Linear Search

```
public class LP1_LinearSearch {
    public static void main(String args[]) {
        Scanner sc=new Scanner(System.in);
        int arr[]=new int [10];
        int i,n,key;
        boolean found=false;
        System.out.print("Enter Number of Elements: ");
        n=sc.nextInt();
        System.out.println("Enter the Elements:");
        for(i=0;i<n;i++)
            arr[i]=sc.nextInt();
        System.out.println();
        System.out.print("Enter the search Element: ");
        key=sc.nextInt();
        for(i=0;i<n;i++) {
            if(key==arr[i]) {
                System.out.println(key+" found at position "+(i+1));
                found=true;
            }
        }
        if(!found) {
            System.out.println(key+" not found!");
        }
    }
}
```

Binary Search

```
public class BS_Time {
    public static int search(int arr[], int key) {
        int low = 0;
        int high = arr.length - 1;

        while (low <= high) {
            int mid = low + (high - low) / 2;
            if (arr[mid] == key) {
                return mid;
            } else if (arr[mid] < key) {
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }
        return -1;
    }
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.println("Enter size:");
        int n = in.nextInt();
        int a[] = new int[n];
        System.out.println("Enter the elements in sorted manner:");
        for (int i = 0; i < n; i++)
            a[i] = in.nextInt();
        int key;
        System.out.println("Enter the search element:");
        key = in.nextInt();

        long start = System.nanoTime();
        int index = search(a, key);
        long stop = System.nanoTime();
        if (index != -1)
            System.out.println("Element found at index " + index);
        else
            System.out.println("Element not found");
        System.out.println("Time taken: " + (stop - start) + " nanoseconds");
    }
}
```

Fiboncacci using Recursion

```
public class LP4_Fib {
    static int fib(int x) {
        if (x == 1) return 15;
        if (x == 2) return 23;
        else return fib(x - 1) + fib(x - 2);
    }
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        System.out.println("The next 3 terms of the series 15,23,38,61 is: ");
        for (int i = 1; i <= 7; i++)
            System.out.print(fib(i) + " ");
    }
}
```

Quick Sort with Time

```
public class QuickSort {
    void quick(int arr[], int i, int j) {
        if (i < j) {
            int s = partition(arr, i, j);
            quick(arr, i, s - 1);
            quick(arr, s + 1, j);
        }
    }
    int partition(int arr[], int l, int h) {
        int p, i, j, temp;
        p = arr[l];
        i = l + 1;
        j = h;
        while (l < h) {
            while (arr[i] < p && i < h)
                i++;
            while (arr[j] > p)
                j--;
            if (i < j) {
                temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            } else {
                temp = arr[l];
                arr[l] = arr[j];
                arr[j] = temp;
                return j;
            }
        }
        return j;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter number of ele: ");
        int n = sc.nextInt();
        Random gen = new Random();
        int arr[] = new int[5000];
        for (int i = 0; i < n; i++)
            arr[i] = gen.nextInt(1000);
        System.out.println("Random ele: ");
        for (int i = 0; i < n; i++)
            System.out.print(arr[i] + " ");
        System.out.println();
        long start = System.nanoTime();
        QuickSort qs = new QuickSort();
        qs.quick(arr, 0, n - 1);
        long stop = System.nanoTime();
        System.out.println("array after sorting: ");
        for (int i = 0; i < n; i++)
            System.out.print(arr[i] + " ");
        System.out.println();
        System.out.println("Time taken: " + (stop - start));
    } }
```

Recursive NCR

```
public class NCR{
      static int factorial(int n){
            if(n == 0 || n ==1)
                  return 1;
            else
                  return n*factorial(n-1);
      }
      public static int nCr(int n,int r){
            if(r == 0|| r == n)
                  return 1;
            else
                  return factorial(n) / (factorial(n-r) * factorial(r));
      }
      public static void main(String[] args){
            Scanner in = new Scanner(System.in);
            System.out.print("Enter number of items to choose from: ");
            int total = in.nextInt();
            System.out.print("Enter number of items to be chosen: ");
            int choose = in.nextInt();
            long combinations = nCr(total, choose);
            System.out.println("No of ways: " + combinations);
      }
}
```

