Introduction to Programming Languages

Friday, April 1, 2022 4:57 PM

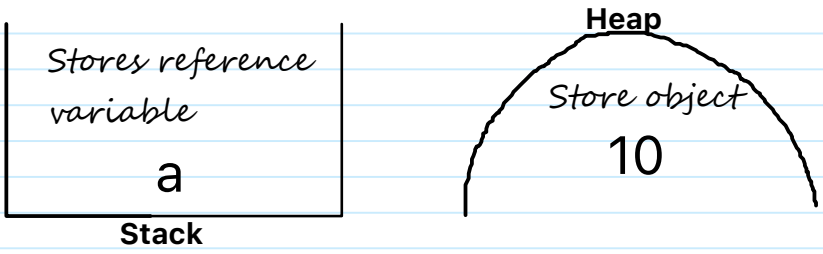
Types of Programming Languages

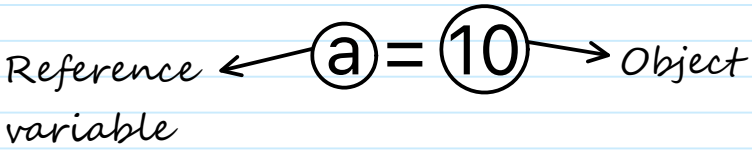
- Procedural
- Functional
- Object Oriented

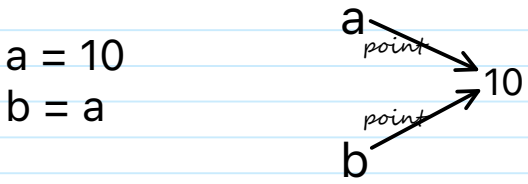
Static Languages = Perform type checking at compile time
Dynamic Languages = Perform type checking at runtime

Memory Management

There are 2 types of memory **Stack** and **Heap**

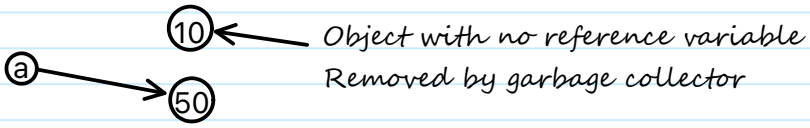






- More than one reference variable can point towords one object.
  If any of the reference variable changes the object then it is changed
- For all reference variable that points towords same object.

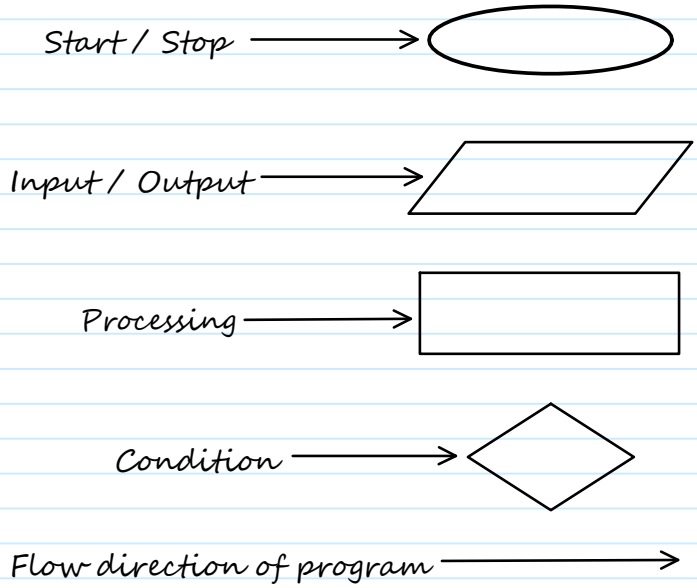
Now intially,
a = 10
a = 50



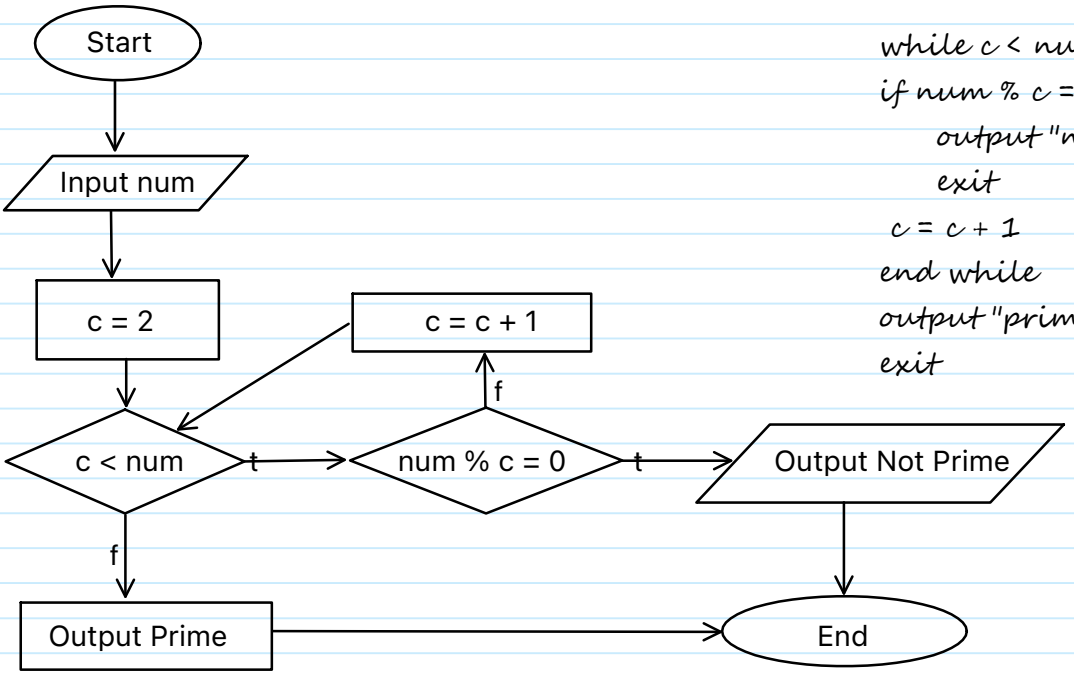
Flow of Program

Friday, April 1, 2022 7:28 PM

FlowCharts



Input a number and print whether it is prime or not.




```
while c < num;
if num % c = 0;
    output "not prime"
    exit
 c = c + 1
end while
output "prime"
exit
```

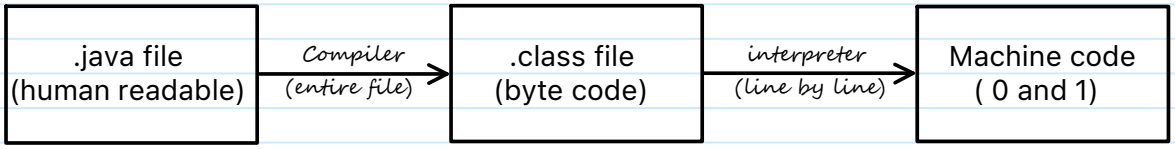
So, while c*c <= num;

1 x 36 = 36
2 x 18 = 36
3 x 12 = 36
4 x 9 = 36
6 x 6 = 36
9 x 4 = 36
12 x 3 = 36
18 x 2 = 36
36 x 1 = 36

Introduction to java

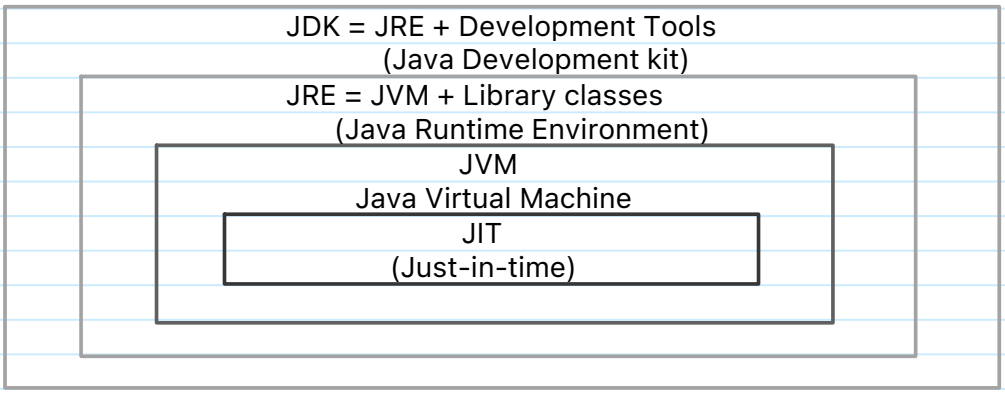
Saturday, April 2, 2022 9:48 AM

How java code executes



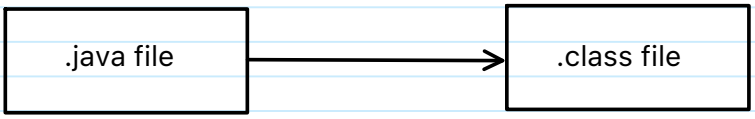
Java Compiler:- This byte code not directly run on system, We need JVM to run this, Reason why java is platform independent

Java Interpreter:- Converts byte code to machine code i.e. 0's and 1's, It translate the byte code line by line to machine code

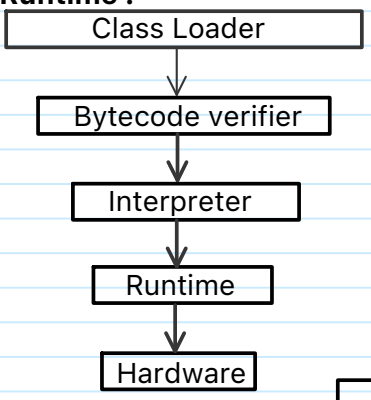


JDK :- Provide Environment to develop and run the java program
JRE :- It is an installation package that provides environment to only run the program

Compile Time :-



Runtime :-



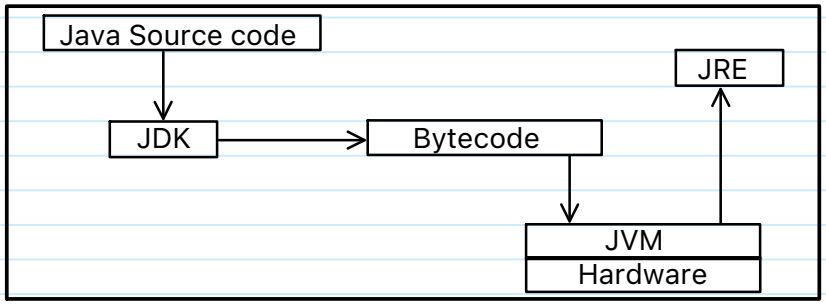
Class Loader *(How JVM Works)* :-

Loading → *Read .class file and generate binary data.*
Linking → *JVM verifies the .class file*
→ *Allocates memory for class variables and default values*
Initialization→ *JVM contains the stack and heap memory locations.*

JVM Execution :-

Interpreter → *Line by line execution, method is called many times*
JIT → *Those methods that are repeated, JIT provides direct machine code so that interpretation is not required.* →*Makes execution Faster.*
→ *Garbage collector*

Working of Java Architecture :-



First Java Program

Tuesday, April 5, 2022 1:45 PM

Converting .java to .class

- Using javac compiler we can convert .java file to .class
- Command to convert .java to .class

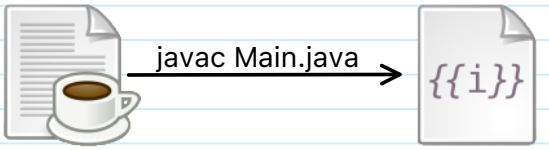
Javac and .java file name

- Let the name of .java file is Main,

Javac Main.java

- . class file contains Bytecode

File name: Demo.java
Class name: Demo



Main
.java file

Main.class
.class file

- Run the program,

java Main

Hello world program

```
public class Main{
        public static void main(String [] args){
                System.out.println("Hello World");
        }
}
```

1. **public (in first line)** :- public is an access modifier which allows to access the class from anywhere.
2. **class** :- It is a name group of properties and functions
3. **Main** :- It is just the name of class as same as the name of file.
4. **public (in second line)** :- It is used to allow the program to use main function from anywhere.
5. **static** :- It is a keyword which helps the main method to run without using objects.
6. **void** :- It is a keyword used when we do not want to return anything from a method/function
7. **main** :- It is the name of method.
8. **String [] args** :- It is a command line argument of string type array.
9. **System** :- It is a final class defined in java.lang package.
10. **out** :- It is a variable of PrintStream type which is public and static member field of the System class.
11. **println** :- It is a method of PrintStream class, It prints the argumants passed to it and adds a new line. **print** can aalso be used here but it prints only arguments passed to it. It do not adds a new line.
12. **Package** :- → It is just a folder in which java files lies.
    → It is used to provide some rules and stuff to our programs.

