**Data Structure :** A **data structure** is a particular way of organizing data in a computer so that it can be used effectively.

**Array :** An array may be defined as the collection of similar types of elements with contiguous memory allocation.In Java, all the arrays are dynamically allocated and by default, each element is 0 if number, false if Boolean and null for reference types.

**Advantages of Array:**

* It allows random access of elements, hence, access of elements are easy here.

**Disadvantages of Array:**

* We can’t change the size of the array because of the static memory allocation.
* Insertion and deletions are difficult because of contiguous memory allocation.

**Program for array rotation :**

def rotate(L,d,n):

k=L.index(d)

new\_lis=[]

new\_lis=L[k+1:]+L[0:k+1]

return new\_lis

d = 2

n = 7

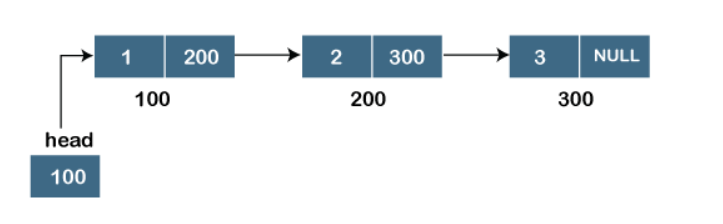
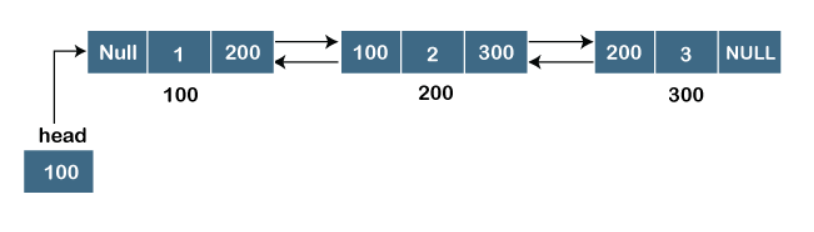
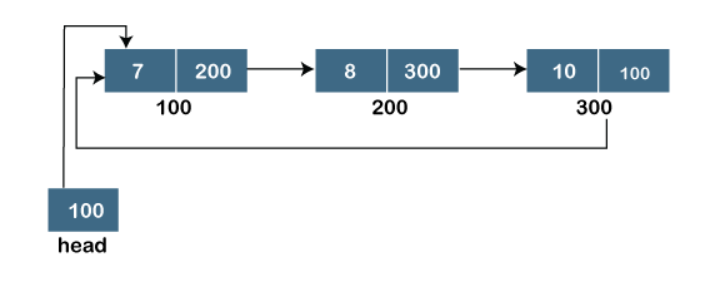
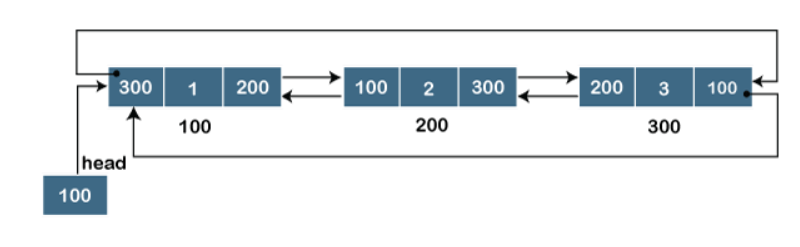
print(rotate([1,2,3,4,5,6,7],d,n))

**Linked List :** Linked list is a linear data structure , without contiguous memory allocations. It uses pointers for memory allocation.

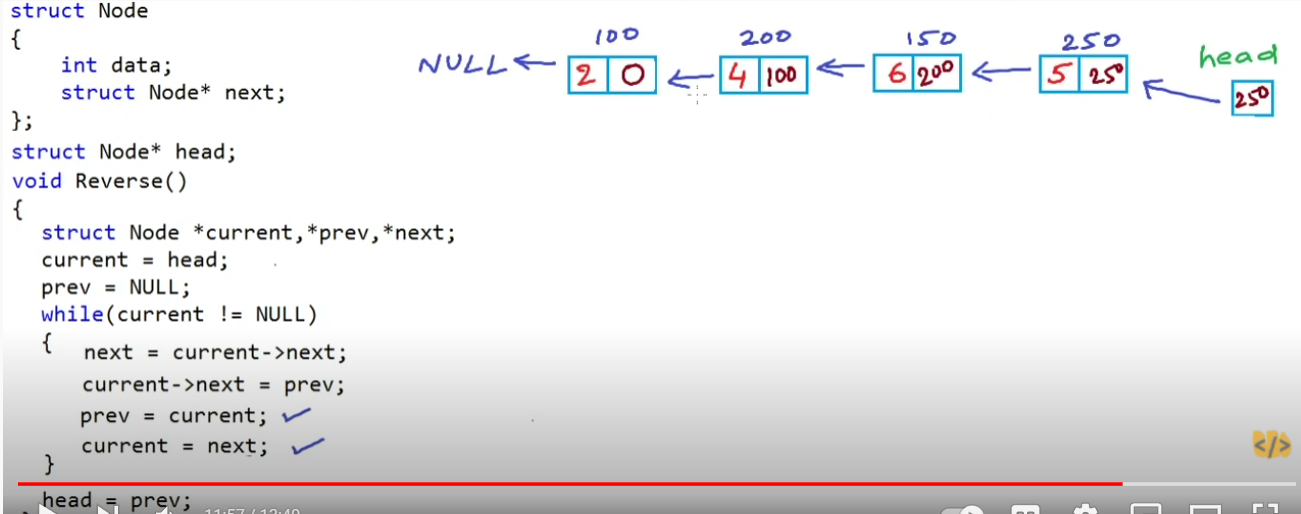
**Advantages over array:**

* Dynamic Size
* Ease of insertion and deletion

**Types of Linked List :**

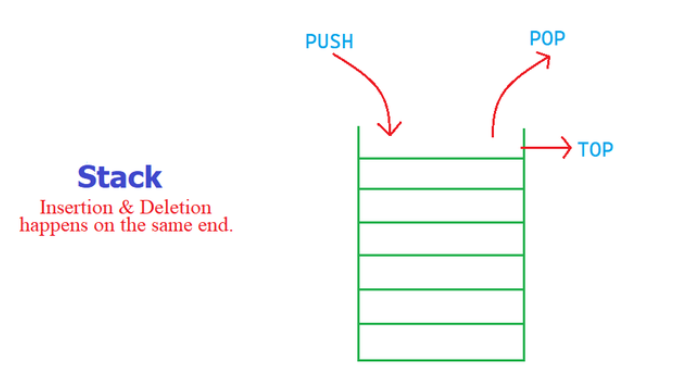
* Singly Linked List
  + 
* Doubly Linked List
  + 
* Circular Linked List
  + 
* Doubly Circular Linked List
  + 

**Reversing a linked list**



**Stack :**

A Stack is a linear data structure that follows a particular order in which operations are performed.It is known as LIFO (Last in First Out)



* Push(),pop(),peek(),isEmpty() all take O(1) time.
* Stack<Integer> stack = new Stack();

Applications :

