✱ MAX-HEAP

```
Public class MaxHeap {
  Private int [] Heap;
  Private int size;
  Private int maxsize;
  Public MaxHeap ( int maxsize) {
   this. maxsize = maxsize;
   this. size = 0;
   Heap = new int [this. maxsize];
  Private int parent ( int pos) {
      return (pos-1)/2; }
  Private int rightchild (int pos)
   {
   return (2 * pos) + 2;
  }
  Private boolean isLeaf (int pos) {
      if ( pos > (size/2) && pos <= size) {
    return true;
   }
   return false;
}
  private void Swap (int fpos, int spos) {
      int tmp;
      tmp = Heap [fpos];
      Heap [fpos] = Heap [spos];
       Heap [spos] = tmp;
    }
  Private Void maxHeapify (int pos) {
       if (isLeaf (pos))
            return;
       if (Heap [pos] < Heap [leftChild (pos)]
```

```
)) Heap [pos] < Heap [rightChild (pos)]) {
    if (Heap [leftchild (pos)] > Heap [rightchild (pos)]) {
     if  swap (pos, leftchild (pos));
          maxHeapify (left child (pos));
      }
    else {
      swap (pos, rightchild (pos));
      maxHeapify (rightchild (pos));
     }
   }
 }
Public void insert (int element) {
    Heap [size] = element;
    int current = size;
  while (Heap [current] > Heap [Parent (current)]) {
   swap (current, parent (current));
    current = parent (current);
     }
    size++;
 }
Public void print ()
   {
   for (int i=0; i< size/2 ; i++) {
    System.out.println ("Parent Node : " + Heap [i]);
   if (leftchild (i) < size)
System.out.print ("left child Node: " + Heap [leftchild (i)]);
if (rightchild (i) < size)
System.out.print ("Right child Node: " + Heap [rightchild (i)]);
System.out.println ();
}
```

✱ MAX-HEAP

```
Public class MaxHeap {
  Private int [] Heap;
  Private int size;
  Private int maxsize;
 Public MaxHeap ( int maxsize) {
  this. maxsize = maxsize;
  this. size = 0;
  Heap = new int [this. maxsize];
 private int parent ( int pos) {
    return (pos-1)/2; }
 Private int rightchild (int pos)
  {
 return (2 * pos) + 2;
}
 Private boolean isLeaf (int pos) {
    if ( pos > (size/2) && pos <= size) {
   return true;
 }
 return false;
}
 private void swap (int fpos, int spos) {
    int tmp;
    tmp = Head [fpos];
    Head [fpos] = Head [spos];
     Heap [spos] = tmp;
  }
 Private Void maxHeapify (int pos) {
      if (isLeaf (pos))
          return;
   if (Heap [pos] < Heap [leftChild (pos)]
```

```
|| Heap [pos] < Heap [rightChild (pos)]) {
    if (Heap [leftChild (pos)] > Heap [rightchild (pos)]) {
     if  swap (pos, leftchild (pos));
          maxHeapify (left Child (pos));
      }
    else {
      swap (pos, rightChild (pos));
      maxHeapify (rightchild (pos));
     }
   }
 }
Public void insert (int element) {
   Heap [size] = element;
   int current = size;
  while (Heap [current] > Heap [Parent (current)]) {
   swap (current, Parent (current));
   Current = parent (current);
     }
    size++;
 }
Public void print ()
   {
   for (int i=0; i< size/2 ; i++) {
    System.out.println ("Parent Node : "+ Heap [i]);
   if [leftchild (i) < size)
System.out.print ("left child Node: "+ Heap [leftchild (i)]);
if(rightchild (i) < size)
System.out.print ("Right child Node: "+ Heap [rightchild (i)]);
System.out.println ();
}
}
```

```
Public static void main (String [] args) {
    System.out.println ("The Max Heap is:");
    MaxHeap maxHeap = new MaxHeap (15);
    maxHeap.insert (5);
    max Heap: insert (3);
    max Heap. insert (17);
    max Heap. insert (10);
    max Heap. insert (84);
    max Heap. insert (19);
    max Heap. insert (6);
    maxHeap. insert (22);
    maxHeap. insert (9);
    maxHeap. print ();
System. out. println ("The Max Val is " + maxHeap. extractMax ());
    }
}
```