Data types	Description	Example
int	int is used to store numeric digits	int i = 26;
char	char is used to store character	char c = 'A';
float	float is used to store floating point numbers	float f = 98.67F;
double *(by default)*	double is used to store larger decimal numbers	double d = 45676.58975;
long	long is used to store numeric digits which is not able to stored in int	long l = 15876954832558315L;
boolean	It only stores store t values i.e., true or false.	boolean b = false;

Primitives data types are those data types which is not breakable, Ex. char ,int etc.

- **Literals** :- It is a synthetic representation of boolean, character, string, and numeric data.
  Ex:- int a = 10; Here 10 is called literal.
- **Identifiers**:- name of variable, methods, class, packages, etc. are known as identifiers.
  Ex:- int a = 10; Here a is Identifier.

Comment:- `//, /* */` and `int a = 23_00_00;` o/t `230000` underscore will be ignored

Inputs in Java

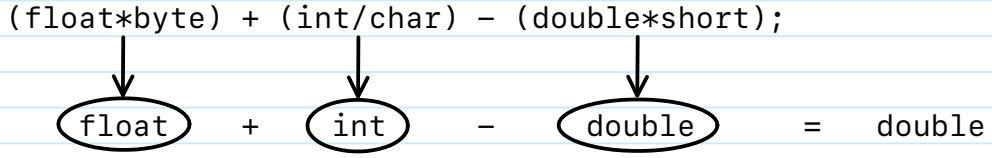
```
import java.util.Scanner;
public class Main{
        public static void main(String [] args){
                Scanner input = new Scanner(System.in);
                int rollno = input.nextInt();
                String s1 = input.next();
                String s2 = input.nextLine();
                System.out.println(rollno + " " + s1 + " " + s2);
        }
}
```

1. **Scanner** :- *It is a class required to take input, it is present in java.util package.*
2. **input** :- *It is an object that we are creating to take input.*
3. **new** :- *It is a keyword used to create an object in java.*
4. **System.in** :- *System is a class and in is a variable that denotes we are taking input from standard input stream (i.e. Keyboard).*

Type conversion :- When one type of data is assigned to another type of variable an automatic type conversion will take place under some condition

1. Two types should be compatible.
2. Destination type should be greater then the source type.

byte -> short -> int -> long -> float -> double



Type Casting :- When we convert one type of data to another type is known as type casting
Ex:- int num = (int) (67.564f)

Prime number program

```
public static void main(String[] args){
     Scanner in = new Scanner(System.in);
     System.out.println("Please enter a number);
     int n = in.nextInt();
     if(n<=1){
          System.out.println("Nither prime nor composite");
          return;
     }
     int c = 2;
     if(n==4){
          System.out.println("Not Prime");
     }
     else{
          while(c*c<n){
               if(n%c==0){
                    System.out.println("Not Prime");
                    return;
               }
               c=c+1;
          }
          if(c*c>n){
               System.out.println("Prime");
          }
      }
}
```

Output :-

```
Please enter a number
17
Prime
```

Celsius to Fahrenheit program.

```
public static void main (String[] args){
    Scanner in = new Scanner(System.in);
    float tempC = in.nextFloat();
    float tempF = (tempC*9/5) + 32;
    System.out.println("Temperature in fahrenheit: " + tempF);
}
```

Output :-

```
45
Temperature in fahrenheit: 113.0
```

Conditional and loops

Wednesday, April 6, 2022 9:37 AM

Condition :- It provide check for the statement.

If-else statement → Used to check the condition, it checks the Boolean condition True of False.

Example :-

```
public static void main(String[] args){
    int salary = 25400;
    if (salary<= 10000) {
        salary +=1000;
    }
    else if (salary <= 20000) {
        salary += 2000;
    }
    else {
        salary += 3000;
    }
    System.out.println(salary);
}
```

Output :- 28400

Loop :- Loops are used to iterate a part of program several times.

for loop → It is generally used when we know how many times loop will iterate.

Example:- print numbers from 1 to 5

```
public static void main(String[] args){
   for (int num=1;num<=5;num++){
         System.out.print(num + " ");
   }
}
```

Output :- 1 2 3 4 5

while Loop → It is used when we don't know how many time the loop will iterate.

Example :-

```
public static void main(String[] args) {
    int num = 1;
    while (num <=5){
        System.out.print(num + " ");
        num ++;
    }
 }
```

Output :- 1 2 3 4 5

do while loop → It is used when we want to execute our statement at least one time.
→ It is called exit control loop because it checks the condition after execution of statement.

Example :-

```
public static void main(String[] args) {
   int n = 1;
   do{
       System.out.println(n + " ");
       n++;
   } while(n<=5);
}
```

Output :- 1 2 3 4 5

Alphabet case check

```
import java.util.Scanner;
public class AlphabetCaseCheck {
    public static void main(String[] args) {
        Scanner in = new Scanner (System.in);
        char ch = in.next().trim().charAt(0);
        if (ch > 'a' && ch <= 'z'){
            System.out.println("Lowercase");
        }
        else {
            System.out.println("Uppercase");
        }
    }
}
```

Input :- a
Output :- Lowercase
Input :- Z
Output :- Uppercase

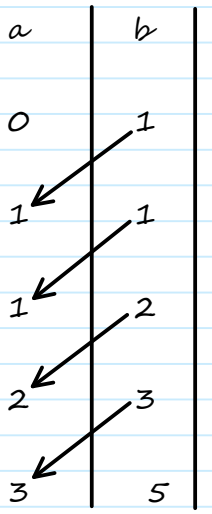
Fibonacci Numbers :- a series of numbers in which each number (Fibonacci number) is the sum of the two preceding numbers.

Ex :- 0,1,1,2,3,5,8,13...

Find the nth Fibonacci number.

```
import java.util.Scanner;
public class FibonacciNumbers{
    public static void main(String[] args) {
        Scanner in = new Scanner (System.in);
        int n = in.nextInt();
        int a = in.nextInt();
        int b = in.nextInt();
        int count = 2;

        while(count <= n){
            int temp = b;
            b = b+a;
            a = temp;
            count++;
        }
        System.out.println(b);
    }
}
```

0, 1, 1, 2, 3, 5, 8, 13,...



Input :- 0 1 7

Output :- 8.

Reverse a number

```
import java.util.Scanner;
public class ReverseANumber {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int num = in.nextInt();
        int ans = 0;
        while(num > 0){
           int rem = num % 10;
           num /= 10;
           ans = ans * 10 + rem;
        }
        System.out.println(ans);
    }
}
```

n = 1 3 8 5 7 5 7 8 7 9

n % 10 = last digit

n = 1 3 8 9 / 10

Ans = 3

```
10 | 1389 | 138
   - 1380
   ------
      9
```

Input :- 458792

Output :- 297854

n = 23597

ans = 7 * 10 + 9 = 79

= 79 * 10 + 5 = 795

= 795 * 10 + 3 = 7,953

= 7953 * 10 + 2 = 79532

Ans : 79532

2 3 5 9 7

Switch Statements + Nested case in java

```
switch ( expression) {

          case one :

                //code block

                break;

          case two :

                //code block

                break;

          default :

                // code block

}
```

Terminate the sequence

→ default will execute when none of above does
→ if default is not at end put break after it

→ if break is not used then it will continue with other cases.
→ duplicate cases not allowed.

`x.equals("word")` → here equals only checks value not reference

`x == " word"` → here it checks reference

```
Scanner in = new Scanner(System.in);
int day = in.nextInt();

switch (day) {
            case 1:
            case 2:
            case 3:
            case 4:
            case 5:
                System.out.println("Weekday");
                break;
            case 6:
            case 7:
                System.out.println("Weekend");
                break;
        }

switch (day) {
            case 1, 2, 3, 4, 5 -> System.out.println("Weekday");
            case 6, 7 -> System.out.println("Weekend");
        }
```

```java
package com.parth;

import java.util.Scanner;

public class NestedSwitch {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int empID = in.nextInt();
        String department = in.next();

        switch (empID) {
            case 1:
                System.out.println("Kunal Kushwaha");
                break;
            case 2:
                System.out.println("Parth Patel");
                break;
            case 3:
                System.out.println("Emp Number 3");
                switch (department) {
                    case "IT":
                        System.out.println("IT Department");
                        break;
                    case "Management":
                        System.out.println("Management Department");
                        break;
                    default:
                        System.out.println("No department entered");
                }
                break;
            default:
                System.out.println("Enter correct EmpID");
        }

        // better way to write

        switch (empID) {
            case 1 -> System.out.println("Kunal Kushwaha");
            case 2 -> System.out.println("Parth Patel");
            case 3 -> {
                System.out.println("Emp Number 3");
                switch (department) {
                    case "IT" -> System.out.println("IT Department");
                    case "Management" -> System.out.println("Management
                                                Department");
                    default -> System.out.println("No department entered");
                }
            }
            default -> System.out.println("Enter correct EmpID");
        }
    }
}
```

Functions/ Methods (in Java)

Thursday, April 7, 2022 10:29 AM

- A method is a block of code which only runs when it is called.
- To reuse code: define the code once, & use it many times.

This method myMethod() not have a return value.

Name of method

Syntax :

```
public class Main{
        static void myMethod(){
            // code
        }
}
```

Calling the function

Syntax :

```
public class Main{
        access_modifier return_type method(){
            // code
            return statement;
        }
}
```

name of function

function ends here

A return statement causes the program control to transfer back to the caller of a method.

Pass by value :

Object/value

```
main () {
        name = " a " ;
        greet ( name );
}
static void greet (naam) {

print(naam)
}
```

name → a

naam → a

Creating copy of value of name i.e. passing value of the reference.

```
psvm(){

        name = "a";
        change(name);
        print(name);
}

change(naam){

        naam = "b";

}
```

Creating copy

name → a

naam → a

name → a

naam → b

Not changing original object, just creating new object. Since it is created inside function it will not change original one.

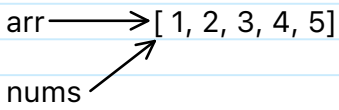
Points to be noted:

1. primitive data type like int, short, char, byte etc.
   → just pass value
2. object & reference:
   → passing value of reference variable

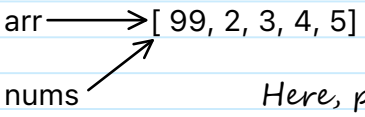
eg-1:

```
psvm(){
        a = 10;                 a → 10      but not
        b = 20;                 b → 10      here
        swap(a,b);
}

swap(num1, num2){
         temp = num1;           temp → 10   at function
         num1 = num2;           num1 → 20   scope level
         num2 = temp;           num2 → 10   they are swapped
}
```

eg-2:



num[0] = 99 [now, the value of 0th position in nums will change which also changes value of arr[0]]



Here, passing value of reference variable

Scopes:

Function scope : Variables declared inside a method / function scope (means inside method) can't be accessed outside the method.

eg :-

```
psvm(){
        }
all(){
        int x;
    }
```

X → can't be accessed outside

Block scope :

```
psvm(){
    int a = 10;
    int b = 20;
    {
        int a = 5;
        a = 100;
        int c = 20;
    }
    c = 10;
    int c = 15;
    a = 50;
}
```

variable initialized outside the block can be updated inside the box.

variables initialized inside the block cannot be updated outside the box but can be reinitialized outside the block.

variables like "a" here, is declared outside the block, updated inside the block and can also be updated outside the block.

Loop scope : Variables declared inside loop are having loop scope

Shadowing :

```
public class shadowing{
    static int x = 90;
    psvm(){
        System.out.println(x);
        x=50;     // here high-level scope is
        System.out.println(x);     shadowed by
    }                               low-level
}                                     scope
```

90

50

Variable Arguments(varargs) : Variable arguments is used to take a variable number of arguments. A method that takes a variable number of arguments is a varargs method.