Selection Sort

```
public class SelectionSort {
      void sort(int arr[]){
            int n = arr.length;
            for(int i = 0; i<n -1; i++){
                  int min_id = i;
                  for(int j = i+1; j< n; j++)
                        if(arr[j] < arr[min_id])
                              min_id = j;
                  int temp = arr[min_id];
                  arr[min_id] = arr[i];
                  arr[i] = temp;
            }
      }
      public static void main(String[] args){
            Scanner in = new Scanner(System.in);
            SelectionSort ss = new SelectionSort();
            System.out.print("Enter the number of elements in the array: ");
            int n = in.nextInt();
            int arr[] = new int[n];
            System.out.print("Enter the elements of the array:");
            for(int i = 0;i<n ; i++)
                  arr[i] = in.nextInt();
            System.out.print("Array before sorting:");
            for(int i = 0; i< n; i++)
                  System.out.print(arr[i] + " ");
            System.out.println();
            ss.sort(arr);
            System.out.print("Array after sorting:");
            for(int i = 0; i<n;++i)
                  System.out.print(arr[i] + " ");
            System.out.println();
      }
}
```

Prims

```
public class Prims {
    final static int MAX = 20;
    static int n;
    static int cost[][];
    static Scanner scan = new Scanner(System.in);
    public static void main(String[] args) {
        ReadMatrix();
        Prims();
    }
    static void ReadMatrix() {
        int i, j;
        cost = new int[MAX][MAX];
        System.out.println("Enter the number of nodes:");
        n = scan.nextInt();
        System.out.println("Enter the adjacency matrix:");
        for (i = 1; i <= n; i++) {
            for (j = 1; j <= n; j++) {
                cost[i][j] = scan.nextInt();
                if (cost[i][j] == 0)
                    cost[i][j] = 999;
            }
        }
    }
    static void Prims() {
        int visited[] = new int[10];
        int ne = 1, i, j, min, a = 0, b = 0, u = 0, v = 0;
        int mincost = 0;
        visited[1] = 1;
        while (ne <= n) {
            for (i = 1, min = 999; i <= n; i++) {
                for (j = 1; j <= n; j++) {
                    if (cost[i][j] < min) {
                        if (visited[i] != 0) {
                            min = cost[i][j];
                            a = u = i;
                            b = v = j;
                        }
                    }
                }
            }
            if (visited[u] == 0 || visited[v] == 0) {
                System.out.println("Edge"+ne+":(" + a + "," + b + ")" + " cost :" + min);
                mincost += min;
                visited[b] = 1;
            }
            cost[a][b] = cost[b][a] = 999;
            ne++;
        }
        System.out.println("Minimun cost: " + mincost);
    }
}
```

Kruskals
```
public class Kruskals {
    final static int MAX = 20;
    static int n;
    static int cost[][];
    static Scanner scan = new Scanner(System.in);
    public static void main(String[] args) {
        ReadMatrix();
        Kruskals();
    }
    static void ReadMatrix() {
        int i, j;
        cost = new int[MAX][MAX];
        System.out.println("Implementation of Kruskal's algorithm");
        System.out.println("Enter the no. of vertices");
        n = scan.nextInt();
        System.out.println("Enter the cost adjacency matrix");
        for (i = 1; i <= n; i++) {
            for (j = 1; j <= n; j++) {
                cost[i][j] = scan.nextInt();
                if (cost[i][j] == 0) {
                    cost[i][j] = 999;
                }
            }
        }
    }
    static void Kruskals() {
        int a = 0, b = 0, u = 0, v = 0, i, j, ne = 1, min, mincost = 0;
        int parent[] = new int[9];
        for (i = 1; i <= n; i++)
            parent[i] = 0;
        System.out.println("The edges of Minimum Cost Spanning Tree are");
        while (ne < n) {
            min = 999;
            for (i = 1; i <= n; i++) {
                for (j = 1; j <= n; j++) {
                    if (cost[i][j] < min) {
                        min = cost[i][j];
                        a = u = i;
                        b = v = j;
                    }
                }
            }
            while (parent[u] != 0)
                u = parent[u];
            while (parent[v] != 0)
                v = parent[v];
            if (u != v) {
                System.out.println(ne++ + "edge (" + a + "," + b + ") =" + min);
                mincost += min;
                parent[v] = u;
            }
            cost[a][b] = cost[b][a] = 999;
        }
        System.out.println("Minimum cost :" + mincost);
    }
}
```

