1] Binary search

```
import java.util.*;

public class BI {
    public static int BinarySearch(char arr[], int n, char c) {
        int f = 0;
        int l = n - 1;

        while (f <= l) {
            int mid = f + (l - f) / 2;

            if (arr[mid] == c) {
                return mid; // Changed from 'mid + 1' to 'mid'
            } else if (arr[mid] < c) {
                f = mid + 1;
            } else {
                l = mid - 1;
            }
        }
        return -1;
    }

    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the size of array:");
        int n = sc.nextInt();
        char arr[] = new char[n];

        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.next().charAt(0);
        }
        Arrays.sort(arr); // Sorting the array before performing binary
search
        System.out.println("Enter the character you want to search:");
        char c = sc.next().charAt(0);


        int k = BinarySearch(arr, n, c);
        if (k != -1) {
            System.out.println("Element found at index: " + k); // This
will now print the 0-based index
        } else {
            System.out.println("Element not found.");
        }


        System.out.println("Array elements:");
        for (int i = 0; i < n; i++) {
            System.out.print(arr[i] + " ");
        }
    }
}
```

2] merge sort

```
import java.util.*;
public class MergeSort {
    public static void merge(int arr[],int f,int mid ,int l)
    {
        int n1=mid-f+1;
        int n2=l-mid;
        int left[]=new int[n1];
        int right[]=new int[n2];
        int i;
        int j;
        int k;

        for(i=0;i<n1;i++)
        {
            left[i]=arr[f+i];
        }
        for(j=0;j<n2;j++)
        {
            right[j]=arr[mid+1+j];
        }
        i=0;
        j=0;
        k=f; // Changed from '0' to 'f'
        while(i<n1 && j<n2)
        {
            if(left[i]<=right[j])
            {
                arr[k]=left[i];
                k++;
                i++;
                }
            else
            {
                arr[k]=right[j];
                k++;
                j++;
            }

        }
        while(i<n1)
        {
            arr[k]=left[i];
            k++;
            i++;
        }
        while(j<n2)
        {
            arr[k]=right[j];
            k++;
            j++;
        }



        }
```

```
public static void mergesort(int arr[], int f,int l)
{

    int mid=(f+l)/2;
    if(f>=l)
    {
        return;
    }
    mergesort(arr, f,mid);
    mergesort(arr,mid+1,l);
    merge(arr,f,mid,l);
}
public static void printarray(int arr[],int n)
{
    for(int i=0;i<n;i++)
    {
        System.out.print(arr[i]+" ");
    }
}

public static void main(String[] args) {
    // TODO Auto-generated method stub
    Scanner sc=new Scanner(System.in);
    System.out.println("Enter the size of array:");
    int n=sc.nextInt();
    System.out.println("enter the elements of array:");
    int arr[]=new int[n];
    for(int i=0;i<n;i++)
    {
        arr[i]=sc.nextInt();
    }
    printarray(arr,n);
    mergesort(arr,0,n-1);
    System.out.println();
    printarray(arr,n);



}
```

```
}
```

3] quick sort

```
import java.util.Arrays;

class Quicksort {

  // method to find the partition position
  static int partition(int array[], int low, int high) {

    // choose the rightmost element as pivot
    int pivot = array[high];

    // pointer for greater element
    int i = (low - 1);

    // traverse through all elements
    // compare each element with pivot
    for (int j = low; j < high; j++) {
      if (array[j] <= pivot) {

        // if element smaller than pivot is found
        // swap it with the greater element pointed by i
        i++;

        // swapping element at i with element at j
        int temp = array[i];
        array[i] = array[j];
        array[j] = temp;
      }
    }

    // swap the pivot element with the greater element specified by i
    int temp = array[i + 1];
    array[i + 1] = array[high];
    array[high] = temp;

    // return the position from where partition is done
    return (i + 1);
  }

  static void quickSort(int array[], int low, int high) {
    if (low < high) {

      // find pivot element such that
      // elements smaller than pivot are on the left
      // elements greater than pivot are on the right
      int pi = partition(array, low, high);

      // recursive call on the left of pivot
      quickSort(array, low, pi - 1);

      // recursive call on the right of pivot
      quickSort(array, pi + 1, high);
    }
  }
}

// Main class
class QU {
  public static void main(String args[]) {

    int[] data = { 8, 7, 2, 1, 0, 9, 6 };
    System.out.println("Unsorted Array");
    System.out.println(Arrays.toString(data));

    int size = data.length;
    // call quicksort() on array data
    Quicksort.quickSort(data, 0, size - 1);

    System.out.println("Sorted Array in Ascending Order ");
    System.out.println(Arrays.toString(data));
  }
}
```