Syntax :

```
static void fun(int…a){
        // method body
}
```

Here, would be array of type int[]

parameters

Method / Function Overloading : function Overloading happens when two functions have same name.

eg :- (1)

```
fun(){
    // code
}
fun(){
    // code
}
```

✕ function overloading

(2)

```
fun(int a){
    // code
}
fun(int a, int b){
    // code
}
```

This is allowed having different arguments with same method name

→ At compile time, it decides with f'n to run.

Armstrong number :

suppose there is number → 153

$153 \rightarrow (1)^3+(5)^3+(3)^3 = 1 + 125 + 27$

$= 153$

Function

```
public class Sum {
    public static void main(String[] args) {
//        int ans = sum2();
//        System.out.println(ans);

        int ans = sum3(20, 30);
        System.out.println(ans);
    }

    // pass the value of numbers when you are calling the method in main()
    static int sum3(int a, int b) {
        int sum = a + b;
        return sum;
    }

   // return the value
    static int sum2() {
        Scanner in = new Scanner(System.in);
        System.out.print("Enter number 1: ");
        int num1 = in.nextInt();
        System.out.print("Enter number 2: ");
        int num2 = in.nextInt();
        int sum = num1 + num2;
        return sum;
//        System.out.println("This will never execute");
    }
}
```

Swap

```
public class Swap {
    public static void main(String[] args) {
        // create an array
        int[] arr = {1, 3, 2, 45, 6};
        change(arr);
        System.out.println(Arrays.toString(arr));

        String name = "Kunal Kushwaha";
        changeName(name);
        System.out.println(name);
    }
    static void change(int[] nums) {
        nums[0] = 99; // if you make a change to the object via this ref variable,
    }                          same object will be changed
    static void changeName(String naam) {
        naam = "Rahul Rana"; // creating a new object
    }
}
```

VarArgs

```
  public static void main(String[] args) {

          multiple(2, 3, "Kunal", "Rahul", "dvytsbhusc");

      }

  static void multiple(int a, int b, String ...v) {

      }
```

Overloading

```
  public class Overloading {
      public static void main(String[] args) {
  //        fun(67);
  //        fun("Kunal Kushwaha");
          int ans = sum(3, 4, 78);
          System.out.println(ans);
      }
      static int sum(int a, int b) {
          return a + b;
      }
      static int sum(int a, int b, int c) {
          return a + b + c;
      }
      static void fun(int a) {
          System.out.println("first one");
          System.out.println(a);
      }
      static void fun(String name) {
          System.out.println("Second one");
          System.out.println(name);
      }
  }
```

Print all the 3 digits Armstrong numbers

```
package com.kunal;

import java.util.Scanner;

public class Questions {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
//        int n = in.nextInt();
//        boolean ans = isPrime(n);
//        System.out.println(ans);

        for (int i = 100; i < 1000; i++) {
            if (isArmstrong(i)) {
                System.out.print(i + " ");
            }
        }
    }

    // print all the 3 digits armstrong numbers
    static boolean isArmstrong(int n) {
        int original = n;
        int sum = 0;

        while (n > 0) {
            int rem = n % 10;
            n = n / 10;
            sum = sum + rem*rem*rem;
        }

        return sum == original;
    }

    static boolean isPrime(int n) {
        if (n <= 1) {
            return false;
        }
        int c = 2;
        while (c * c <= n) {
            if (n % c == 0) {
                return false;
            }
            c++;
        }
        return c * c > n;
    }
}
```

Array & Arraylist

Saturday, April 9, 2022 10:12 AM

An array is a collection of similar data type values.

Syntax of an Array :

datatype[] variable_name = new datatype[size]

eg :-

```
int[] rollno = new int[5];
```

store 5 roll numbers

or

it will create an object in heap me. of object size 5

```
int[] rollno = {51, 82, 13, 15,16}
```

represent the type of data store in array.

All the type of data in array should be same!

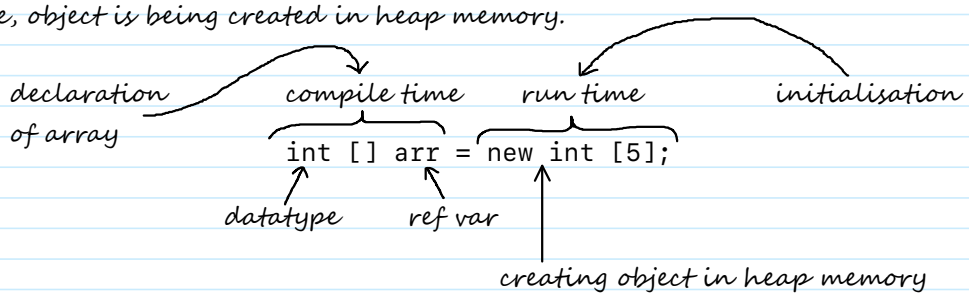
```
int [] rollnos;  // declaration of array
```

rollno are getting defined in stack

```
rollnos = new int [5]; // initialisation
```

actual memory allocation happens here

Here, object is being created in heap memory.



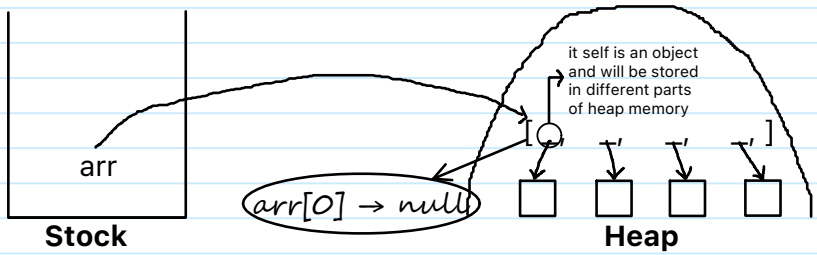
This above concept is known as Dynamic memory allocation which means at runtime OR execution time memory is allocated.

Internal Representation of Array :
internally in Java, memory allocation totally depends on JVM whether it be continuous or not!

- Objects are stored in heap memory.
- In JLS(Java Language specification) if is mentioned that heap objects are not continuous
- Dynamic memory allocation. Hence, array objects in Java may not be continous(depend on JVM)

Index of an array :-

index →	0	1	2	3	4	5
	3	8	9	10	53	33



- Primitives (int, char etc) are stored in stored in stack.
- All other objects are stored in heap memory

Array.toString(array**)** → internally uses for loop and gives the output in proper format.

- *In an array, since we can change the objects, hence they are mutable.*
- *String are immutable.*

```
1 2 3
4 5 6
7 8 9
```

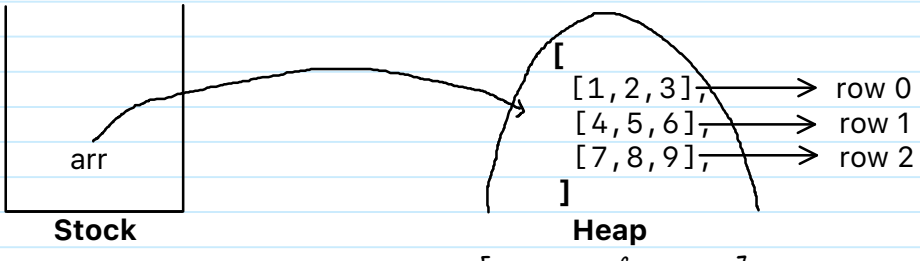
```
⇒ int [][] arr = new int[size][]
```

row — mandatory to give size of row

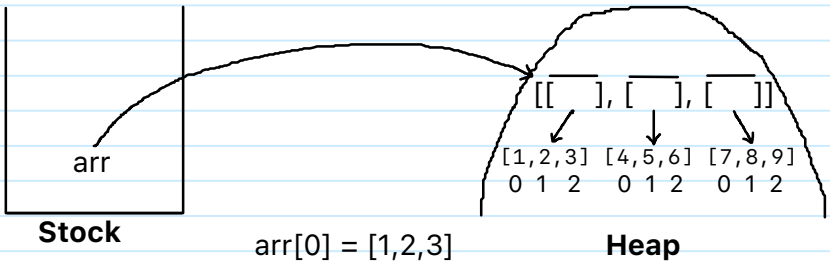
column — not mandatory to give size of column

or

```
int [][] arr = {
                    {1, 2, 3},
                    {4, 5, 6},
                    {7, 8, 9}
                }
```



[arrays of arrays]



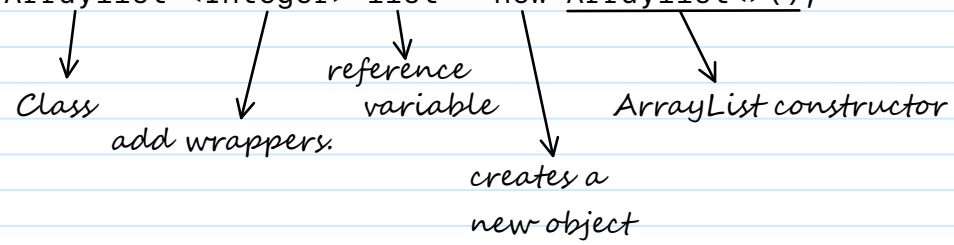
arr[0] = [1,2,3]
arr[0][2] = 3

ArrayList:

Arraylist is a part of collection framework and is present in java.util.package. It provides us with dynamic arrays in java. It is slower than standard arrays.

Syntax :-

```
Arraylist <Integer> list = new Arraylist<>();
```



- **add()** → Adds a new element to the ArrayList
- **set(index, value)** → Updates an existing value for a specified index
- **get(index)** → Used to retrieve an existing value for a specified index

Internal working of ArrayList :-

- Actually the size of the ArrayList is fixed but not permanently. It can change according to the input you provide.
- Suppose arraylist gets filled by some amount
  (a) It will make an arraylist of say double the size of arraylist initially.
  (b) Old elements are copied in the new arraylist.
  (c) Old ones are deleted.

Array

```
public class Main {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);

        // array of primitives
        int[] arr = new int[5];
        arr[0] = 23;
        arr[1] = 45;
        arr[2] = 233;
        arr[3] = 543;
        arr[4] = 3;
        // [23, 45, 233, 543, 3]
        System.out.println(arr[3]);

// input using for loops
        for (int i = 0; i < arr.length; i++) {
            arr[i] = in.nextInt();
        }

        System.out.println(Arrays.toString(arr));

       for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + " ");
        }

       for(int num : arr)//for every element in array,print the element
       {
        System.out.print(num + " ");//num represents element of the array
       }

//        System.out.println(arr[5]); // index out of bound error

        // array of objects
        String[] str = new String[4];
        for (int i = 0; i < str.length; i++) {
            str[i] = in.next();
        }

        System.out.println(Arrays.toString(str));

        // modify
        str[1] = "Patel";

        System.out.println(Arrays.toString(str));
    }
}
```

ArrayList

```
public class ArrayList {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        // Syntax
        ArrayList<Integer> list = new ArrayList<>(5);

//        list.add(67);
//        list.add(234);
```

```
//        list.add(654);
//        list.add(43);
//        list.add(654);
//        list.add(8765);

//        System.out.println(list.contains(765432));
//        System.out.println(list);
//        list.set(0, 99);
//
//        list.remove(2);
//
//        System.out.println(list);

// input using for loops
        for (int i = 0; i < 5; i++) {
            list.add(in.nextInt());
        }

        // get item at any index
        for (int i = 0; i < 5; i++) {
            System.out.println(list.get(i)); // pass index here,
list[index] syntax will not work here
        }

        System.out.println(list);

    }
}
```

Max

```
public class Max {
    public static void main(String[] args) {
        int[] arr = {1, 3, 2, 9, 18};
        System.out.println(maxRange(arr, 1, 3));
    }

    // work on edge cases here, like array being null
    static int maxRange(int[] arr, int start, int end) {

        if (start > end) {
            return -1;
        }

        if (arr == null) {
            return -1;
        }

        int maxVal = arr[start];
        for (int i = start; i <= end; i++) {
            if (arr[i] > maxVal) {
                maxVal = arr[i];
            }
        }
        return maxVal;
    }
```