Ans-2)

```
package oops 1;
Public class HeapSort {
Public void Sort (int arr [])
    int n = arr.length;
for (int i = n/2 - 1 ; i >= 0 ; i--)
    heapify (arr, n, i);
for (int i = n-1; i >= 0; i--) {
    int temp = arr [0];
    arr [0] = arr [i];
    arr [i] = temp;
    heapify (arr, i, 0);
}
}
Void heapify (int arr [], int n, int i) {
    int largest = i;
    int l = 2 * i + 1;
    int r = 2 * i + 2;
    if (l < n && arr [l] > arr [largest]) {
        largest = l;
    if (r < n && arr [r] > arr [largest])
        largest = r;
    if (largest != i) {
        int swap = arr [i];
        arr [i] = arr [largest];
        arr [largest] = swap;
        heapify (arr, n, largest);
    }
}
```

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

Public static void main (String [] args) {
    int arr [] = { 12, 11, 13, 5, 6, 7 }
    int n = arr.length;
   HeapSort ob = new HeapSort ( );
   ob.sort (arr);
System.out.println ("Sorted array is");
PrintArray (arr);
  }

}
```

* Bubble Sort :-

```
import Java.util.*;
Public class BubbleSort {
   Void bubbleSort (int arr[]) {
      int n = arr.length;
      for (int i=0; i<n-1; i++)
         for(int j=0; j<n-i-1; j++)
            if (arr[j]> arr[j+1]) {
               int temp = arr[j];
               arr[j] = arr[j+1];
               arr[j+1] = temp;
            }
      }
   Void PrintArray (int arr[]) {
      int n= arr.length;
      for (int i=0; i<n; i++) {
      System.out.println (arr[i]+" ");
      System.out.println();
   }
   Public static void main (String[] args) {
      Bubble Sort ob = new BubbleSort();
      int arr[] = {5,1,4,2,8}
      ob.bubbleSort (arr);
      System.out.println ("Sorted Array");
      ob.printArray (arr);
   }
}
```

*) Insertion Sort

```
Public class InsertionSort {
    void Sort (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 = j-1;
        }
        arr[j+1]=key;
    }
}
Static void printArray [int arr[]) {
    int n = arr.length;
    for (int i=0; i<n; ++i) {
        System.out.println (arr[i]+" ");
        System.out.println ();
}
Public static void main (String [] args) {
    int arr [] = {12, 11, 13, 5, 6};
    InsertionSort ob = new InsertionSort ();
    ob.sort (arr);
    PrintArray (arr);
}
}
```

```
Public class SelectionSort {
    Void sort (int arr[]) {
      int n = arr.length;
      for (int i=0; i<n-1; i++) {
      int min_idx = 1;
      for (int j = i+1; j<n; j++) {
       if (arr[j] < arr [min_idx])
            min_idx = j;
       int temp = arr [min_idx];
       arr [min_idx] = arr [i];
       arr [i] = temp;
       }
    }
    Void printArray (int arr[]) {
       int n = arr.length;
       for (int i=0; i<n; ++i).
        System.out.println (arr[i]+" ");
        System.out.println ();
    }
    Public static void main (String [] args) {
      Selection Sort ob = new SelectionSort ();
      int arr [] = {64, 25, 12, 22, 11};
       ob. Sort (arr);
      System.out. println ("Sorted array");
      ob. printArray (arr);
```

QuickSort

```
import java.io.*;
class QuickSort1 {
static void swap(int[] arr, int i, int j) {
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
}
static int partition(int[] arr, int low, int high) {
    int pivot = arr[high];
    int i = (low-1);
  for (int j = low; j <= high-1; j++) {
      if (arr[j] < pivot) {
        i = i+1;
      swap(arr, i, j);
    }
  }
    swap(arr, i+1, high);
    return (i+1);
}
static void quickSort(int[] arr, int low, int high)
  {
    if (low < high) {
      int pi = partition(arr, low, high);
    }
}
static void printArray(int[] arr, int size) {
      for (int i=0; i< size; i++)
        System.out.println(arr[i] + " ");
        System.out.println();
    }
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);
```

MergeSort

```
Public class MergeSort {
  Void merge (int arr[], int l, int m, int r) {
    int n1 = m-l+1;
    int n2 = r-m;
    int L[] = new int [n1];
    int R[] = new int [n2];
   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];
    int i=0, j=0;
   int k=1;
 while ( i<n1 && j<n2){
    if (L[i] <= R[j]){
       arr [k] = L [i];
        i = i+1;
    }
    else {
       arr [k] = a [j];
       j= j+1;
    }
    k= k+1;
  }
 while ( i<n1 ) {
    arr[k]= L[i];
    i++;
    k++;
 }
 while (j<n2) {
   arr [k] = R[j];
    j++;
   k++;
 }
```