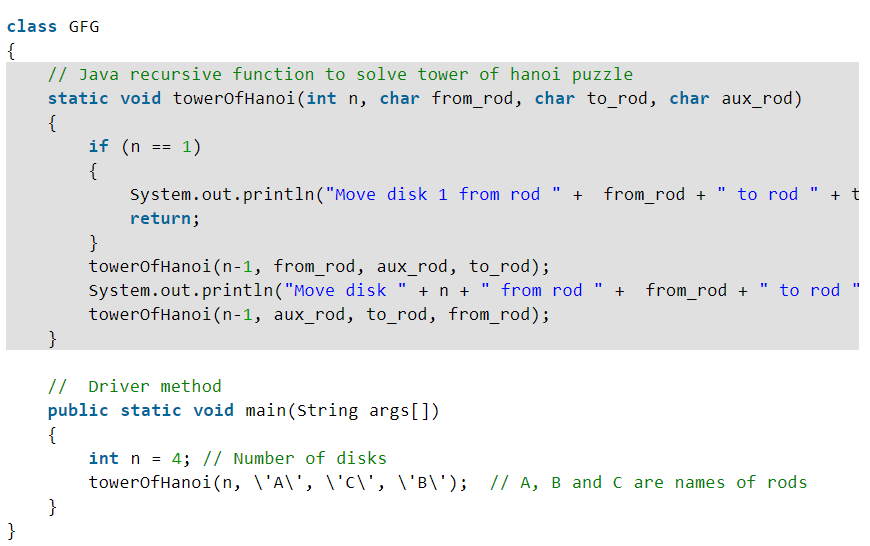
* Reversing a list
* Recursion
* Conversion of infix to postfix expression
* Tower of Hanoi
* Parenthesis Checker

Infix : Operator is placed between operands. A+B = A+B

Postfix : Operator is placed after operands . A+B = AB+

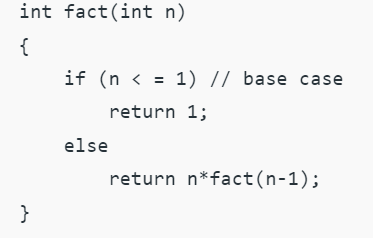
Prefix : Operator is placed before operands. A+B = +AB

Tower of Hanoi : move out all the disks from tower A to tower C by maintaining the same order.



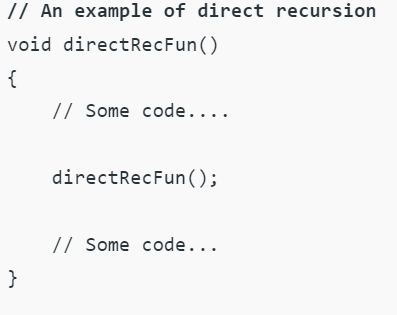
**Recursion :**

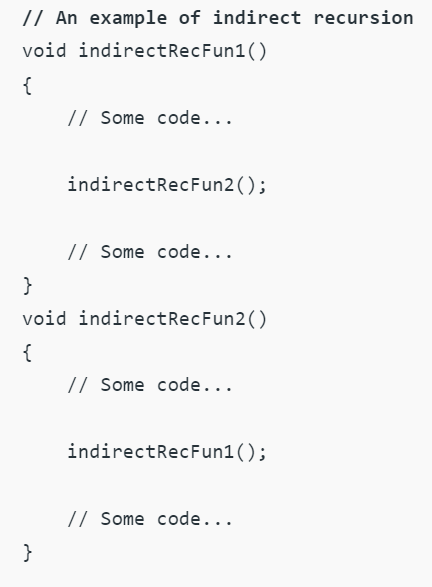
The process in which the functions calls itself directly or indirectly is called recursion. In recursive program, the solution to the base case is provided and the solution to the bigger problem is expressed in terms of smaller problems



Direct and Indirect Recursion :

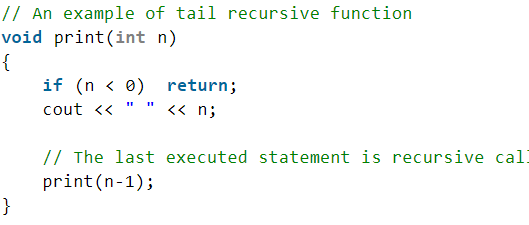
A function is called direct recursion if it calls itself again and again, but it is called indirect recursion if it calls another method to call itself.





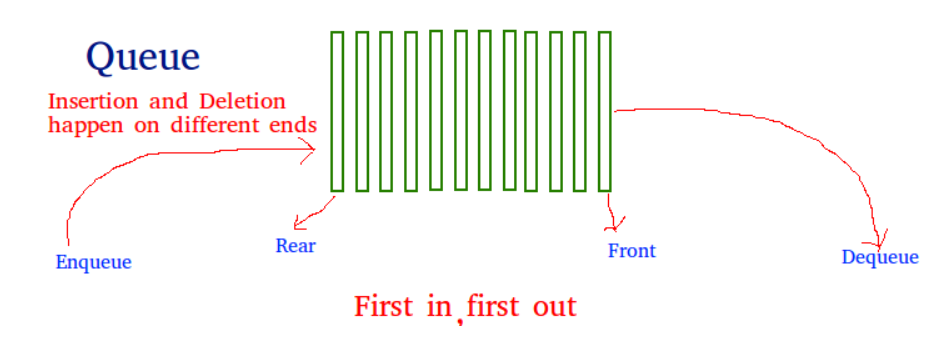
Tail recursion :

A recursion is called tail recursion if the recursive call is the last thing executed by the function.



**Queue :**

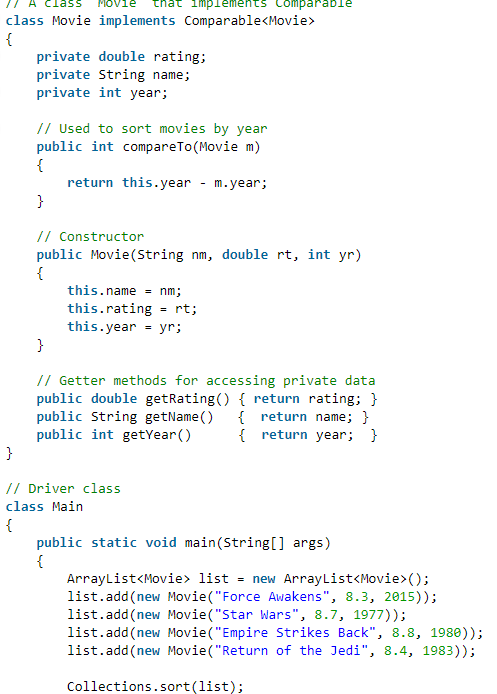
A queue is a linear data structure which follows a particular order in which operations are performed. It is FIFO (First In First Out). Addition of elements are done at rear end and deletion of elements are done at front end.