Floyds Algorithm

```
import java.util.*;
class Floyds{
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter the number of vertices: ");
    int n =sc.nextInt();
    System.out.println("Enter the adj matrix:(enter 999 for infinity) ");
    int adj[][] = new int[10][10];
    for(int i=1;i<=n;i++)
      for(int j=1;j<=n;j++)
        adj[i][j] = sc.nextInt();
    flyod(adj,n);
    System.out.println("the all pair shoretst path is: ");
    for(int i=1;i<=n;i++){
      for(int j=1;j<=n;j++){
        System.out.print(adj[i][j]+" ");
      }
      System.out.println();
    }
  }
  static void flyod(int arr[][],int n){
    for(int k=1;k<=n;k++)
      for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++)
          arr[i][j] = min(arr[i][j],(arr[i][k]+arr[k][j]));
  }
  static int min(int a,int b){
    if(a<b)
      return a;
    return b;
  }
```

Dijkstra's Algorithm

```
public class LP12_Dijkstra {
      public static void main(String[] args) {
            int i, j;
            int dist[]=new int[10], visited[]=new int[10];
            int cost[][]=new int[10][10], path[]=new int[10];
            Scanner in = new Scanner(System.in);
            System.out.print("No of nodes: ");
            int n = in.nextInt();
            System.out.println("Cost matrix:");
            for(i=1;i<=n;i++)
                  for(j=1;j<=n;j++)
                        cost[i][j] = in.nextInt();
            System.out.println("Cost matrix is");
            for(i=1;i<=n;i++){
                  for(j=1;j<=n;j++)
                        System.out.print(cost[i][j]+"\t");
                  System.out.println();
            }
            System.out.print("Source vertex: ");
            int sv = in.nextInt();
            dij(cost,dist,sv,n,path,visited);
            printpath(sv,n,dist,path,visited );
      }
      static void dij(int cost[][],int dist[],int src,int n, int path[],int visited[]){
            int count=2,min,v=0;
            for(int i=1;i<=n;i++) {
                  visited[i]=0;
                  dist[i]=cost[src][i];
                  if(cost[src][i]==999)
                        path[i]=0;
                  else
                        path[i]=src;
            }
            visited[src]=1;
            while(count<=n) {
                  min=999;
                  for(int w=1;w<=n;w++)
                        if(dist[w]<min && visited[w]==0) {
                              min=dist[w];
                              v=w;
                        }
                  visited[v]=1;
                  count++;
                  for(int w=1;w<=n;w++)
                        if(dist[w]> (dist[v]+cost[v][w])) {
                              dist[w]=dist[v]+cost[v][w];
                              path[w]=v;
                        }
            }
      }
      static void printpath(int src,int n,int dist[], int path[],int visited[]) {
            for(int w=1;w<=n;w++) {
                  if(visited[w]==1 && w!=src) {
                        System.out.println("Short dist:"+src+"-->"+w+" is: "+dist[w]);
                        System.out.print("Path is: "+w+"-->");
                        int t=path[w];
                        while(t!=src) {
                              System.out.print(t+"-->");
                              t=path[t];
                        }
                        System.out.print(src+"\n");
                  }
            }
      }
}
```