Continued on the next page

```
static int max(int[] arr) {
        if (arr.length == 0) {
            return -1;
        }
        int maxVal = arr[0];
        for (int i = 1; i < arr.length; i++) {
            if (arr[i] > maxVal) {
                maxVal = arr[i];
            }
        }
        return maxVal;
    }
}
```

Multidimensional Array

```
public class MultiDimension {
    public static void main(String[] args) {

        Scanner in = new Scanner(System.in);
//        int[][] arr = new int[3][];

//        int[][] arr = {
//                {1, 2, 3}, // 0th index
//                {4, 5}, // 1st index
//                {6, 7, 8, 9} // 2nd index -> arr[2] = {6, 7, 8, 9}
//        };

        int[][] arr = new int[3][3];
        System.out.println(arr.length); // no of rows

        // input
        for (int row = 0; row < arr.length; row++) {
            // for each col in every row
            for (int col = 0; col < arr[row].length; col++) {
                arr[row][col] = in.nextInt();
            }
        }

        // output
//        for (int row = 0; row < arr.length; row++) {
//            // for each col in every row
//            for (int col = 0; col < arr[row].length; col++) {
//                System.out.print(arr[row][col] + " ");
//            }
//            System.out.println();
//        }

        // output
//        for (int row = 0; row < arr.length; row++) {
//            System.out.println(Arrays.toString(arr[row]));
//        }

        for(int[] a : arr) {
            System.out.println(Arrays.toString(a));
        }
    }
}
```

Multidimensional ArrayList

```
public class MultiAL {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        ArrayList<ArrayList<Integer>> list = new ArrayList<>();

        // initialisation
        for (int i = 0; i < 3; i++) {
            list.add(new ArrayList<>());
        }

        // add elements
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                list.get(i).add(in.nextInt());
            }
        }

        System.out.println(list);
    }
}
```

Swap

```
public class Swap {
    public static void main(String[] args) {
        int[] arr = {1, 3, 23, 9, 18, 56};
//      swap(arr, 0, 4);
        reverse(arr);
        System.out.println(Arrays.toString(arr));
    }

    static void reverse(int[] arr) {
        int start = 0;
        int end = arr.length-1;

        while (start < end) {
            // swap
            swap(arr, start, end);
            start++;
            end--;
        }
    }

    static void swap(int[] arr, int index1, int index2) {
        int temp = arr[index1];
        arr[index1] = arr[index2];
        arr[index2] = temp;
    }
}
```

Linear Search

Tuesday, April 12, 2022 7:47 PM

Linear / Sequential Search :-

- *It is basic & simple search algorithm.*
- *In sequential search, we compare the target value with all the other elements given in the list.*

eg:-

target = 77

```
          0   1   2   3   4   5
arr = [18, 12, 19, 77, 29, 50]
```

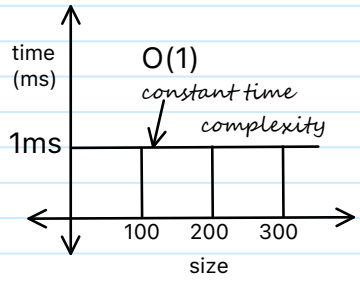
(unsorted array)

start

In above example, the target value is compared with all the elements in array in sequential/linear way.

Time Complexity :-

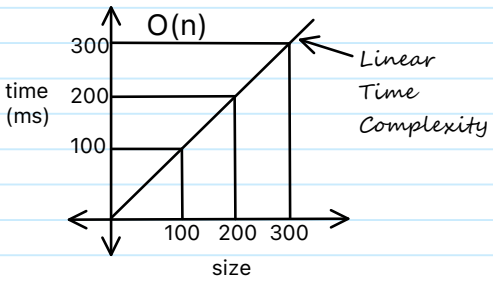
Best case :- O(1) → constant



Best Case

case i.e., the element will be found at 0th index
i.e., only one comparision will be made for best case.

worst case :- O(n)



Worst Case

case → Worst case, here it will go through every element and then it says element not found.

Linear Search

```
public class Main {

    public static void main(String[] args) {
        int[] nums = {23, 45, 1, 2, 8, 19, -3, 16, -11, 28};
        int target = 19;
        boolean ans = linearSearch3(nums, target);
        System.out.println(ans);
    }
```

```java
    // search the target and return true or false
    static boolean linearSearch3(int[] arr, int target) {
        if (arr.length == 0) {
            return false;
        }

        // run a for loop
        for (int element : arr) {
            if (element == target) {
                return true;
            }
        }
        // this line will execute if none of the return statements above
have executed
        // hence the target not found
        return false;
    }

    // search the target and return the element
    static int linearSearch2(int[] arr, int target) {
        if (arr.length == 0) {
            return -1;
        }

        // run a for loop
        for (int element : arr) {
            if (element == target) {
                return element;
            }
        }
        // this line will execute if none of the return statements above
have executed
        // hence the target not found
        return Integer.MAX_VALUE;
    }

    // search in the array: return the index if item found
    // otherwise if item not found return -1
    static int linearSearch(int[] arr, int target) {
        if (arr.length == 0) {
            return -1;
        }

        // run a for loop
        for (int index = 0; index < arr.length; index++) {
            // check for element at every index if it is = target
            int element = arr[index];
            if (element == target) {
                return index;
            }
        }
        // this line will execute if none of the return statements above
have executed
        // hence the target not found
        return -1;
    }
}
```

SearchInStrings

```
public static void main(String[] args) {
        String name = "Kunal";
        char target = 'u';
//        System.out.println(search(name, target));

        System.out.println(Arrays.toString(name.toCharArray()));
    }


    static boolean search2(String str, char target) {
        if (str.length() == 0) {
            return false;
        }

        for(char ch : str.toCharArray()) {
            if (ch == target) {
                return true;
            }
        }
        return false;
    }
```

SearchInRange

```
public static void main(String[] args) {
        int[] arr = {18, 12, -7, 3, 14, 28};
        int target = 3456;
        System.out.println(linearSearch(arr, target, 1, 4));
    }

    static int linearSearch(int[] arr, int target, int start, int end){
        if (arr.length == 0) {
            return -1;
        }

        for (int index = start; index <= end; index++) {
            // check for element at every index if it is = target
            int element = arr[index];
            if (element == target) {
                return index;
            }
        }
        // this line will execute if none of the return statements above
           have executed
        // hence the target not found

        return -1;
    }
```

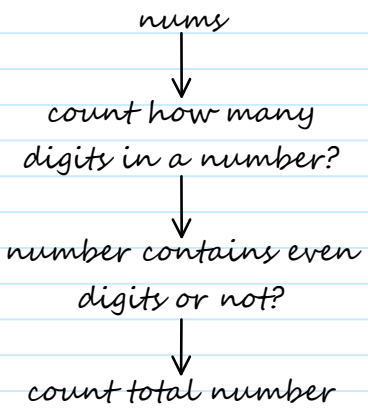
Find Minimum Number

```
public static void main(String[] args) {
        int[] arr = {18, 12, 7, 3, 14, 28};
        System.out.println(min(arr));
    }
    // assume arr.length != 0
```

```
    // return the minimum value in the array
static int min(int[] arr) {
        int ans = arr[0];
        for (int i = 1; i < arr.length; i++) {
            if (arr[i] < ans) {
                ans = arr[i];
            }
        }
        return ans;
    }
```

EvenDigits

```
// https://leetcode.com/problems/find-numbers-with-even-number-of-digits

public class EvenDigits {

    public static void main(String[] args) {

        int[] nums = {12,345,2,6,7896};
        System.out.println(findNumbers(nums));

        System.out.println(digits2(-345678));
    }

    static int findNumbers(int[] nums) {
        int count = 0;
        for(int num : nums) {
            if (even(num)) {
                count++;
            }
        }
        return count;
    }


    // function to check whether a number contains even digits or not

    static boolean even(int num) {
        int numberOfDigits = digits(num);

//        if (numberOfDigits % 2 == 0) {
//            return true;
//        }
//        return false;

        return numberOfDigits % 2 == 0;
    }

//    static int digits2(int num) {
//        if (num < 0) {
//            num = num * -1;
//        }
//        return (int)(Math.log10(num)) + 1;
//    }
```



Continued on the next page

```java
// count number of digits in a number
    static int digits(int num) {

        if (num < 0) {
            num = num * -1;
        }

        if (num == 0) {
            return 1;
        }

        int count = 0;
        while (num > 0) {
            count++;
            num = num / 10; // num /= 10
        }

        return count;
    }
}
```

MaxWealth

```java
// https://leetcode.com/problems/richest-customer-wealth

public class MaxWealth {

    public static void main(String[] args) {
        int[][] arr = {{1,5},{7,3 },{3,5}};
        int sum = maximumWealth(arr);
        System.out.println("The total sum is: " + sum);
    }

    static int maximumWealth(int[][] accounts) {
        // person = rol
        // account = col

        int ans = Integer.MIN_VALUE;
        for (int[] ints : accounts) {
            // when you start a new row, take a new sum for that row
            int sum = 0;
            for (int anInt : ints) {
                sum += anInt;
            }

            // now we have sum of accounts of person
            // check with overall ans
            if (sum > ans) {
                ans = sum;
            }
        }
        return ans;
    }
}
```

SearchIn2DArray

```
public class SearchIn2DArray {

    public static void main(String[] args) {

        int[][] arr = {
                {23, 4, 1},
                {18, 12, 3, 9},
                {78, 99, 34, 56},
                {18, 12}
        };

        int target = 56;
        int[] ans = search(arr,target); //format of return value {row,col}

        System.out.println(Arrays.toString(ans));

        System.out.println(max(arr));

        System.out.println(Integer.MIN_VALUE);
    }

    static int[] search(int[][] arr, int target) {
        for (int row = 0; row < arr.length; row++) {
            for (int col = 0; col < arr[row].length; col++) {
                if (arr[row][col] == target) {
                    return new int[]{row, col};
                }
            }
        }
        return new int[]{-1, -1};
    }

    static int max(int[][] arr) {
        int max = Integer.MIN_VALUE;

        for (int[] ints : arr) {
            for (int element : ints) {
                if (element > max) {
                    max = element;
                }
            }
        }
        return max;
    }
}
```

Binary Search

Friday, April 15, 2022 11:00 AM

Algorithm :-

arr = [2, 4, 6, 9, 11, 12, 14, 20, 36, 48]

sorted array → ascending order

target 36

1. Find the middle element
2. check:
3. if target > middle → search in right
4. else → search in left
5. if target == middle → we found element.

