1. BFS(bridth first search):

```java
import java.util.*;

class BFS {
    void bfs(int start, List<List<Integer>> graph) {
        boolean[] visited = new boolean[graph.size()];
        Queue<Integer> queue = new LinkedList<>();

        visited[start] = true;
        queue.add(start);

        while (!queue.isEmpty()) {
            int node = queue.poll();
            System.out.print(node + " ");

            for (int neighbor : graph.get(node)) {
                if (!visited[neighbor]) {
                    visited[neighbor] = true;
                    queue.add(neighbor);
                }
            }
        }
    }
}

```

2. linear search

```
public class BruteForceSearch {
   static int linearSearch(int[] arr, int key) {
      int i =0;
      while ( i < arr.length) {
         if (arr[i] == key) {
            return i;
         }
      }
      return -1;
   }
   public static void main(String[] args) {
```

```
        int[] arr = {10, 25, 30, 45, 50};
        int key = 30;
        int result = linearSearch(arr, key);
        System.out.println(result != -1
                ? "Found at index " + result
                : "Not found");
    }
}
```

3. DFS

```
import java.util.List;
class DFS {
    void dfs(int node, boolean[] visited, List<List<Integer>> graph) {
        visited[node] = true;
        System.out.print(node + " ");

        for (int neighbor : graph.get(node)) {
            if (!visited[neighbor]) {
                dfs(neighbor, visited, graph);
            }
        }
    }
}
```

4. FibonacciRecursive:

```
public class FibonacciRecursive {
```

```
  static int fibonacci(int n) {
    if (n == 0)
      return 0;
    if (n == 1)
      return 1;
    return fibonacci(n - 1) + fibonacci(n - 2);
  }
  public static void main(String[] args) {
    int n = 6;
    System.out.println("Fibonacci of " + n + " is: " + fibonacci(n));
  }
}
```

5.floyed algorithm:

```
class Floyd {

  static final int INF = 99999;

  static void floydWarshall(int[][] graph) {
    int V = graph.length;
    int[][] dist = graph.clone();

    for (int k = 0; k < V; k++)
      for (int i = 0; i < V; i++)
        for (int j = 0; j < V; j++)
          if (dist[i][k] + dist[k][j] < dist[i][j])
            dist[i][j] = dist[i][k] + dist[k][j];
```

```
    for (int i = 0; i < V; i++) {
      for (int j = 0; j < V; j++)
        System.out.print(dist[i][j] + " ");
      System.out.println();
    }
  }
}
```

6. Heap sort:

```
class HeapSort {

  static void heapify(int[] arr, int n, int i) {
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    if (left < n && arr[left] > arr[largest])
      largest = left;

    if (right < n && arr[right] > arr[largest])
      largest = right;

    if (largest != i) {
      int temp = arr[i];
      arr[i] = arr[largest];
      arr[largest] = temp;
```

```
            heapify(arr, n, largest);
        }
    }

    static void heapSort(int[] arr) {
        int n = arr.length;

        // Build max heap
        for (int i = n / 2 - 1; i >= 0; i--)
            heapify(arr, n, i);

        // Extract elements
        for (int i = n - 1; i > 0; i--) {
            int temp = arr[0];
            arr[0] = arr[i];
            arr[i] = temp;

            heapify(arr, i, 0);
        }
    }
}
```

7. Huffman

```
import java.util.*;

public class Huffman {
```

```
static class Node {
   char ch;
   int freq;
   Node left;
   Node right;
}


static class Compare implements Comparator<Node> {
   @Override
   public int compare(Node a, Node b) {
      return a.freq - b.freq;
   }
}


static Node buildTree(char[] chars, int[] freq) {

   PriorityQueue<Node> pq = new PriorityQueue<>(new Compare());


   for (int i = 0; i < chars.length; i++) {
      Node node = new Node();
      node.ch = chars[i];
      node.freq = freq[i];
      node.left = null;
      node.right = null;
```

```
        pq.add(node);
    }


    while (pq.size() > 1) {
        Node left = pq.poll();
        Node right = pq.poll();

        Node parent = new Node();
        parent.ch = '-';
        parent.freq = left.freq + right.freq;
        parent.left = left;
        parent.right = right;

        pq.add(parent);
    }

    return pq.poll();
}


static void printCode(Node root, String code) {


    if (root.left == null && root.right == null) {
        System.out.println(root.ch + ": " + code);
        return;
    }
```

```
        printCode(root.left, code + "0");
        printCode(root.right, code + "1");
    }

    public static void main(String[] args) {

        char[] chars = {'a', 'b', 'c', 'd', 'e', 'f'};
        int[] freq   = {5, 9, 12, 13, 16, 45};

        Node root = buildTree(chars, freq);

        System.out.println("Huffman Codes:");
        printCode(root, "");
    }
}
```