Bellman Ford Algorithm

```
class Graph {
  static class Edge {
    int src, dest, weight;
    Edge(int s, int d, int w) {
      src = s;
      dest = d;
      weight = w;
    }
  };
  int V, E;
  Edge edge[];
  Graph(int v, int e) {
    V = v;
    E = e;
    edge = new Edge[e];
  }
  void BellmanFord(Graph graph, int src) {
    int V = graph.V, E = graph.E;
    int dist[] = new int[V];
    for (int i = 0; i < V; ++i)
      dist[i] = Integer.MAX_VALUE;
    dist[src] = 0;
    for (int i = 1; i < V; ++i) {
      for (int j = 0; j < E; ++j) {
        int u = graph.edge[j].src;
        int v = graph.edge[j].dest;
        int weight = graph.edge[j].weight;
        if (dist[u] != Integer.MAX_VALUE && dist[u] + weight < dist[v])
          dist[v] = dist[u] + weight;
      }
    }
    for (int j = 0; j < E; ++j) {
      int u = graph.edge[j].src;
      int v = graph.edge[j].dest;
      int weight = graph.edge[j].weight;
      if (dist[u] != Integer.MAX_VALUE && dist[u] + weight < dist[v]) {
        System.out.println("Graph contains negative weight cycle");
        return;
      }
    }
    printArr(dist, V);
  }
  void printArr(int dist[], int V) {
    System.out.println("Vertex Distance from Source");
    for (int i = 0; i < V; ++i)
      System.out.println(i + "\t" + dist[i]);
  }
  public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    System.out.print("Enter no. of vertices: ");
    int V = in.nextInt();
    System.out.print("Enter no. of edges: ");
    int E = in.nextInt();
    Graph graph = new Graph(V, E);
    for (int i = 0; i < E; i++) {
      System.out.print("Enter src, dest and weight for edge " + (i + 1) + " : ");
      int src = in.nextInt();
      int dest = in.nextInt();
      int weight = in.nextInt();
      graph.edge[i] = new Edge(src, dest, weight);
    }
    graph.BellmanFord(graph, 0);
  }
}
```

Travelling Salesman Problem

```
public class TSPDynamicProgramming {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the number of cities: ");
        int n = scanner.nextInt();
        System.out.println("Enter the distance between cities: ");
        int[][] distance = new int[n][n];
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                distance[i][j] = scanner.nextInt();
        int minCost = tsp(distance, n);
        System.out.println("Minimum cost to visit all cities: " + minCost);
    }
    public static int tsp(int[][] distance, int n) {
        int[][] dp = new int[1 << n][n];
        final int INF = 1_000_000_000;
        for (int[] row : dp)
            Arrays.fill(row, INF);
        dp[1][0] = 0;
        for (int mask = 1; mask < (1 << n); mask++)
            for (int i = 0; i < n; i++)
                if ((mask & (1 << i)) != 0)
                    for (int j = 0; j < n; j++)
                        if (i != j && (mask & (1 << j)) != 0)
                            dp[mask][i] = Math.min(dp[mask][i], dp[mask^(1 << i)][j]+
distance[j][i]);
        int minCost = INF;
        for (int i = 1; i < n; i++)
            if (dp[(1 << n) - 1][i] + distance[i][0] < minCost)
                minCost = dp[(1 << n) - 1][i] + distance[i][0];
        return minCost;
    }
}
```

Sum of Subset

```
public class LP17_SumOfSubSubset {
    static int count = 0;
    static void subset(int cs, int k, int r, int[] x, int[] w, int d) {
        x[k] = 1;
        if (cs + w[k] == d) {
            count++;
            System.out.print("Solution " + count + ": {");
            for (int i = 0; i < w.length; i++)
                if (x[i] == 1) System.out.print(w[i] + (i < w.length - 1 ? ", " : ""));
            System.out.println("}");
        } else if (cs + w[k] + w[k + 1] <= d) subset(cs + w[k], k + 1, r - w[k], x,w,d);
        if (cs + r - w[k] >= d && cs + w[k + 1] <= d) {
            x[k] = 0;
            subset(cs, k + 1, r - w[k], x, w, d);
        }
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("No of elements in the set: ");
        int n = sc.nextInt(); int[] w = new int[n]; int[] x = new int[n]; int sum = 0;
        System.out.print("Enter the elements: ");
        for (int i = 0; i < n; i++) {
            w[i] = sc.nextInt();
            sum += w[i];
        }
        System.out.print("Enter the desired sum: ");
        int d = sc.nextInt();
        System.out.println("Sum is: " + sum);
        subset(0, 0, sum, x, w, d);
    }
}
```