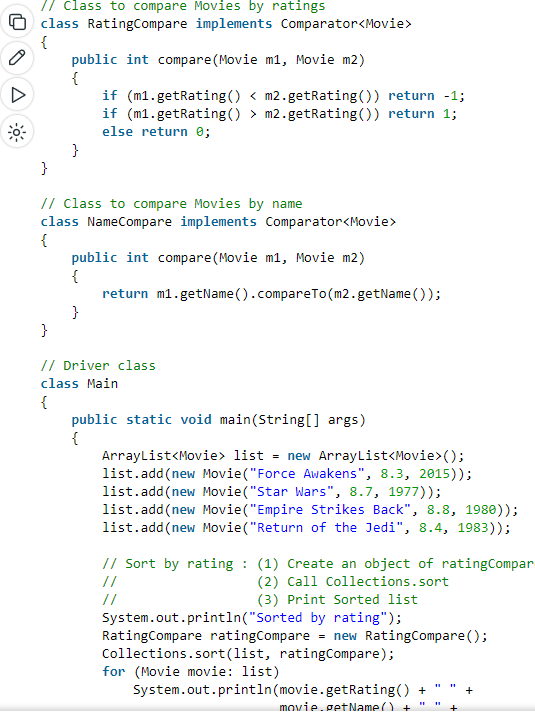


* Queue<Integer> queue = new PriorityQueue<>();
* Queue<Integer> queue = new LinkedList<>();

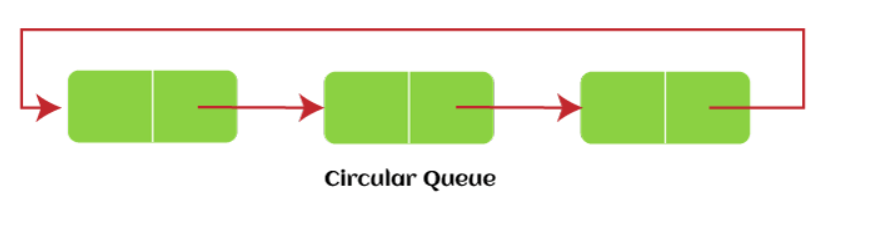
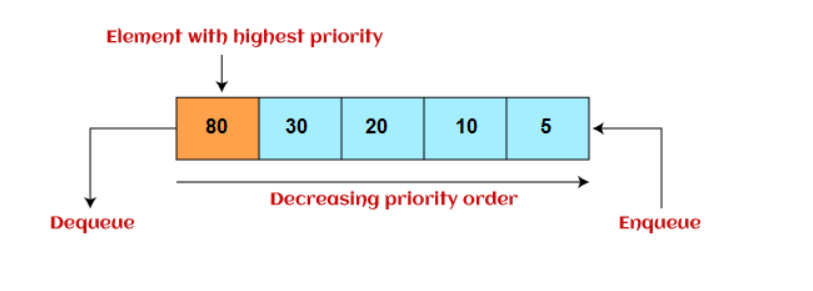
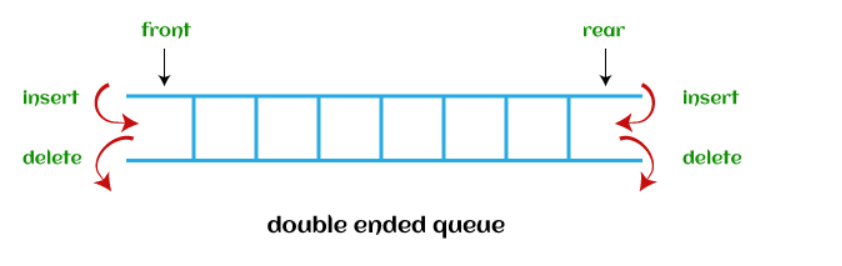
Comparable and Comparator:

Comparable is internal to the element type we are comparing while comparator is external to the element type we are comparing.





Types of Queues :

* Circular Queue : It is almost same as the normal queue only but the last element is connected to the first element.
  + 
* PriorityQueue : It is the type of queue where the elements with highest priority come first.
  + 
* Deque or Double Ended Queue : In this queue, the insertion and deletion are done from both the ends. Deque can be used as a palindrome checker.
  + 

**Tree**

A tree is a non-linear data structure which stores the data more efficiently.

**Binary Tree :**

A tree with atmost 2 children is called binary tree.

**Binary Search Tree :**

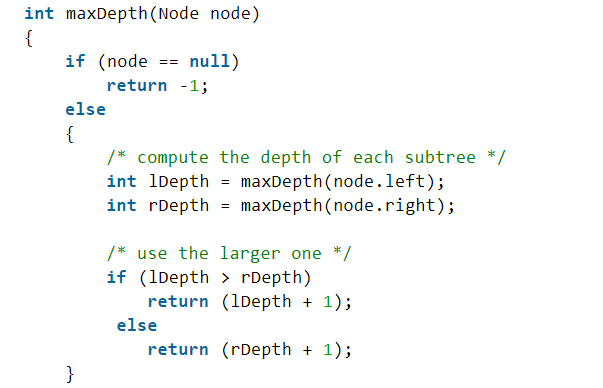
A binary tree with the left node smaller than the root node and the right node greater than the root node is called BST.

**Tree Traversal :**

* Preorder
* Postorder
* Inorder

\*\* Deletion from Binary Tree : there happens problem in case of a node with two nodes. In that case we replace the value of a current node with the inorder successor (smallest value in the right sub tree) or inorder predecessor (largest value in the left sub tree)

**Height of a tree**



**Threaded Binary Tree :**

Threaded binary tree is a binary tree which is almost same as BST with the only difference is about storing NULL values.

**AVL Tree :**

AVL Tree can be defined as the height balanced tree in which each node is associated with a balancing factor which is calculated by subtracting the height of right subtree from left subtree.The tree is said to be balanced if the balancing factor is from -1 to 1 , else the tree need to be balanced.

Balancing factor = Height(Left) – Height(Right)

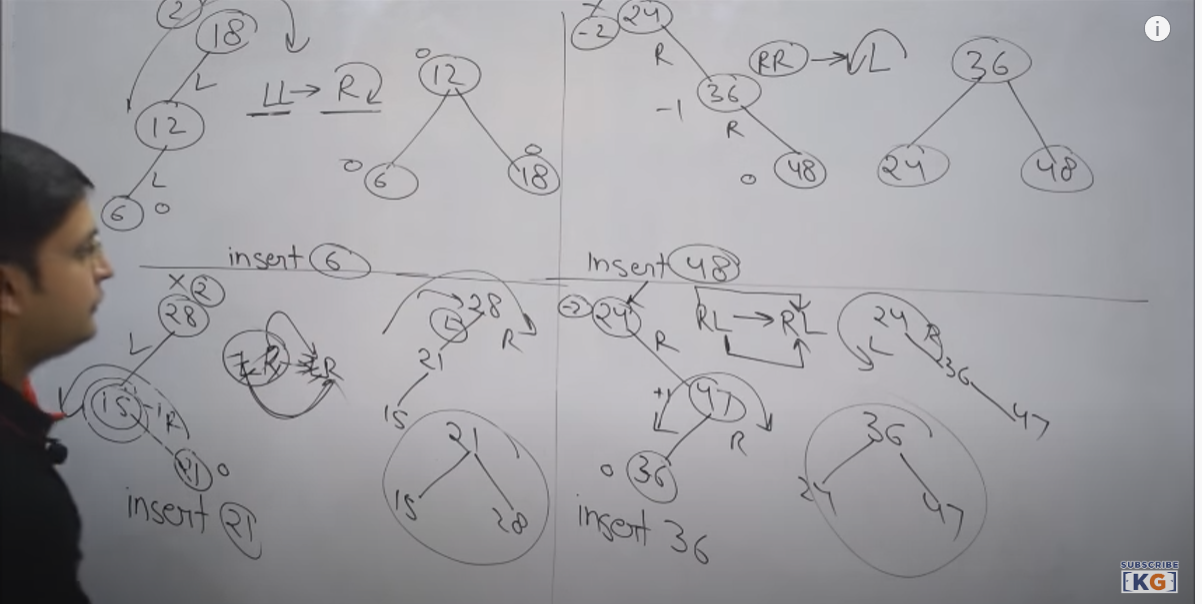
If the balancing factor is 1, it means left subtree is higher to right subtree by level 1

If the balancing factor is 0, both are at the same level.