4]heap

sort

```
import java.util.*;

public class HeapSort {

        public static void heapify(int arr[],int i,int size)

        {

                int right=2*i+1;

                int left=2*i+2;



                int maxIdx=i;



                if(left<size && arr[left]>arr[maxIdx] )

                {

                        maxIdx=left;

                }

                if(right<size && arr[right]>arr[maxIdx])

                {

                        maxIdx=right;

                }



                if(maxIdx != i)

                {

                        int temp=arr[i];

                        arr[i]=arr[maxIdx];

                        arr[maxIdx]=temp;

                        heapify(arr,maxIdx,size);

                }

        }

        public static void heapsort(int arr[],int n)
```

```
    {
        for(int i=n/2;i>=0;i--)
        {
            heapify(arr,i,n);
        }
        for(int i=n-1;i>=0;i--)
        {
            int temp=arr[0];
            arr[0]=arr[i];
            arr[i]=temp;
            heapify(arr,0,i);
        }

    }

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Scanner sc=new Scanner(System.in);
System.out.println("Enter the elements of array:");
  int n=sc.nextInt();
        System.out.println("Enter elements of array:");
        int arr[]=new int[n];
        for(int i=0;i<n;i++)
        {
            arr[i]=sc.nextInt();

        }

        for(int i=0;i<n;i++)
        {
            System.out.print(arr[i]+" ");
        }
        heapsort(arr,n);                System.out.println();
```

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

        }

}
```

5] 0/1 knapsack

```
import java.util.*;
public class Knapsack {
    public static void print(int dp[][])
    {
        for(int i=0;i<dp.length;i++)
        {
            for(int j=0;j<dp[0].length;j++)
            {
                System.out.print(dp[i][j]+" ");
            }
            System.out.println();
        }
    }
    public static int knapsa(int val[],int wt[],int w)
    {
        int n=val.length;
        int dp[][]=new int[n+1][w+1];
        for(int i=0;i<dp.length;i++)
        {
            dp[i][0]=0;
        }
        for(int j=0;j<dp[0].length;j++)
        {
            dp[0][j]=0;
        }

        for(int i=1;i<n+1;i++)
        {
            for(int j=1;j<w+1;j++)
            {
                int v=val[i-1];
                int W=wt[i-1];
                if(W<=j)
                {
                    int incProfit=v+dp[i-1][j-W];
                    int excProfit=dp[i-1][j];
                    dp[i][j]=Math.max(incProfit, excProfit);
                }
                else
                {
                    int excProfit=dp[i-1][j];
                    dp[i][j]=excProfit;
                }
            }
        }
        print(dp);

        return dp[n][w];
    }
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        int val[]= {15,14,10,45,30};
        int wt[]= {2,5,1,3,4};
        int w=7;

        System.out.println(knapsa(val,wt,w));
    }
}
```

6] coin changing

```
public class CoinChange {
        public static int coinchange(int arr[],int sum)
        {
                int n=arr.length;
                int dp[][] = new int[n+1][sum+1];
                for(int i=0;i<n+1;i++)
                {
                        dp[i][0]=1;
                }
                for(int j=1;j<sum+1;j++)
                {
                        dp[0][j]=0;
                }

                for(int i=1;i<n+1;i++)
                {
                        for(int j=1;j<sum+1;j++)
                        {
                                if(arr[i-1]<=j)
                                {
                                dp[i][j]=dp[i][j-arr[i-1]]+dp[i-1][j];
                                }
                        else
                        {
                                dp[i][j]=dp[i-1][j];
                        }
                        }
                }
                return dp[n][sum];
        }



        public static void main(String[] args) {                 // TODO
Auto-generated method stub

                int arr[]= {1,2,3};

                int sum= 4;



                int c=coinchange(arr,sum);

                System.out.println(c);



        }

 }
```

7] binomial coefficient

```
 class Binomialcoeeficient {

    static int binomialCoeff(int n, int k)
    {
        int C[][] = new int[n + 1][k + 1];
        int i, j;

        for (i = 0; i <= n; i++) {
            for (j = 0; j <= min(i, k); j++) {
                // Base Cases
                if (j == 0 || j == i)
                    C[i][j] = 1;