0/1 Knapsack using Branch and Bound

```
public class KnapsackBranchBound {
    static int calculateUpperBound(Item[] items, int capacity, int currentWeight, int
currentValue, int currentIndex) {
        int remainingCapacity = capacity - currentWeight;
        int upperBound = currentValue;
        while (currentIndex < items.length && items[currentIndex].weight <=
remainingCapacity) {
            remainingCapacity -= items[currentIndex].weight;
            upperBound += items[currentIndex].value;
            currentIndex++;
        }
        if (currentIndex < items.length) {
            upperBound += (remainingCapacity * items[currentIndex].value) /
items[currentIndex].weight;
        }
        return upperBound;
    }
    static void knapsack(Item[] items, int capacity, int currentWeight, int currentValue,
int currentIndex, int[] maxProfit) {
        if (currentWeight > capacity || currentIndex == items.length) {
            if (currentValue > maxProfit[0]) maxProfit[0] = currentValue;
            return;
        }
        int upperBound = calculateUpperBound(items, capacity, currentWeight, currentValue,
currentIndex);
        if (upperBound < maxProfit[0]) return;
        knapsack(items, capacity, currentWeight + items[currentIndex].weight, currentValue
+ items[currentIndex].value, currentIndex + 1, maxProfit);
        knapsack(items, capacity, currentWeight, currentValue, currentIndex + 1,
maxProfit);
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("No of items: ");
        int numItems = scanner.nextInt();
        int[] weights = new int[numItems];
        int[] values = new int[numItems];
        System.out.println("Weights of items:");
        for (int i = 0; i < numItems; i++) weights[i] = scanner.nextInt();
        System.out.println("Values of items:");
        for (int i = 0; i < numItems; i++) values[i] = scanner.nextInt();
        System.out.print("Capacity of knapsack: ");
        int capacity = scanner.nextInt();
        Item[] items = new Item[numItems];
        for (int i = 0; i < numItems; i++) items[i] = new Item(weights[i], values[i]);
        int[] maxProfit = {0};
        knapsack(items, capacity, 0, 0, 0, maxProfit);
        System.out.println("Maximum value: " + maxProfit[0]);
    }
}
class Item {
    int weight, value;
    public Item(int weight, int value) {
        this.weight = weight;
        this.value = value;
    }
}
```

NQueens
```
public class NQueens {
    private final int[] result;
    private final boolean[] column;
    private final boolean[] leftDiagonal;
    private final boolean[] rightDiagonal;
    private final int n;
    public NQueens(int n) {
        this.n = n;
        result = new int[n];
        column = new boolean[n];
        leftDiagonal = new boolean[2 * n - 1];
        rightDiagonal = new boolean[2 * n - 1];
    }
    public boolean solve(){
        return solveNQueens(0);
    }
    private boolean solveNQueens(int row) {
        if (row == n) {
            printSolution();
            return true;
        }
        boolean res = false;
        for (int col = 0; col < n; col++)
            if (isSafe(row, col)) {
                placeQueen(row, col);
                res = solveNQueens(row + 1) || res;
                removeQueen(row, col); // Backtrack
            }
        return res;
    }
    private boolean isSafe(int row, int col) {
        return !column[col] && !leftDiagonal[row-col+n-1] && !rightDiagonal[row+col];
    }
    private void placeQueen(int row, int col) {
        result[row] = col;
        column[col] = true;
        leftDiagonal[row - col + n - 1] = true;
        rightDiagonal[row + col] = true;
    }
    private void removeQueen(int row, int col) {
        column[col] = false;
        leftDiagonal[row - col + n - 1] = false;
        rightDiagonal[row + col] = false;
    }
    private void printSolution() {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++)
                if (result[i] == j)
                    System.out.print("Q ");
                else
                    System.out.print(". ");
            System.out.println();
        }
        System.out.println();
    }
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.print("Enter n size: ");
        int n = in.nextInt();
        NQueens queens = new NQueens(n);
        if (!queens.solve())
            System.out.println("No solution exists");
    }
}
```