Example :- target 36

In the above array,

arr = [2(0), 4(1), 6(2), 9(3), 11(4), 12(5), 14(6), 20(7), 36(8), 48(9)]

1st → find middle element

$mid = \frac{start + end}{2} = \frac{0 + 9}{2} = 4$ [the element at index 4 is the middle element]

2nd → let's check:

Is target > middle ➡ 36 > 11 ➡ yes! ➡ check in right side

Now,

arr = [2, 4, 6, 9, 11, 12(5), 14(6), 20(7), 36(8), 48(9)]

mid = (5+9)/2 = 7 [the element at index 7 is the middle element]

Let's check :

Is target > middle ➡ 36 > 20 ➡ yes! ➡ check in right side

i.e.,

arr = [2, 4, 6, 9, 11, 12, 14, 20, 36(8), 48(9)]

mid = (8+9)/2 = 8 [the element at index 8 is the middle element]

Here,

Is target == middle ➡ 36 == 36 ➡ we found element at index 8

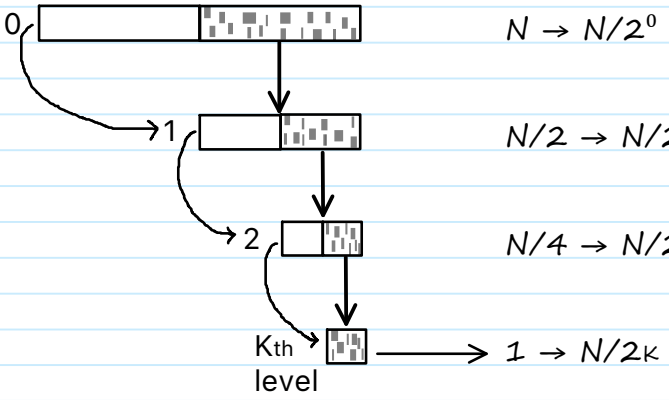
Here, the space in which we are searching is getting divided into two spaces

Time Complexity :-

Best case : O(1)

Worst case: O(log n)

Explanation : find the maximum number of comparisons



at the end
only one element
will be there

$N/2^K = 1$

$N = 2^K$

$\log(N) = \log(2^K)$

$\log(N) = k\log 2$

$k = \log N / \log 2$

$k = \log_2 N$ → size of array

total number of comparison in worst case

$m = (s + e)/2$ → this may exceeds the int range

$m = s + (e - s)/2$
$= (2s + e - s)/2$
$= (s+e)/2$

Order agnostic Binary Search

Let's say if we don't know that the array is sorted in ascending or descending order.

```
arr = [ 96, 78, 44, 16, 12, 9, 3, 1]
```

Target = 78

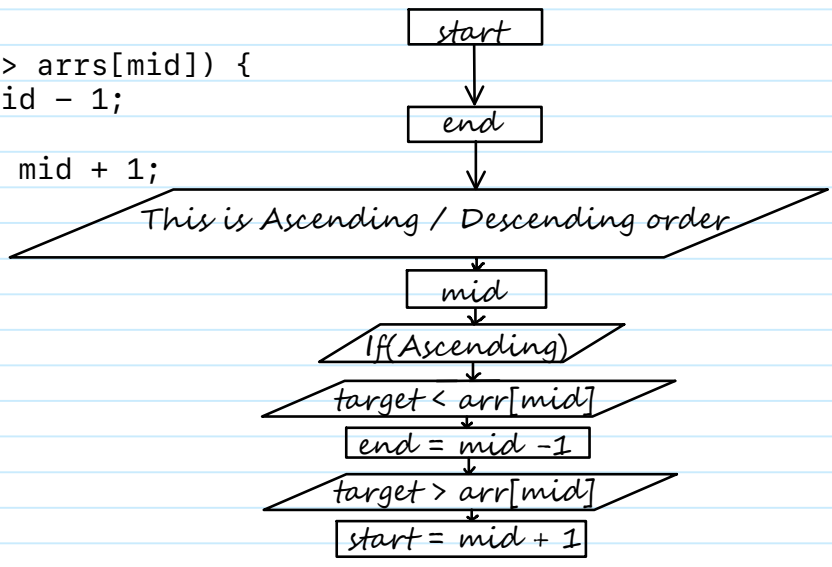
Here, target > middle ➡ search in left

Here, start > end] → Descending order

When start < end → Ascending order

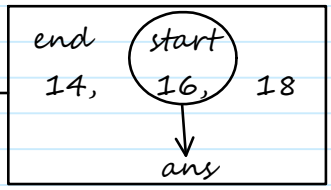
BinarySearch

```
public class BinarySearch {
    public static void main(String[] args) {
//        int[] arr = {2, 3, 5, 9, 14, 16, 18};
        int[] arr = {99, 80, 75, 22, 11, 10, 5, 2, -3};
        int target = 11;
        int arr1 = binarySearch(arr, target);
        System.out.println(arr1);
    }

    // return the index
    // return -1 if it does not exist
    static int binarySearch(int[] arrs, int target) {
        int start = 0;
        int end = arrs.length - 1;
        // find whether the array is sorted in ascending or descending
        boolean isAsc = arrs[start] < arrs[end];
//        if(arr[start] <= arr[end]){
//            isAsc = true;
//        }
//        else{
//            isAsc = false;
//        }

        while (start <= end) {

            // find the middle element
            // int mid = (start + end) / 2; // might be possible that (start
            + end) exceeds the range of int in java
            int mid = start + ((end - start) / 2);

            if (arrs[mid] == target) {
                return arrs[mid];
            } else if (isAsc) {
                if (target < arrs[mid]) {
                    end = mid - 1;
                } else {
                    start = mid + 1;
                }
            } else {
                if (target > arrs[mid]) {
                    end = mid - 1;
                } else {
                    start = mid + 1;
                }
            }
        }
        return -1;
    }
}
```



Celling of a Number
// Floor of a Number

```
public class Ceiling {

    public static void main(String[] args) {
        int[] arr = {2, 3, 5, 9, 14, 16, 18};
        int target = 15;
        int ans = ceiling(arr, target);
        System.out.println(ans);

        //int target = 1;
        //int ans = floor(arr, target);
        //System.out.println(ans);

    }

    // return the index of smallest no >= target
    static int ceiling(int[] arr, int target) {

        // but what if the target is greater than the greatest number in
           the array
        if (target > arr[arr.length - 1]) {
            return -1;
        }

        int start = 0;
        int end = arr.length - 1;

        while(start <= end) {

            int mid = start + (end - start) / 2;

            if (target < arr[mid]) {
                end = mid - 1;
            } else if (target > arr[mid]) {
                start = mid + 1;
            } else {
                // ans found
                return mid;
            }
        }
        return arr[start];
        // return the index: greatest number <= target
        //return arr[end];
    }
}
```



start Target end

↓

end Target start // condition for while loop voileted

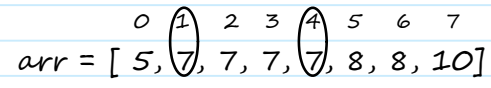
start <= end → when while loop breake,

because start = end + 1

34. Find First and Last Position of Element in Sorted Array

// https://leetcode.com/problems/find-first-and-last-position-of-element-in-sorted-array

```
public static void main(String[] args) {
        int[] nums = {5, 7, 7, 7, 7, 8, 8, 10};
        int target = 7;
        System.out.println("{ " + findFirst(nums, target) + " , " +
                findLast(nums, target) + " }");
    }


    static int findFirst(int[] nums, int target) {
        int start = 0;
        int end = nums.length - 1;
        int pos = -1;
        while (start <= end) {
            int mid = start + ((end - start) / 2);
            if (target < nums[mid]) {
                end = mid - 1;

            } else if (target > nums[mid]) {
                start = mid + 1;
            } else {
                pos = mid;
                end = mid - 1;
            }
        }
        return pos;
    }

    static int findLast(int[] nums, int target) {
        int start = 0;
        int end = nums.length - 1;
        int pos = -1;
        while (start <= end) {
            int mid = start + ((end - start) / 2);
            if (target < nums[mid]) {
                end = mid - 1;

            } else if (target > nums[mid]) {
                start = mid + 1;
            } else {
                pos = mid;
                start = mid + 1;
            }
        }
        return pos;
    }
```



target = 7

first occurrence of 7 → end = mid - 1;

last occurrence of 7 → start = mid + 1;

Infinite Array

```
// https://www.geeksforgeeks.org/find-position-element-sorted-array-infinite-numbers

public class InfiniteArray {
    public static void main(String[] args) {
        int[] arr = {3, 5, 7, 9, 10, 90, 100, 130, 140, 160, 170};
        int target = 10;
        System.out.println(ans(arr, target));
    }
    static int ans(int[] arr, int target) {
        // first find the range
        // first start with a box of size 2
        int start = 0;
        int end = 1;

        // condition for the target to lie in the range
        while (target > arr[end]) {
            int temp = end + 1; // this is my new start
            // double the box value
            // end = previous end + sizeofbox*2
            end = end + (end - start + 1) * 2;
            start = temp;
        }
        return binarySearch(arr, target, start, end);

    }
    static int binarySearch(int[] arr, int target, int start, int end) {
        while(start <= end) {
            // find the middle element
//            int mid = (start + end) / 2; // might be possible that (start + end) exceeds the range of int in java
            int mid = start + (end - start) / 2;

            if (target < arr[mid]) {
                end = mid - 1;
            } else if (target > arr[mid]) {
                start = mid + 1;
            } else {
                // ans found
                return mid;
            }
        }
        return -1;
    }
}
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13
[3,	5,	7,	9,	10,	90,	100,	130,	140,	160,	170,	190,	220,	250]

4 = end - start + 1

= 5 - 2 + 1 = 4

double the box value

end = previous end + sizeofbox*2

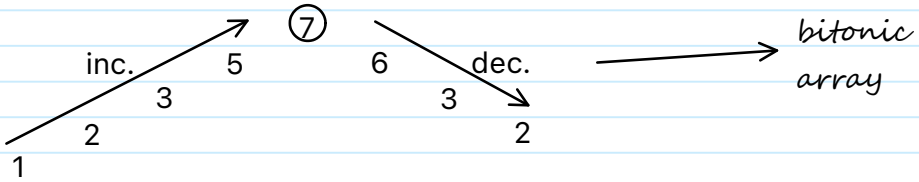
end = end + (end - start + 1) * 2 = 5 + 4 * 2 = 13

Peak Index in a Mountain Array

// https://leetcode.com/problems/peak-index-in-a-mountain-array
// https://leetcode.com/problems/find-peak-element

```
public static void main(String[] args){
        int[] arr = {3, 5, 7, 9, 10, 90, 100, 13, 10, 6, 2};
        System.out.println(peakIndexinaMountainArray(arr));
    }

    static int peakIndexinaMountainArray(int[] arr){
        int start = 0;
        int end = arr.length-1;

        while (start<end){
            int mid = start+(end-start)/2;
            if(arr[mid]>arr[mid+1]){
                // you are in dec part of array
                // this may be the ans, but look at left
                // this is why end != mid - 1
                end = mid;
            } else {
                // you are in asc part of array
                start = mid + 1; // because we know that mid+1 element>
                                    mid element
            }
        }

        // in the end, start == end and pointing to the largest number
        because of the 2 checks above
        // start and end are always trying to find max element in the
        above 2 checks
        // hence, when they are pointing to just one element, that is
        the max one because that is what the check say
        // more elaboration: at every point of time for start and end,
        they have the best possible answer till that time
        // and if we are saying that only one item is remaining, hence
        cuz of above line that is the best possible ans

        return start; // or return end as both are equal
    }
```

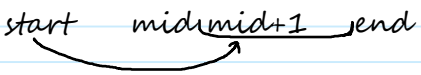
arr = [1, 2, 3, 5, 7, 6, 3, 2]



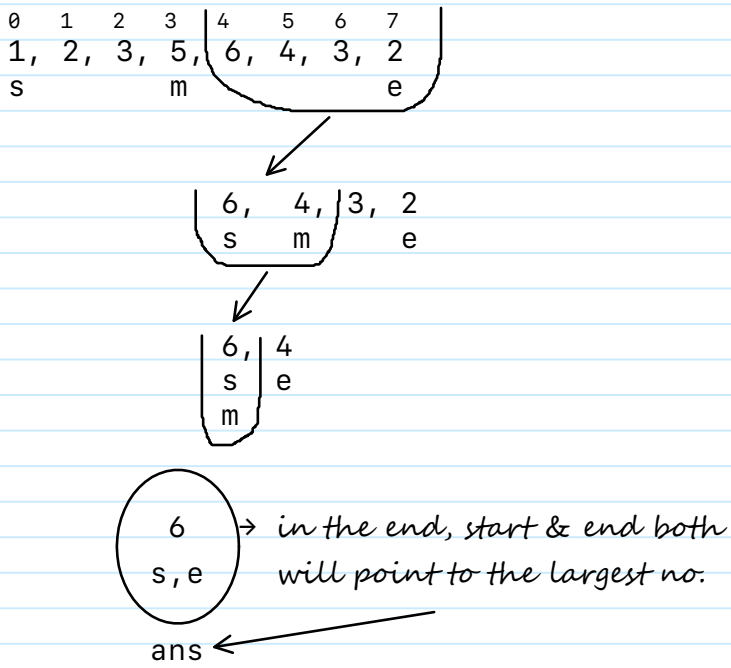
case 1: if element[mid] > element[mid + 1] → you arr in the decending part of array
end = mid // because checking left hand side

case 2: if element[mid] < element[mid + 1] → you arr in the ascending part of array

start = mid + 1



case 3: when will loop break?