                // Calculate value using previously
stored values
                else
                    C[i][j] = C[i - 1][j - 1] + C[i -
1][j];
            }
        }
        return C[n][k];
    }
    static int min(int a, int b) { return (a < b) ? a
: b; }

    public static void main(String args[])
    {
        int n = 6, k = 3;
        System.out.println("Value of C(" + n + "," +
k
                                 + ") is " +
binomialCoeff(n, k));
    }
}
```

8]BFS

```
import java.util.*;
public class BFS {
    static class Edge {
        int src;
        int dest;
        public Edge(int s, int d) {
            this.src = s;
            this.dest = d;
        }
    }
    static void createGraph(ArrayList<Edge> graph[]) {
        for(int i=0; i<graph.length; i++) {
            graph[i] = new ArrayList<>();
        }
        graph[0].add(new Edge(0, 1));
        graph[0].add(new Edge(0, 2));
        graph[1].add(new Edge(1, 0));
        graph[1].add(new Edge(1, 3));
        graph[2].add(new Edge(2, 0));
        graph[2].add(new Edge(2, 4));
        graph[3].add(new Edge(3, 1));
        graph[3].add(new Edge(3, 4));
        graph[3].add(new Edge(3, 5));
        graph[4].add(new Edge(4, 2));
        graph[4].add(new Edge(4, 3));
        graph[4].add(new Edge(4, 5));
        graph[5].add(new Edge(5, 3));
        graph[5].add(new Edge(5, 4));
        graph[5].add(new Edge(5, 6));
        graph[6].add(new Edge(6, 5));
    }
    public static void bfs(ArrayList<Edge> graph[], int V) {
        boolean visited[] = new boolean[V];
        Queue<Integer> q = new LinkedList<>();
        q.add(0); //Source = 0
        while(!q.isEmpty()) {
            int curr = q.remove();
            if(!visited[curr]) {
                System.out.print(curr+" ");
                visited[curr] = true;
                for(int i=0; i<graph[curr].size(); i++) {
                    Edge e = graph[curr].get(i);
                    q.add(e.dest);
                }
            }
        }
        System.out.println();
    }
    public static void main(String args[]) {
        /*
        1 --- 3
        / | \
        0 | 5 -- 6
        \ | /
        2 ---- 4
        */
        int V = 7;
        ArrayList<Edge> graph[] = new ArrayList[V];
        createGraph(graph);
        bfs(graph, V);
    }
}
```

9]Prims algorithm’

```
import java.util.Scanner;
public class PrimsAlgorithm{
    public static void primsAlgorithm(int arr[][],int v )
    {
        int no_of_edge=0;
        int selected[]=new int[v];
        selected[0]=1;
        int x;
        int y;
        int sum=0;
        while(no_of_edge < v - 1) // Changed from 'v' to 'v - 1'
        {
            x=0;
            y=0;
            int min=Integer.MAX_VALUE;
            for(int i=0;i<v;i++)
            {
                if(selected[i]==1)
                {
                    for(int j=0;j<v;j++)
                    {
                        if(selected[j]==0 && arr[i][j] != 0)
                        {
                            if(min>arr[i][j])
                            {
                                min=arr[i][j];
                                sum=sum+min;
                                x=i;
                                y=j;
                            }
                        }
                    }
                }
            }
            System.out.println(x+"-->"+y+" ");
            System.out.println(arr[x][y]);
            selected[y]=1;
            no_of_edge++;
        }
        System.out.println(sum);
    }
    public static void main(String args[])
    {
        System.out.println("Enter number of vertices:");
        Scanner sc=new Scanner(System.in);
        int v=sc.nextInt();
        System.out.println("Enter the element of graphs :");
        int arr[][]=new int[v][v];
        for(int i=0;i<v;i++)
        {
            for(int j=0;j<v;j++)
            {
                arr[i][j]=sc.nextInt();
            }
        }
        primsAlgorithm(arr,v );
    }
}
```

Input and output

```
Enter number of vertices:
5
Enter the element of graphs :
0 2 0 6  0
2 0 3 8 5
0 3 0 0 7
6 8 0 0 9
0 5 7 9 0
0-->1
2
1-->2
3
1-->4
5
0-->3
6
28
```