If the balancing factor is -1, it means the right subtree is higher to left subtree by level 1

It is helpful when the tree is left-skewed or right-skewed.

**Rotations in AVL Tree :**

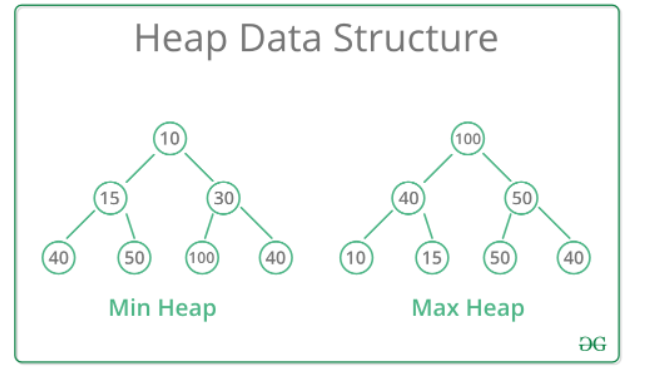


\*\* Insert the node and go towards the root and check if balancing factor for any node is violated. If you find any node, then come toward the newly inserted node and get to know the type of rotations problem it is.

**Heap :**

A heap is a non-linear data structure which is a complete binary tree or almost complete binary tree (only last level not having full children).It can be of two types :

* Max Heap – The value present at root node must be greater than both the child nodes.
* Min Heap – The value present at the root node must be smaller than the both the child nodes.



Heapify : O(log n)

Extract min or max : O(1)

Insertion or deletion : O(log n)

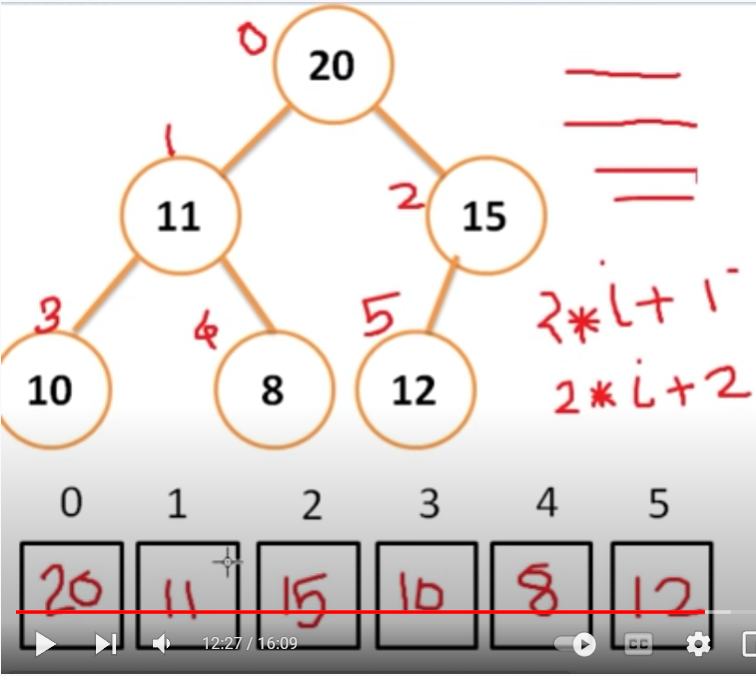
Building a heap : O(n) [ It is because the initial height of the tree is small only ]

Heap Sort : O(n log n)

Heap initialisaiton :

PriorityQueue<Integer> queue = new PriorityQueue<Integer>(Collections.reverseOrder());

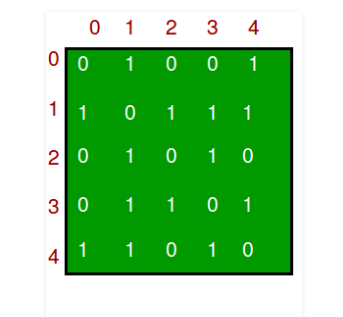
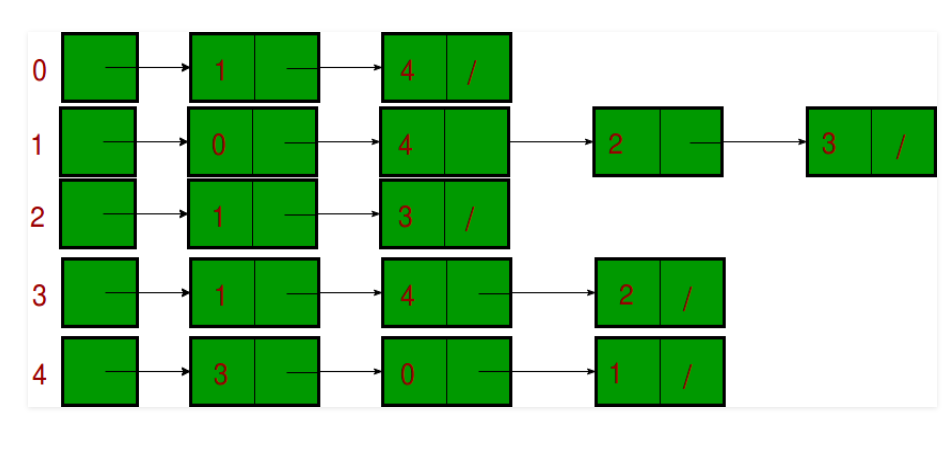
All the elements in the sequence will be the node of the heap.



**Graph :**

A graph is a non-linear data structure which consists of nodes and edges.

A graph can be represented in two ways :

* Adjacency matrix :
  + 
* Adjacency List :
  + 

**Algorithms :**

**Asymptotic Notations :**

They are basically used to analyse the performance of the algorithm. There are three types of asymptotic notations.

* Theta Notation : It provides the exact asymptotic behaviour. It covers the graph from both above and below.
* Big O Notation : It provides the worst case behaviour of the algorithm. It covers the graph from above only.
* Omega Notation : It provides the best case complexity of the algorithm. It covers the graph from below only.

**Graph Traversal Algorithms :**

**BFS (Breadth First Search) :**

It is a graph traversal algorithm in which all the nodes are traversed breadth wise. The graph can contain cycles, hence, we use a visited array. This is implemented using queue.

The time complexity of BFS is O(V+E) when adjacency list is used and O(V^2) when adjacency matrix is used.