8.Insertion sort:

```
public class InsertionSort {


    public static void insertionSort(int[] arr) {


        int n = arr.length;
```

```
    for (int i = 1; i < n; i++) {

        int key = arr[i];

        int j = i - 1;

        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }

        arr[j + 1] = key;
    }
}

public static void main(String[] args) {

    int[] arr = {9, 5, 1, 4, 3};

    insertionSort(arr);
```

```
    System.out.println("Sorted array:");
    for (int num : arr) {
      System.out.print(num + " ");
    }
  }
}
```

9.Kruskal:

```
class Kruskal {

  static class Edge implements Comparable<Edge> {
    int src, dest, weight;

    public int compareTo(Edge compareEdge) {
      return this.weight - compareEdge.weight;
    }
  }

  static class Subset {
    int parent, rank;
  }

  static int find(Subset[] subsets, int i) {
  while (subsets[i].parent != i)
    i = subsets[i].parent;
```

```
    return i;
    }

    static void union(Subset[] subsets, int x, int y) {
        int xroot = find(subsets, x);
        int yroot = find(subsets, y);

        if (subsets[xroot].rank < subsets[yroot].rank)
            subsets[xroot].parent = yroot;
        else if (subsets[xroot].rank > subsets[yroot].rank)
            subsets[yroot].parent = xroot;
        else {
            subsets[yroot].parent = xroot;
            subsets[xroot].rank++;
        }
    }
}
```

10. prims:

```
class Prims {

    static int minKey(int[] key, boolean[] mstSet, int V) {
        int min = Integer.MAX_VALUE, minIndex = -1;

        for (int v = 0; v < V; v++)
            if (!mstSet[v] && key[v] < min) {
```

```
            min = key[v];
            minIndex = v;
        }

    return minIndex;
}

static void primMST(int[][] graph) {
    int V = graph.length;
    int[] parent = new int[V];
    int[] key = new int[V];
    boolean[] mstSet = new boolean[V];

    for (int i = 0; i < V; i++)
        key[i] = Integer.MIN_VALUE;

    key[0] = 0;
    parent[0] = -1;

    for (int count = 0; count < V - 1; count++) {
        int u = minKey(key, mstSet, V);
        mstSet[u] = true;

        for (int v = 0; v < V; v++)
            if (graph[u][v] != 0 && !mstSet[v] && graph[u][v] < key[v]) {
                parent[v] = u;
                key[v] = graph[u][v];
            }
```

```
    }


    for (int i = 1; i < V; i++)
      System.out.println(parent[i] + " - " + i);
  }
}
```

11. quick sort:

```
class QuickSortFirstPivot {

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

    int pivot = arr[low];
    int i = low + 1;
    int j = high;

    while (i <= j) {

      while (i <= high && arr[i] <= pivot)
        i++;

      while (arr[j] > pivot)
        j--;

      if (i < j) {
        int temp = arr[i];
```

```
            arr[i] = arr[j];
            arr[j] = temp;
        }
      }



      int temp = arr[low];
      arr[low] = arr[j];
      arr[j] = temp;

      return j;
    }

    static void quickSort(int[] arr, int low, int high) {
      if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
      }
    }
}
```

12. selection sort:

```
public class SelectionSort {



    public static void selectionSort(int[] arr) {
```

```
    int n = arr.length;

    for (int i = 0; i < n - 1; i++) {

        int minIndex = i;

        for (int j = i + 1; j < n; j++) {

            if (arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }

        if (minIndex != i) {
            int temp = arr[minIndex];
            arr[minIndex] = arr[i];
            arr[i] = temp;
        }
    }
}
```

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

        int[] arr = {64, 25, 12, 22, 11};

        selectionSort(arr);

        System.out.println("Sorted array:");
        for (int num : arr) {
            System.out.print(num + " ");
        }
    }
}
```

13.Tower of Hanoi:

```
public class TowerOfHanoi {

    static void hanoi(int n, char start, char temp, char end) {

        if (n == 1) {
            System.out.println("Move disk 1 from " + start + " to " + end);
            return;
        }
```

```
        hanoi(n - 1, start, end, temp);



        System.out.println("Move disk " + n + " from " + start + " to " + end);



        hanoi(n - 1, temp, start, end);
    }

    public static void main(String[] args) {
        int n = 3;



        hanoi(n, 'A', 'B', 'C');
    }
}
```