```
          0  1  2  3  4  5  6
arr = [ 1, 2, 3, 4, 5, 3, 1]
target = 3
```

1. find peak element → 4 index
2. Binary search in asc array → (0, 4)
3. if not found, binary search in dec array = (4, 6)

Search in Rotated Sorted Array

https://leetcode.com/problems/search-in-rotated-sorted-array/submissions

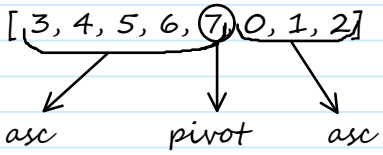
```
public class Search_in_Rotated_Sorted_Array_33 {
    public static void main(String[] args) {
        int[] arr = {1,2,3,4,5,5,6};
        System.out.println(findPivotWithDuplicates(arr));
    }

    static int search(int[] nums, int target) {
        int pivot = findPivot(nums);

        // if you did not find a pivot, it means the array is not rotated
        if (pivot == -1) {
            // just do normal binary search
            return binarySearch(nums, target, 0 , nums.length - 1);
        }
```

```
        // if pivot is found, you have found 2 asc sorted arrays
        if (nums[pivot] == target) {
            return pivot;
        }

        if (target >= nums[0]) {
            return binarySearch(nums, target, 0, pivot - 1);
        }

        return binarySearch(nums, target, pivot + 1, nums.length - 1);
    }

    static int binarySearch(int[] arr, int target, int start, int end)
{
        while(start <= end) {
            int mid = start + (end - start) / 2;

            if (target < arr[mid]) {
                end = mid - 1;
            } else if (target > arr[mid]) {
                start = mid + 1;
            } else {
                // ans found
                return mid;
            }
        }
        return -1;
    }

    // this will not work in duplicate values
    static int findPivot(int[] arr) {
        int start = 0;
        int end = arr.length - 1;
        while (start <= end) {
            int mid = start + (end - start) / 2;
            // 4 cases over here
            if (mid < end && arr[mid] > arr[mid + 1]) {
                return mid;
            }
            if (mid > start && arr[mid] < arr[mid - 1]) {
                return mid-1;
            }
            if (arr[mid] <= arr[start]) {
                end = mid - 1;
            } else {
                start = mid + 1;
            }
        }
        return -1;
    }

    static int findPivotWithDuplicates(int[] arr) {
        int start = 0;
        int end = arr.length - 1;
```

```
while (start <= end) {
            int mid = start + (end - start) / 2;
            // 4 cases over here
            if (mid < end && arr[mid] > arr[mid + 1]) {
                return mid;
            }
            if (mid > start && arr[mid] < arr[mid - 1]) {
                return mid-1;
            }

            // if elements at middle, start, end are equal then just skip the
               duplicates
            if (arr[mid] == arr[start] && arr[mid] == arr[end]) {
                // skip the duplicates
                // NOTE: what if these elements at start and end were the
                   pivot??
                // check if start is pivot
                if (start < end && arr[start] > arr[start + 1]) {
                    return start;
                }
                start++;

                // check whether end is pivot
                if (end > start && arr[end] < arr[end - 1]) {
                    return end - 1;
                }
                end--;
            }
            // left side is sorted, so pivot should be in right
            else if(arr[start] < arr[mid] || (arr[start] == arr[mid] &&
                    arr[mid] > arr[end])) {
                start = mid + 1;
            } else {
                end = mid - 1;
            }
        }
        return -1;
    }
```

1. find the pivot in the array

   pivot → from where your next numbers are asc.

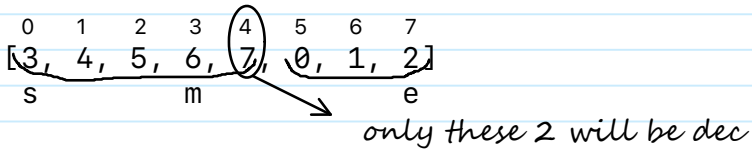


→ find pivot

→ search in first half : (0, pivot) → simple Binary Search

→ otherwise, search in second half : (pivot + 1, end)

Find pivot :-



case 1:

Ans when? → When you find that mid > mid + 1 element that is pivot

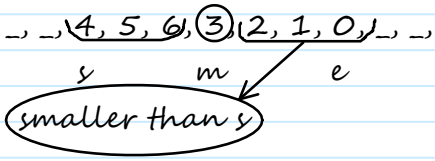
case 2:

if mid element < (mid - 1) element

i.e also my ans → ans = mid - 1

case 3:

start element >= mid element



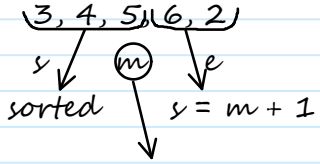
In this care, all elements from mid, will be < start

Hence, we can ignore all these elements since we are looking for peak i.e. largest element.

end = mid - 1

case 4:

start element < mid element

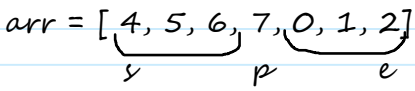


if this way pivot, it would have been returned in case 1 & 2

Hence proved, that bigger nos. lie a head.

Hence, ignore mid & put start = mid + 1

Do Rotate Binary Search using pivot :-



case 1: pivot element = target // Ans

case 2: target > start element // search for 6

search space = (s, p-1)

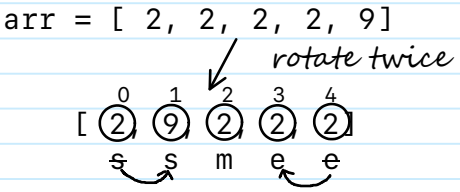
why? because all nos. after pivot are < start

case 3: target < start element

i.e. we know that all elements from s, pivot are then going to be bigger then target // target = 1

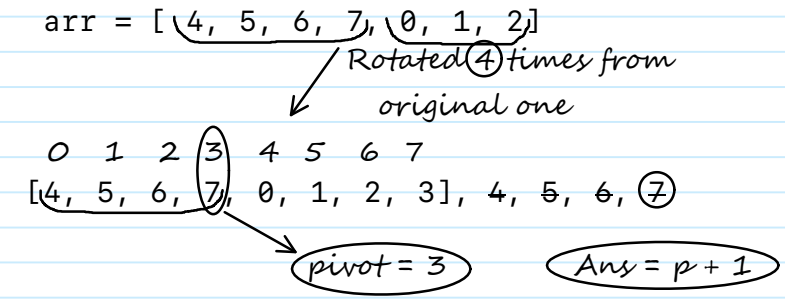
search space = (pivot + 1 till end)

Rotated Binary Search in Array with Duplicate Values :-



Rotation Count

```
public static void main(String[] args) {
        int[] arr = {4,5,6,7,0,1,2};
        System.out.println(countRotations(arr));
    }

    private static int countRotations(int[] arr) {
        int pivot = findPivot(arr);
        return pivot + 1;
    }

    // use this for non duplicates
    static int findPivot(int[] arr) {
        int start = 0;
        int end = arr.length - 1;
        while (start <= end) {
            int mid = start + (end - start) / 2;
            // 4 cases over here
            if (mid < end && arr[mid] > arr[mid + 1]) {
                return mid;
            }
            if (mid > start && arr[mid] < arr[mid - 1]) {
                return mid-1;
            }
            if (arr[mid] <= arr[start]) {
                end = mid - 1;
            } else {
                start = mid + 1;
            }
        }
        return -1;
    }
```



Split Array Largest Sum

```
public static void main (String[] args){
        int[] nums = {7,2,5,10,8};
        int m = 2;
        System.out.println(splitArray(nums, m));
    }

    static int splitArray(int[] nums, int m){
        int start = 0;
        int end = 0;

        for(int i = 0; i<nums.length; i++){
            start = Math.max(start, nums[i]); // in the end of the loop
                                                this will contain the max
                                                item of the array
            end += nums[i];
        }

        while(start<end){
            // try for the middle as potential ans
            int mid = start + (end - start) / 2;

            // calculate how many pieces you can divide this in with this max
               sum
            int sum = 0;
            int pieces = 1;

            for(int num: nums){

                if(sum+num > mid){
                   // you cannot add this in this subarray, make new one
                   // say you add this num in new subarray, then sum = num
                    sum = num;
                    pieces++;
                }
                else{
                    sum += num;
                }
            }

            if(pieces > m){
                start = mid + 1;
            }
            else{
                end = mid;
            }
        }
        return end; // here start == end
    }
}
```

arr = [7 , 2, 5, 10, 8] , m = 2

largest

[7, 2, 5, 10] 24	[8] 8	= 24
[7, 2, 5] 14	[10, 8] 18	= 18 (ans)
[7, 2] 9	[5, 10, 8] 23	= 23
[7] 7	[2, 5, 10, 8] 25	= 25

1. minimum no. of partitions that we can make = 1
2. what is the maxmum no. of partitions or m that can be = N

arr = [3, 4, 1, 2]

= [3], [4], [1], [2]

what will be the ans

in case 1 :

[7, 2, 5, 10, 8] → sum of entire array

= 32

in case 2 :

→ Ans for this = max element in array = 4

max value of ans of question = case 1

min value of ans of question = case 2

minAns = max value in array

maxAns = sum of all values in array

[10, 32]

start = 10 end = 32

mid = s + e / 2 = 42/2 = 21

Try to see if you can split the array with 21 or the maximum sum.

7, 2, 5, 8, 10

Pieces = 2

[7, 2, 5] , [8, 10]

Check 1:

if (pieces <= m) ➡ end = mid

start = 10, end = 21

mid = 15 7, 2, 5, 8, 10

[7, 2, 5] , [8] , [10] Pieces = 3

Check 2:

if (pieces > m) ➡ start = mid + 1

start = 16, end = 21
mid = 18 7, 2, 5, 8, 10

[7, 2, 5] , [8, 10] Pieces = 2

start = 16, end = 18
mid = 17 7, 2, 5, 8, 10

[7, 2, 5] , [8] , [17] Pieces = 3

start = mid + 1 = 18
start = 18, end = 18
mid = 18 mid

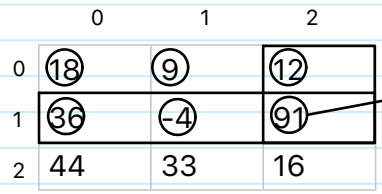
Ans

// the ans exists definitely, hence by the above 2 checks we will reach that ans

Binary Search in 2D Arrays

Saturday, April 23, 2022 11:00 AM

Searching in Matrixes :-



target = 91

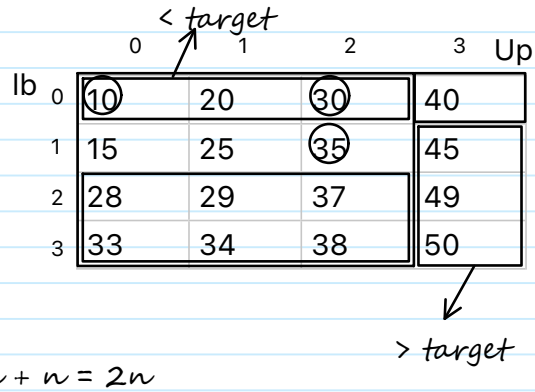
Ans = [1, 2]

N*N = N^2 = $O(N^2)$

complexity = O(N * M)

```
for r = 0; r < n; r++;{
    for c = 0; c < n; c++;{
        if arr[r][c] = target{
            ans;
        }
    }
}
return -1;
```

Matrix is sorted in a row wise & column wise manner :-



target = 37

n + n = 2n

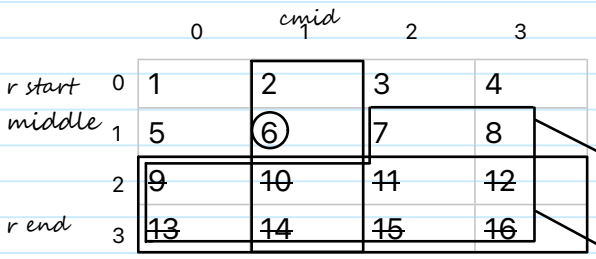
O(n) O(1)

case 1 : if element == target
→ ans found
case 2 : if element < target
→ row++
case 3 : if element > target
→ col--

```
public static void main(String[] args){
        int[][] arr = {
                {10, 20, 30, 40},
                {15, 25, 35, 45},
                {28, 29, 37, 49},
                {33, 34, 38, 50}
        };

        System.out.println(Arrays.toString(search(arr, 37)));
    }

    static int[] search(int[][] matrix, int target){
        int r = 0;
        int c = matrix.length - 1;

        while (r < matrix.length && c >= 0){
            if(matrix[r][c] == target){
                return new int[]{r, c};
            }
```