**DFS (Depth First Search)**

It is a graph traversal algorithm in which all the nodes are traversed depth-wise. The graph can contain cycles, hence, we use a visited array. This is implemented using stack.

The time complexity of BFS is O(V+E) when adjacency list is used and O(V^2) when adjacency matrix is used.

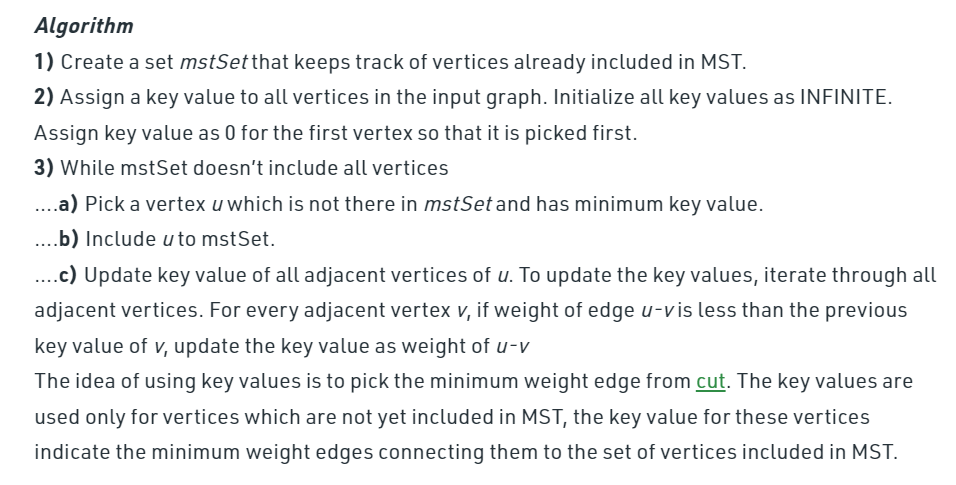
**Spanning Tree :**

A spanning tree is a tree which connects all the vertices of the graph with minimum possible number of edges. And a spanning tree whose cost is minimum, it is called minimum cost spanning tree.

**Minimum Cost Spanning Tree Algorithms :**

**Prim’s Algorithm :**

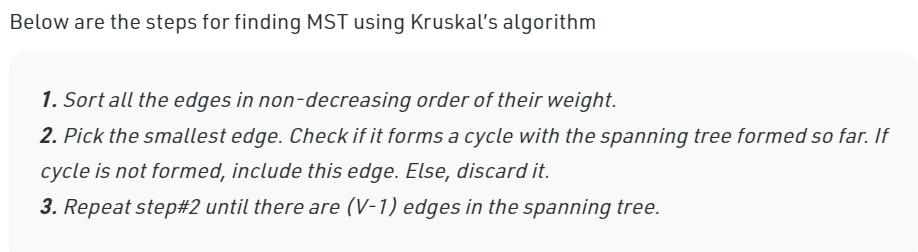
Prim’s algorithm is a greedy approach. It starts with an empty spanning tree. The idea is to maintain two sets of vertices. The first set contains the vertices already included in the MST and the other set contains the vertices not included yet. At every step, it considers all the edges that connect the two sets and pick the minimum edge weight from the edges.



The time complexity of this algorithm is O(V^2) when adjacency matrix is used and when adjacency list as well as binary heap is used, then the complexity reduces to O( (V+E) log V).

**Krushkal’s Algorithm :**

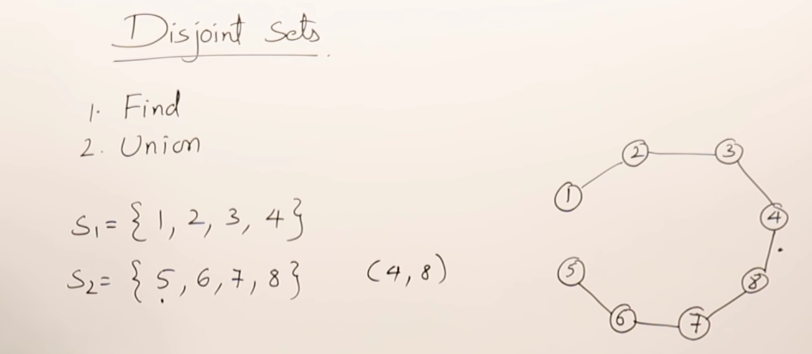
It is also a greedy algorithm. It is based on Union-Find Algorithm to detect cycles .



The time complexity of Krushkal’s algorithm is O(ElogV).

For fully connected graph , E = V^2.

**Disjoint Sets :**



**Single Source Shortest Path Algorithm :**

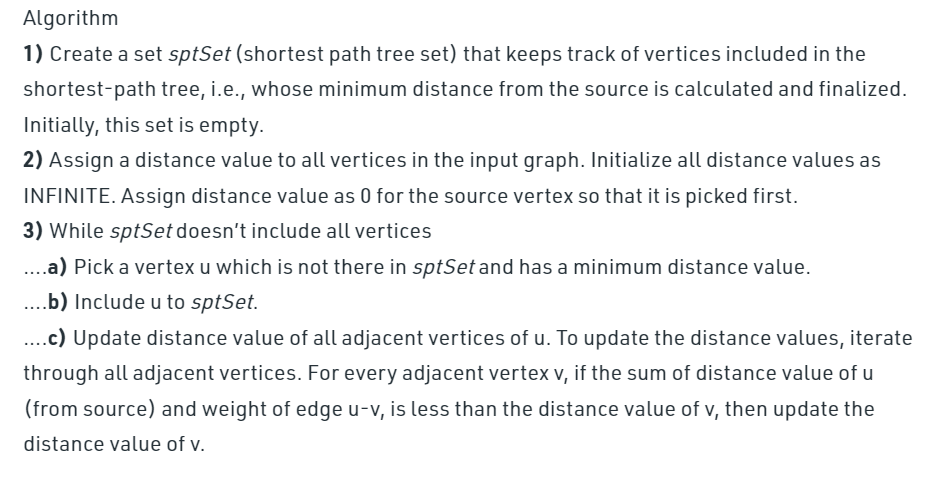
**Dijkstra Algorithm :**

It is also having a greedy approach. It is known as single source shortest path. It is almost similar to Prims algorithms. We first find the shortest path for direct edges , once we find it, we relax all the vertices connected to the shortest path.

