * +
- 2 4 + 7 * 3 9 5 - * +

- Infix vs postfix
- Why do we want to do this transformation?

Infix to postfix

Read ch from input until empty

If ch is arg , output = output + arg

- If ch is “(“, push ‘(‘;
- If ch is op and higher than top push ch
- If ch is “)” or end of input,
  - output = output + pop() until empty or top is “(“
- Read next input
- C:\Documents and Settings\box\My Documents\CS\CSC\220\ReaderPrograms\ReaderFiles\Chap04\Postfix\postfix.java

Postfix eval

- 5 + 2 * 3 -> 5 2 3 * +
- Algorithm
  - While input is not empty
  - If ch is number , push (ch)
  - Else
    - Pop (a)
    - Pop(b)
    - Eval (ch, a, b)

Quick XML Review

- XML – Wave of the future

Another Real world example

- <?xml version = "1.0"?
  >
- <!-- An author -->
- <author>
  - <name gender = "male">
    - <first> Art </first>
    - <last> Gittleman
  - </name> </last>
- </author>

Any questions?