```
if(matrix[r][c] < target){
                r++;
            } else {
                c--;
            }
        }
        return new int[]{-1, -1};
    }
```

Search in a sorted matrix:



target = 2

- Take middle column & perform BS on it

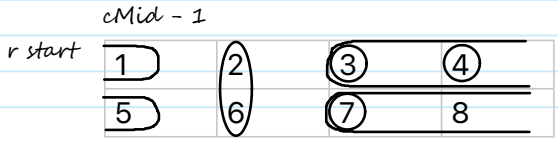
> mid = 6

6 > 2

> 2

1. if element == target
   // ans
2. if element > target
   // Ignore rows after it
3. if element < target
   // Ignore above rows

In the end 2 rows are remaining :-



$$O(\log(N) + \log(M))$$

target = 3

1. Check whether the mid column you are at contains the ans i.e. [2, 6]

2. Consider the four parts

```
public class SortedMatrix {
    public static void main(String[] args) {
        int[][] arr = {
                {1, 2, 3},
                {4, 5, 6},
                {7, 8, 9}
        };
        System.out.println(Arrays.toString(search(arr, 9)));
    }
```

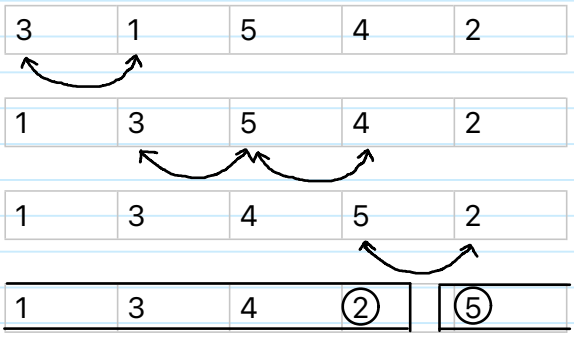
```
// search in the row provided between the cols provided
static int[] binarySearch(int[][] matrix, int row, int cStart, int
    cEnd, int target) {
    while (cStart <= cEnd) {
        int mid = cStart + (cEnd - cStart) / 2;
        if (matrix[row][mid] == target) {
            return new int[]{row, mid};
        }
        if (matrix[row][mid] < target) {
            cStart = mid + 1;
        } else {
            cEnd = mid - 1;
        }
    }
    return new int[]{-1, -1};
}

static int[] search(int[][] matrix, int target) {
    int rows = matrix.length;
    int cols = matrix[0].length; // be cautious, matrix may be empty
    if (cols == 0){
        return new int[] {-1,-1};
    }
    if (rows == 1) {
        return binarySearch(matrix,0, 0, cols-1, target);
    }

    int rStart = 0;
    int rEnd = rows - 1;
    int cMid = cols / 2;

    // run the loop till 2 rows are remaining
    while (rStart < (rEnd - 1)) { // while this is true it will have
                                  more than 2 rows
        int mid = rStart + (rEnd - rStart) / 2;
        if (matrix[mid][cMid] == target) {
            return new int[]{mid, cMid};
        }
        if (matrix[mid][cMid] < target) {
            rStart = mid;
        } else {
            rEnd = mid;
        }
    }

    // now we have two rows
    // check whether the target is in the col of 2 rows
    if (matrix[rStart][cMid] == target) {
        return new int[]{rStart, cMid};
    }
    if (matrix[rStart + 1][cMid] == target) {
        return new int[]{rStart + 1, cMid};
    }
```

```
            // search in 1st half
            if (target <= matrix[rStart][cMid - 1]) {
                return binarySearch(matrix, rStart, 0, cMid-1, target);
            }

            // search in 2nd half
            if (target >= matrix[rStart][cMid + 1] && target <=
                matrix[rStart][cols - 1]) {
                return binarySearch(matrix, rStart, cMid + 1, cols - 1,
                    target);
            }

            // search in 3rd half
            if (target <= matrix[rStart + 1][cMid - 1]) {
                return binarySearch(matrix, rStart + 1, 0, cMid-1, target);
            } else {
                return binarySearch(matrix, rStart + 1, cMid + 1, cols - 1,
                    target);
            }
        }
    }
```

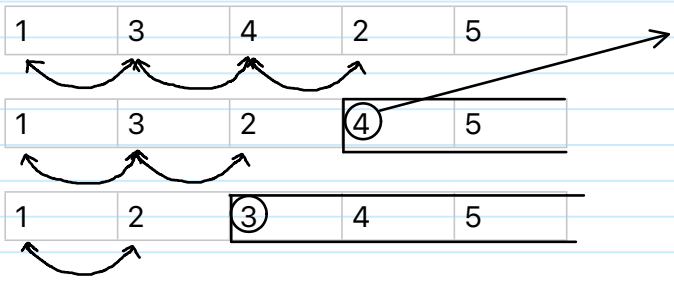
Output :- [2, 2]

Bubble Sort

Monday, April 25, 2022 10:30 AM

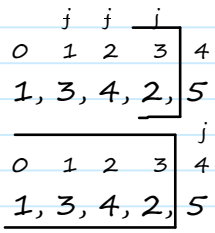


Why?
With the first pass through the array, the largest element came to the end.



With pass no: 2
2nd larget element is at the 2nd from last ended

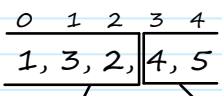
for i = 1 // second pass



< length - 1
= 5 - 1 = 4

or

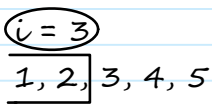
for i = 2 // third pass



<= len - i - 1
length - 1 = 5 - 2
= 3

j will only check this because

This is already sorted



Space complexity = O(1) // constant // No extra space required

also known as inplace sorting algorithmes

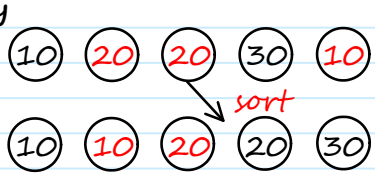
i.e. copying the orray, etc. not required

Time complexity :- Best case : $O(N)$ → sorted
Worst case : $O(N^2)$ → sorted in opposite

- As the size of array is growing, the no. of comparison is also growing

Total comparison = $(N-1) + (N-2) + (N-3) + (N-4)$
$= 4N - (1 + 2 + 3 + 4)$
$= 4N - (N * (N+1))/2$
$= 4N - (N^2 + N)/2$
$= (8N - N^2 - N)/2$
$=(7N - N^2)/2$
$= O(N^2)$

Stability



in original array black ball of 10 nos before red ball of 10. And in the sorted arr, this order if maintained.

10, 10, 20, 20, 30 → Unstable

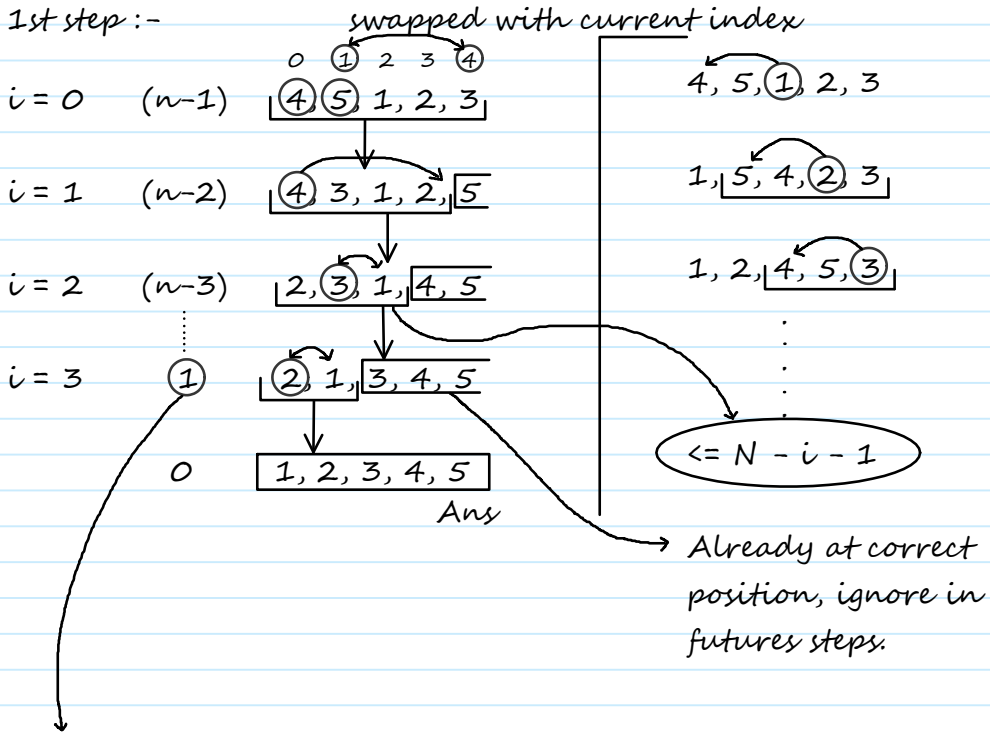
```
public static void main(String[] args) {
        int[] arr = {5, 3, 4, 1, 2};
        insertion(arr);
        System.out.println(Arrays.toString(arr));
    }

    static void insertion(int[] arr) {
        for (int i = 0; i < arr.length - 1; i++) {
            for (int j = i+1; j > 0; j--) {
                if (arr[j] < arr[j-1]) {
                    swap(arr, j, j-1);
                } else {
                    break;}
            }
        }
    }

    static void selection(int[] arr) {
        for (int i = 0; i < arr.length; i++) {
            // find the max item in the remaining array and swap with
            correct index
            int last = arr.length - i - 1;
            int maxIndex = getMaxIndex(arr, 0, last);
            swap(arr, maxIndex, last);
        }
    }
```

```
static void swap(int[] arr, int first, int second) {
        int temp = arr[first];
        arr[first] = arr[second];
        arr[second] = temp;
    }

    static int getMaxIndex(int[] arr, int start, int end) {
        int max = start;
        for (int i = start; i <= end; i++) {
            if (arr[max] < arr[i]) {
                max = i;
            }
        }
        return max;
    }

    static void bubble(int[] arr) {
        boolean swapped;
        // run the steps n-1 times
        for (int i = 0; i < arr.length; i++) {
            swapped = false;
            // for each step, max item will come at the last respective index
            for (int j = 1; j < arr.length - i; j++) {
                // swap if the item is smaller than the previous item
                if (arr[j] < arr[j-1]) {
                    // swap
                    int temp = arr[j];
                    arr[j] = arr[j-1];
                    arr[j-1] = temp;
                    swapped = true;
                }
            }
            // if you did not swap for a particular value of i, it means the
            array is sorted hence stop the program
            if (!swapped) { // !false = true
                break;
            }
        }
    }
}
```

Output :-