Relaxation :

If (d[u] + cost(u,v) < d[v])

d[v] = d[u] + cost(u,v);

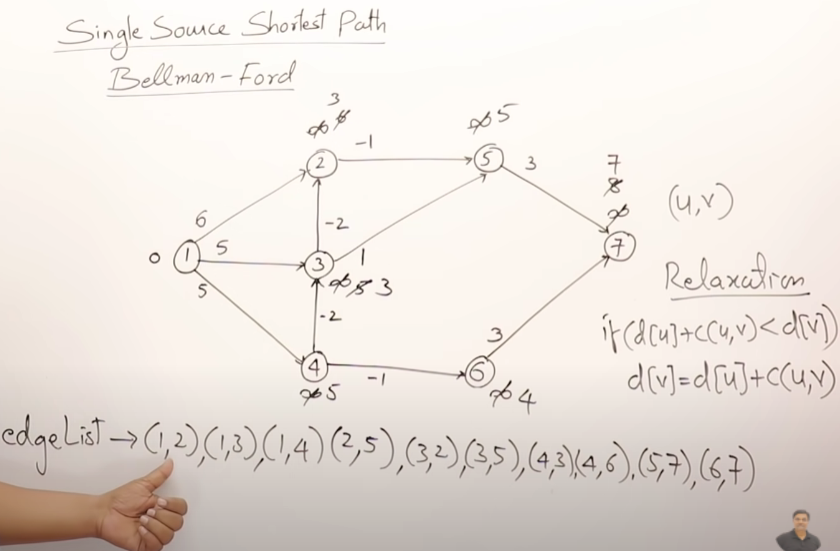


Dijkastra’s algorithm may or may not work for negative edge weights.

The time complexity of Dijkstra’s algorithm is O(V^2) as we may have to traverse V edges and relax V edges in case of fully connected graph.

**Bellman Ford Algorithm :**

To overcome issues with Dijkstra’s algorithm, bellman ford algorithm came into picture. It is a dynamic programming approach. The main idea behind it is we are going to relax all the vertices for n-1 times.

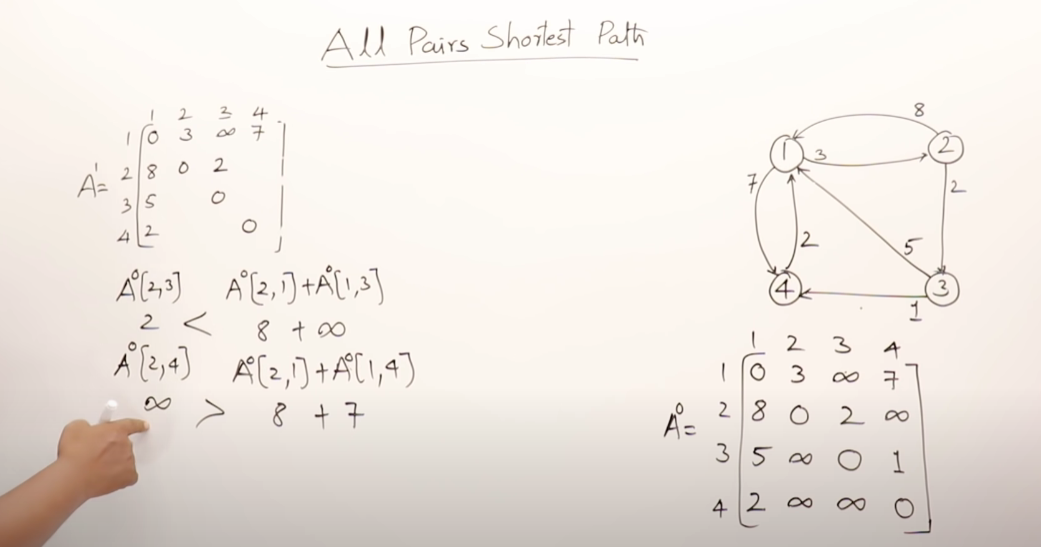


Bellman Ford doesn’t work for negative edge cycles.

The time complexity of bellman’s ford is O(V.E) i.e E = V^2 for fully connected graph i.e O(V^3).

**All Pair Shortest Path (Floyd Warshall ) :**

All pair shortest path can also be solved using Dijkstra’s when applied n times for n vertices. This algorithm is applied with dynamic programming.



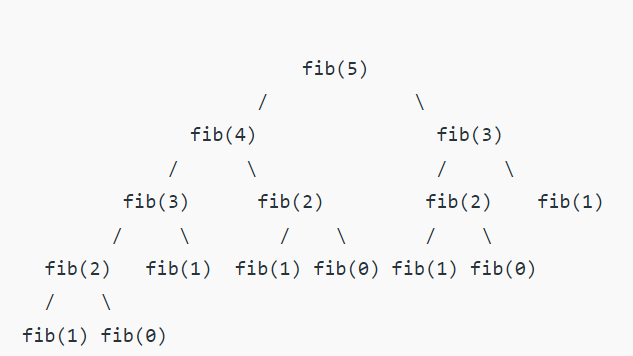
The time complexity for this algorithm is O(V^3) as n intermediate path matrix are generated for n^2 edges.

**Greedy Algorithm :**

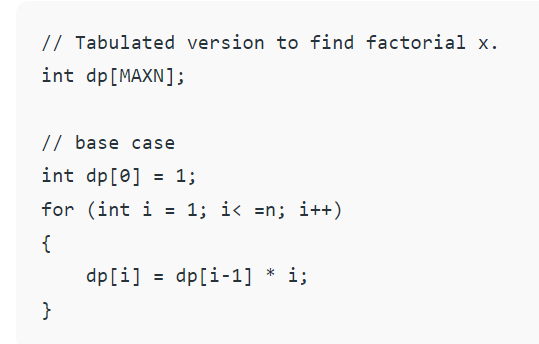
Greedy method is simple and quite straight forward. It is the best technique to solve a specific problem, and thus it is specific to a problem. It is not an algorithm, but a technique , which is used for optimization problems where decisions are taken on the basis of currently available information.

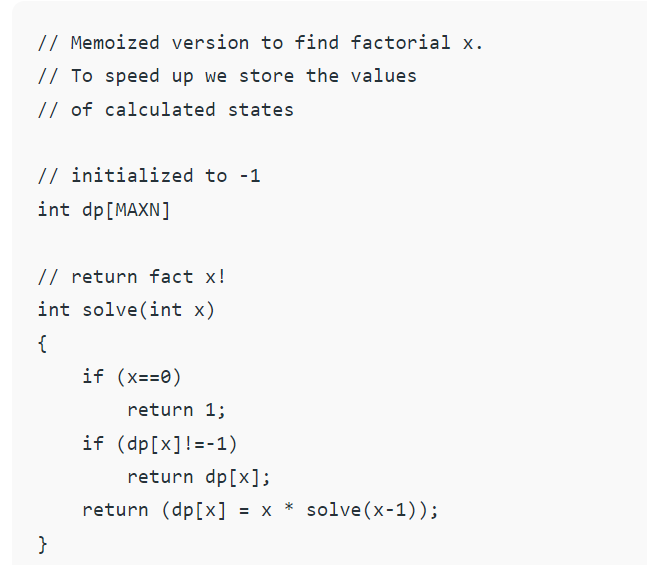
**Dynamic Programming :**

Dynamic Programming is a algorithm paradigm which solves a complex problem by breaking it into smaller sub problems and stores the result of sub problems to avoid computing it again. It helps to solve a problem from exponential time to polynomial time. There are two ways to determine if we can solve a problem using DP :

* Overlapping Subproblems : Ex – Recursive program for Fibonacci
  + 
* Optimal Substructure : Bellman Ford & Floyd Warshall, we have to take decision while relaxing the vertices or while creating intermediate path matrix.