```
[1, 2, 3, 4, 5]
```

Selection Sort :-



Total comparisons :- $0 + 1 + 2 + ... + (n-1)$

$((n-1) * (n-1+1))/2$

$(n(n-1))/2$

$O(n^2)$

why constants removed?

why less dominating term remove??

Worst case = $O(N^2)$

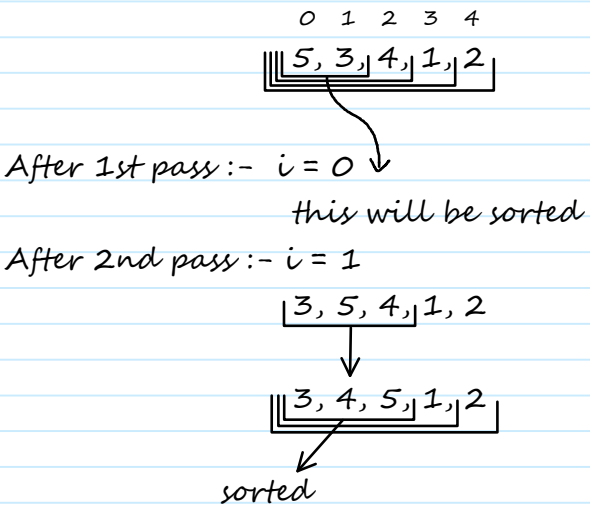
Best case = $O(N^2)$

Stable = No

// It performs well on small lists / arrays

Time Complexity

Insertion Sort :-



For every index :-

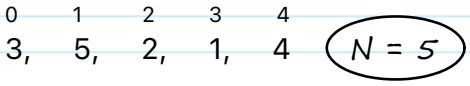
Put that index element at the correct index of LHS.

Cycle sort

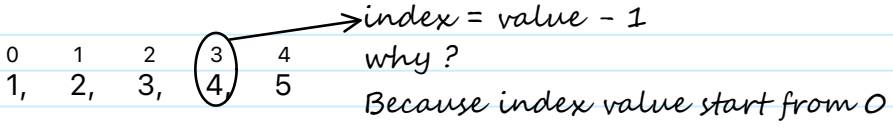
Sunday, May 29, 2022 11:00 AM

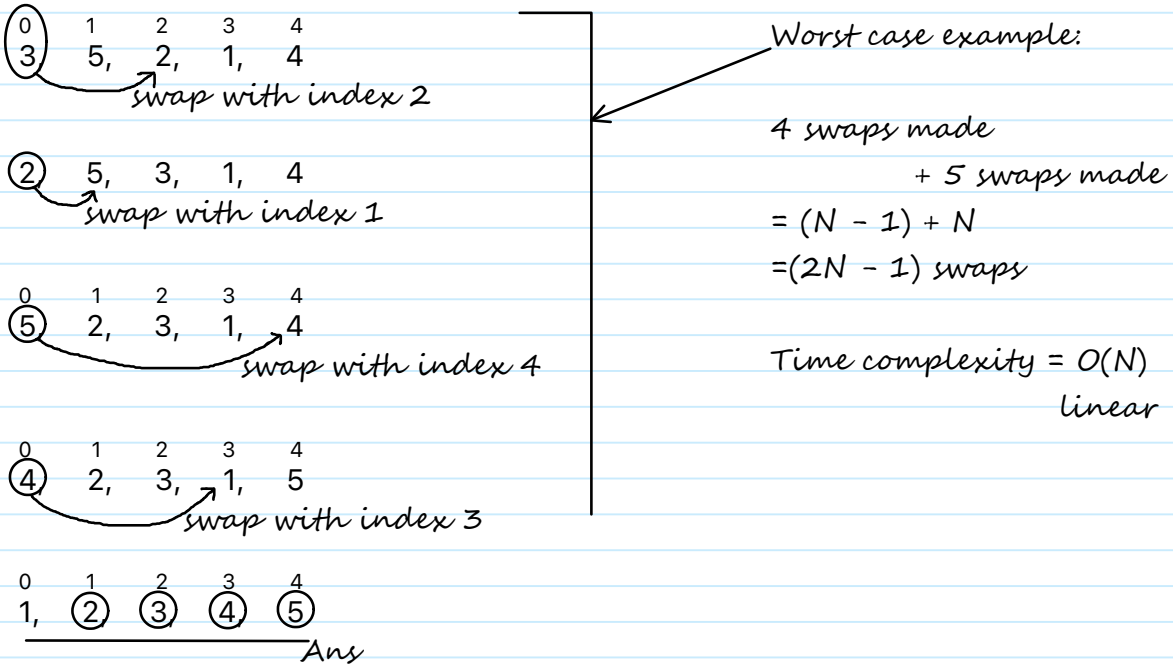
When given numbers from range 1 to N → use cycle sort

e.g :



After Sorting:






```
public class CyclicSort {
    public static void main(String[] args) {
        int[] arr = {3, 5, 2, 1, 4};
        sort(arr);
        System.out.println(Arrays.toString(arr));
    }

    static void sort(int[] arr) {
        int i = 0;
        while (i < arr.length) {
            int correct = arr[i] - 1;
            if (arr[i] != arr[correct]) {
                swap(arr, i , correct);
            } else {
                i++;
            }
        }
    }

    static void swap(int[] arr, int first, int second) {
        int temp = arr[first];
```

```
            arr[first] = arr[second];
            arr[second] = temp;
        }
    }
```

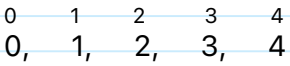
Output :-

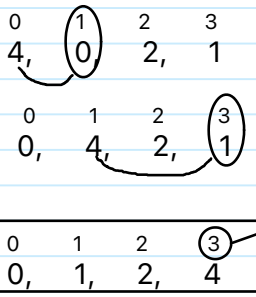
```
[1, 2, 3, 4, 5]
```

Numbers from 0 till N → Total there will be N+1 numbers.

Ex: N=4, arr=[4, 0, 2, 1]

case 1: [0,1,2,3,4] → Here, you can see that in sorted version,
element == index





index(No:4) does not exist?
= Ignore

case 2: when n is not there in the array.

N=4 arr = [1, 0, 3, 2] → [0, 1, 2, 3]
0 1 2 3
Ans = N

Tips :-

- If range → [0, N]
  every element will be at index = value
- If ravge → [1, N]
  every element will be at index = value - 1

Missing Number

https://leetcode.com/problems/missing-number/

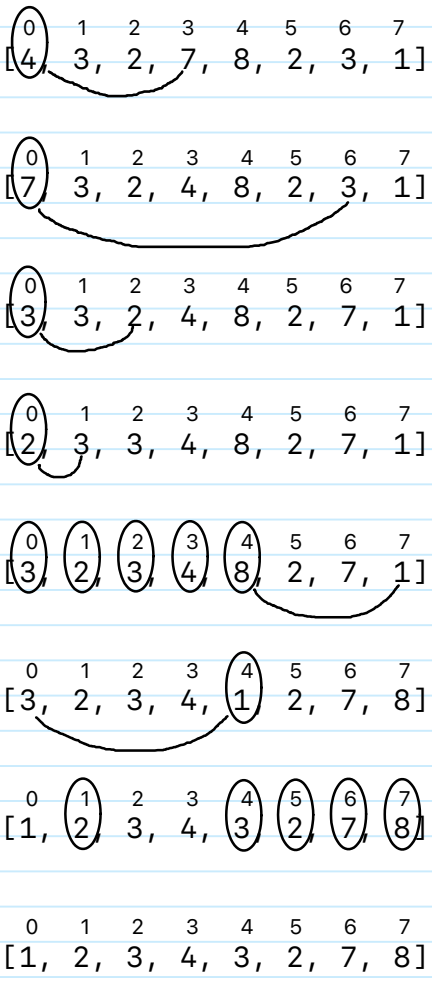
```
public static void main(String[] args) {
    int[] arr = {4, 0, 2, 1};
    System.out.println(missingNumber(arr));
}

public static int missingNumber(int[] arr) {
    int i = 0;
    while (i < arr.length) {
        int correct = arr[i];
        if (arr[i] < arr.length && arr[i] != arr[correct]) {
            swap(arr, i , correct);
        }
```

```
            else {
                i++;
            }
        }
        // search for first missing number
        for (int index = 0; index < arr.length; index++) {
            if (arr[index] != index) {
                return index;
            }
        }
        // case 2
        return arr.length;
    }

    static void swap(int[] arr, int first, int second) {
        int temp = arr[first];
        arr[first] = arr[second];
        arr[second] = temp;
    }
}
```

Output :- 3

Find All Numbers Disappeared in an Array

N = 8



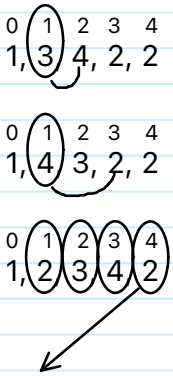
5, 6

Ans

https://leetcode.com/problems/find-all-numbers-disappeared-in-an-array/

```
public List<Integer> findDisappearedNumbers(int[] nums) {
        int i = 0;
        while (i < nums.length) {
            int correct = nums[i] - 1;
            if (nums[i] != nums[correct]) {
                swap(nums, i , correct);
            } else {
                i++;
            }
        }

        // just find missing numbers
        List<Integer> ans = new ArrayList<>();
        for (int index = 0; index < nums.length; index++) {
            if (nums[index] != index+1) {
                ans.add(index + 1);
            }
        }

        return ans;
    }

    static void swap(int[] arr, int first, int second) {
        int temp = arr[first];
        arr[first] = arr[second];
        arr[second] = temp;
    }
```

Find the Duplicate Number



if element != index + 1

2 things :-

if element of value - 1

i.e. (2 - 1 = 1)

if

!= element at current index ➡ swap

else

you have found the answer

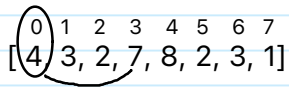
https://leetcode.com/problems/find-the-duplicate-number/

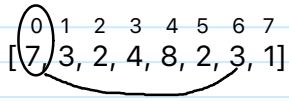
```
public static void main(String[] args){
        int[] arr = {1, 3, 4, 2, 2};
        System.out.println(findDuplicate(arr));
    }

static int findDuplicate(int[] arr) {
        int i = 0;
        while (i < arr.length) {

            if (arr[i] != i + 1) {
                int correct = arr[i] - 1;
                if (arr[i] != arr[correct]) {
                    swap(arr, i , correct);
                } else {
                    return arr[i];
                }
            } else {
                i++;
            }
        }
        return -1;
    }

static void swap(int[] arr, int first, int second) {
        int temp = arr[first];
        arr[first] = arr[second];
        arr[second] = temp;
    }
```

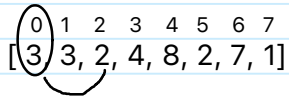
Output :-

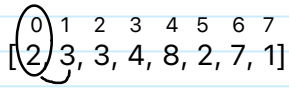
```
2
```

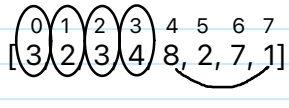
Find all duplicates in an array :-

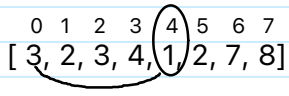


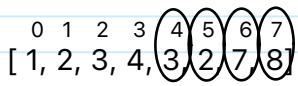


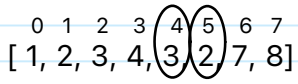












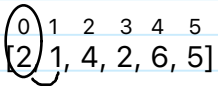
Ans = 2, 3

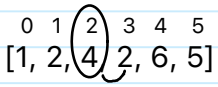
https://leetcode.com/problems/find-all-duplicates-in-an-array/

```
public List<Integer> findDuplicates(int[] arr) {
    int i = 0;
    while (i < arr.length) {
        int correct = arr[i] - 1;
        if (arr[i] != arr[correct]) {
            swap(arr, i , correct);
        } else {
            i++;
        }
    }

    List<Integer> ans = new ArrayList<>();
    for (int index = 0; index < arr.length; index++) {
        if (arr[index] != index+1) {
            ans.add(arr[index]);
        }
    }

    return ans;
}

static void swap(int[] arr, int first, int second) {
    int temp = arr[first];
    arr[first] = arr[second];
    arr[second] = temp;
}
```

Set Mismatch

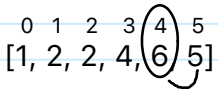
N = 6

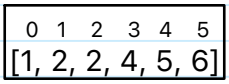




1 no. is missing

1 no. is repeating





missing = index + 1

duplicate = no. at the missing index

https://leetcode.com/problems/set-mismatch/

```
public static void main(String[] args)
{
    int[] arr = {2, 1, 4, 2, 6, 5};
    int[] ans = findErrorNums(arr);
    System.out.println(Arrays.toString(ans));
}

static int[] findErrorNums(int[] arr)
{
    int i = 0;
    while (i < arr.length)
    {
        int correct = arr[i] - 1;
        if (arr[i] != arr[correct])
        {
            swap(arr, i , correct);
        }
        else
        {
            i++;
        }
    }

    // search for first missing number
    for (int index = 0; index < arr.length; index++)
    {
        if (arr[index] != index + 1)
        {
            return new int[] {arr[index], index+1};
        }
    }

    return new int[] {-1, -1};
}

static void swap(int[] arr, int first, int second)
{
    int temp = arr[first];
    arr[first] = arr[second];
    arr[second] = temp;
}
```

Output :-

[2, 3]

First Missing Positive :-

3, 4, -1, 1

↓

0 1 2 3

[-1, 1, 3, 4]

Ans = 2

[1, 2, 0] → [0, 1, 2] Ans = 3

Note :

Ignore -ve because +ve no are asked

✧ start checking from 1.