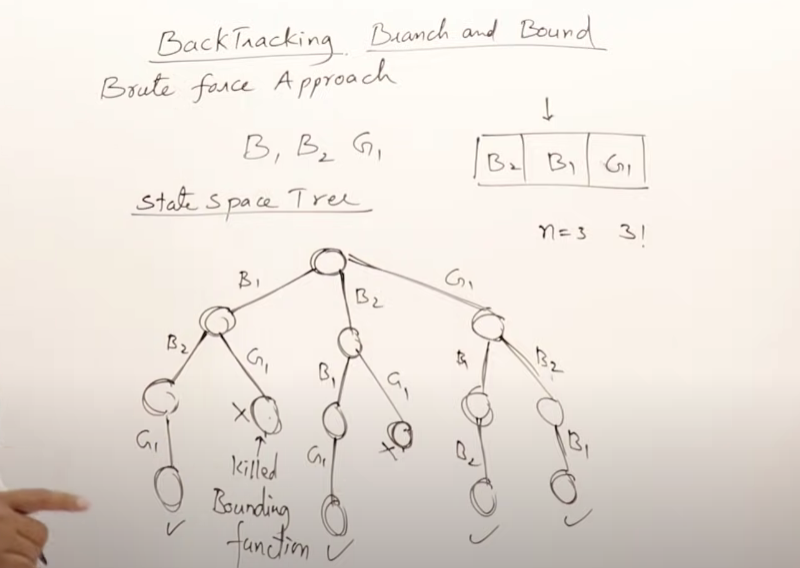
There are two ways to solve overlapping subproblems.

* Tabulation : Bottom Up
  + 
* Memoization : Top Down



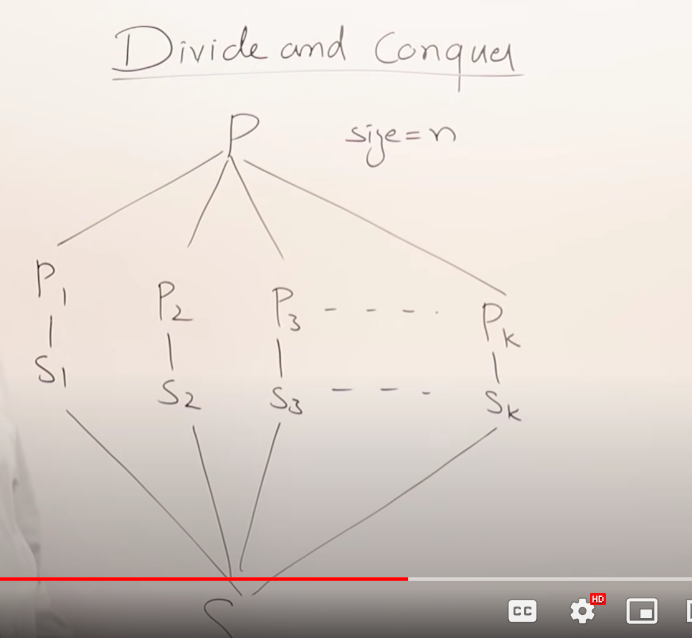
**Backtracking :**

DP is used when there is optimization problem. But backtracking is used when we are having multiple solutions to the problem.

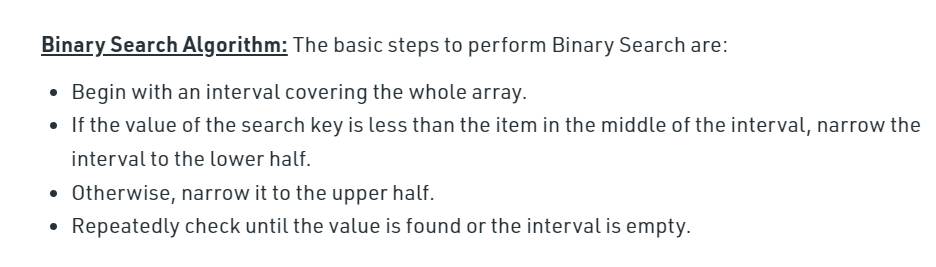


Ex - N Queens Problem, Graph Coloring etc.

**Divide And Conquer :**



**Binary Search :**

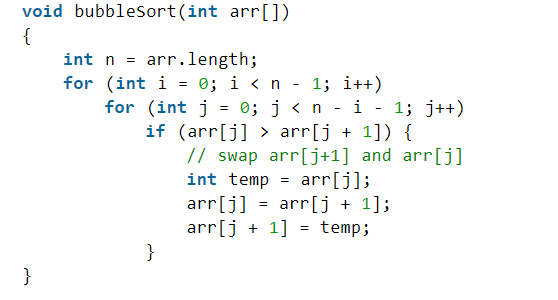


The time complexity of binary search is O(log n).

**Sorting :**

Bubble Sort :

Bubble sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order.

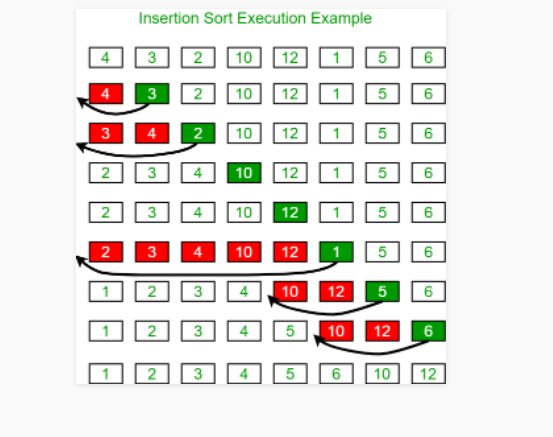


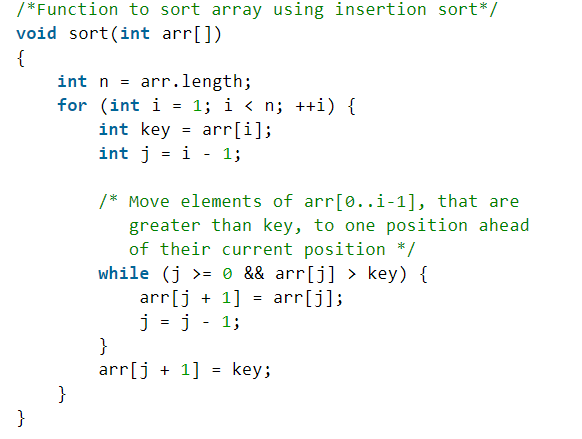
Best case time complexity : O(N) , when the array is already sorted.

Worst case time complexity : O(N^2) , when the array is not sorted.

Insertion Sort :

It is the sorting algorithm which works as the same way we sort playing cards. We pick up the cards and put it at a correct place of all the cards we have in hand.



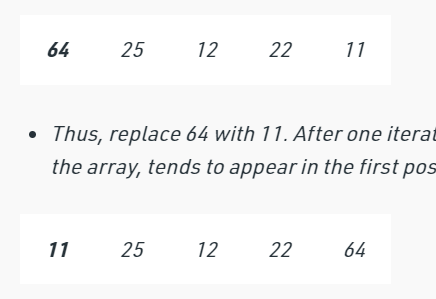


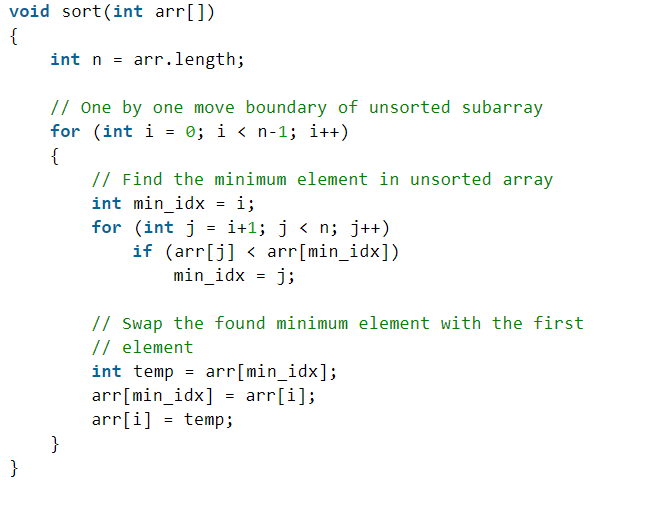
Best case time complexity : O(N) [when the array is sorted]

Worst case time complexity : O(N^2) [when the array is unsorted]

Selection Sort :

In selection sort, we choose the smallest element and put it in the first place and likewise.

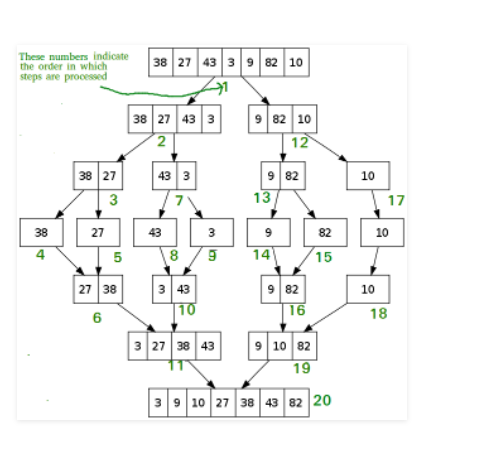




The time complexity is always O(N^2).

Merge Sort :

Merge sort follows divide and conquer algorithm . We first divide the array in two halves and then merge the array.



/\* Java program for Merge Sort \*/

class MergeSort

{

// Merges two subarrays of arr[].

// First subarray is arr[l..m]

// Second subarray is arr[m+1..r]

void merge(int arr[], int l, int m, int r)

{

// Find sizes of two subarrays to be merged

int n1 = m - l + 1;

int n2 = r - m;

/\* Create temp arrays \*/

int L[] = new int[n1];

int R[] = new int[n2];

/\*Copy data to temp arrays\*/

for (int i = 0; i < n1; ++i)

L[i] = arr[l + i];

for (int j = 0; j < n2; ++j)

R[j] = arr[m + 1 + j];

/\* Merge the temp arrays \*/

// Initial indexes of first and second subarrays

int i = 0, j = 0;

// Initial index of merged subarray array

int k = l;

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

}

else {

arr[k] = R[j];

j++;

}

k++;

}

/\* Copy remaining elements of L[] if any \*/

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

/\* Copy remaining elements of R[] if any \*/

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

// Main function that sorts arr[l..r] using

// merge()

void sort(int arr[], int l, int r)

{

if (l < r) {

// Find the middle point

int m =l+ (r-l)/2;

// Sort first and second halves

sort(arr, l, m);

sort(arr, m + 1, r);

// Merge the sorted halves

merge(arr, l, m, r);

}

}

/\* A utility function to print array of size n \*/

static void printArray(int arr[])

{

int n = arr.length;

for (int i = 0; i < n; ++i)

System.out.print(arr[i] + " ");

System.out.println();

}

// Driver code

public static void main(String args[])

{

int arr[] = { 12, 11, 13, 5, 6, 7 };

System.out.println("Given Array");

printArray(arr);

MergeSort ob = new MergeSort();

ob.sort(arr, 0, arr.length - 1);

System.out.println("\nSorted array");

printArray(arr);

}

}

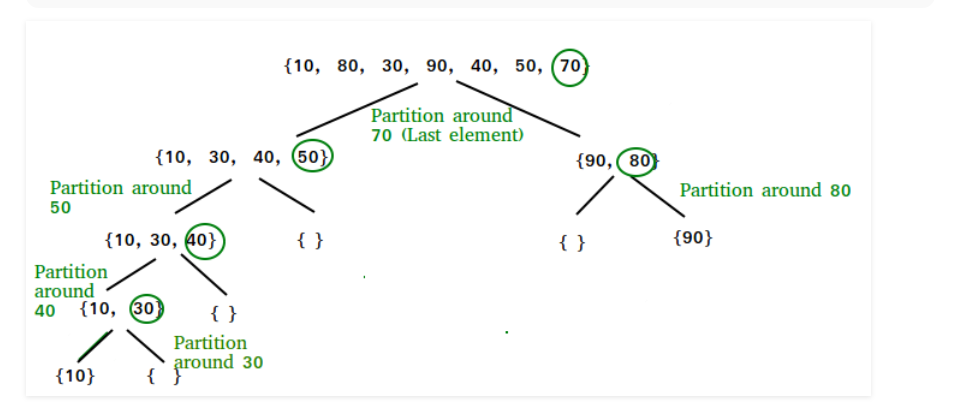
/\* This code is contributed by Rajat Mishra \*/

The time complexity is O(n log n).

Space complexity : O(n)

Quick Sort :

Quick sort is also a divide and conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot.



The best case and average case time complexity is O(nlogn)

The worst case time complexity is O(n^2), when we choose the last element/first element as pivot and the array is already sorted.

// Java implementation of QuickSort

import java.io.\*;

class GFG{

// A utility function to swap two elements

static void swap(int[] arr, int i, int j)

{

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

/\* This function takes last element as pivot, places

the pivot element at its correct position in sorted

array, and places all smaller (smaller than pivot)

to left of pivot and all greater elements to right

of pivot \*/

static int partition(int[] arr, int low, int high)

{

// pivot

int pivot = arr[high];

// Index of smaller element and

// indicates the right position

// of pivot found so far

int i = (low - 1);

for(int j = low; j <= high - 1; j++)

{

// If current element is smaller

// than the pivot

if (arr[j] < pivot)

{

// Increment index of

// smaller element

i++;

swap(arr, i, j);

}

}

swap(arr, i + 1, high);

return (i + 1);

}

/\* The main function that implements QuickSort

arr[] --> Array to be sorted,

low --> Starting index,

high --> Ending index

\*/

static void quickSort(int[] arr, int low, int high)

{

if (low < high)

{

// pi is partitioning index, arr[p]

// is now at right place

int pi = partition(arr, low, high);

// Separately sort elements before

// partition and after partition

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

// Function to print an array

static void printArray(int[] arr, int size)

{

for(int i = 0; i < size; i++)

System.out.print(arr[i] + " ");

System.out.println();

}

// Driver Code

public static void main(String[] args)

{

int[] arr = { 10, 7, 8, 9, 1, 5 };

int n = arr.length;

quickSort(arr, 0, n - 1);

System.out.println("Sorted array: ");

printArray(arr, n);

}

}

// This code is contributed by Ayush Choudhary