```
void sort (int arr[], int l, int r) {
    if (l<r){
        int m = l+(r-1)/2;
        Sort (arr, i, m);
        Sort (arr, m+1, r);
        merge (arr, l, m, r);
    }
}
Static void printArray (int arr[]){
  int n = arr.length;
  for (int i=0; i<n; ++i) {
     System.out.println (arr[i]+" ");
     System.out.println ();
  }
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 ("\n Sorted array");
  printArray (arr);
}
}
```

Output:-

Adjacency list :-

```
import java.util.*;
Class Graph {
  Static void addedge ( ArrayList <ArrayList <Integer>> am,
  am.get(s).add(d);          int s, int d ) {
  am.get(d).add(s);
}

Public static void main (String [] args) {
  int v = 5;
ArrayList <ArrayList <Integer>> am = new ArrayList <ArrayList <Integer>>();
for ( int i = 0; i < v ; i++ ) {
  am.add (new ArrayList <Integer> ());
addEdge (am, 0, 1);
addEdge (am, 0, 2 );
add Edge (am, 0, 3 );
add Edge (am, 1, 2 );
  Printgraph (am);
}
Static void printGraph (ArrayList <ArrayList <Integer>> am ) {
  for (int i = 0; i < am.size (); i++ ) {
    System.out.println ("\n vertex" + i + ":");
for (int j = 0; j < am.get (i).size (); j++ ) {
  System.out.println ("→ ") + am.get (i).get (j));
  }
System.out.println ( );
  }
}
}
```

Adjacency Matrix

```
import Java.util.Scanner;
Public class Represent_Graph_Adjacency_Matrix {
  Private final int Vertices;
  Private int[][] adjacency_matrix;
  Public Represent_Graph_Adjacency_Matrix (int v) {
     Vertices = V;
    adjacency_matrix = new int [vertices +1 ] [vertices +1];
  }
 Public void makeEdge (int to, int from, int edge) {
     try {
    adjacency_matrix [to] [from] = edge;
  }
 Catch (ArrayIndexOutofBoundException index)
  {
 System.out.println ("The Vertices doesnot exists");
  }
}
Public int getEdge (int to, int from)
  {
   try {
      return adjacency_matrix [to] [from];
   }
```

```
for (int i = 1 ; i <= V ; i++)
  {
   System.out.print (i + " ");
   for (int j = 1 ; j <= V ; j++)
  System.out.print (graph.getEdge (i, j) + " ");
   System.out.println ( );
   }
}
Catch ( Exception E )
 {
System.out.println ("Somethig were wrong.");
   }
Sc.close ();
```

```
Catch (ArrayIndexOutOfBoundException index)
    {
System.out.println ("The vertices does not exists");
 }
return -1;
}
Public static void main (String [] args) {
  int v, e, count = 1, to = 0, from = 0
  Scanner sc = new Scanner (System.in);
  Represent_Graph_Adjacency_Matrix graph;
  try {
   System.out.println ("Enter the number of vertices:");
   V = sc.nextInt ();
   System.out.println ("Enter the number of edges:");
     e = sc.nextInt ();
   graph = new Represent_Graph_Adjacency_Matrix (v);
   System.out.println ("Enter the edges: <to> <from>");
   while (count <= e)
     {
    to = sc.nextInt ();
    from = sc.nextInt ();
   graph.makeEdge (to, from, 1);
   count++;
  }
System.out.println ("The Adjacency matrix for the given graph is");
System.out.print ("  ");
 for (int i=1; i<= v; i++)
   System.out.print (i+ " ");
   System.out.println ();
```

DFS Traversal

```
import Java.io.*;
import Java.util.*;
Public class DFS {
  Privede int V;
  Private LinkedList<Integer> adj[];
  DFS(int v) {
    V=v;
    adj = new LinkedList[v];
    for (int i=0; i<v; ++i)
      adj[i] = new LinkedList();
  }
  Void addEdge (int v, int w) {
    adj[v].add(w);
  }
  Void DFSUntil (int v, boolean visited[]) {
    visited[v] =true;
    System.out.print (v+" ");
    Iterator<Integer> i = adj[v].listIterator();
    while (i.hasNext()) {
      int n = i.next();
      if (!visited[n])
        DFS Until (n, visited);
    }
  }
  void DFS(int v) {
  boolean visited[] = new boolean[v];
  DFS Until (v, visited);
}
Public static void main (String args[]) {
  DFS g = new DFS(4);
```

```
g.addEdge(0,1);
g.addEdge(0,2);
g.addEdge(1,2);
g.addEdge(2,0);
g.addEdge(2,3);
g.addEdge(3,3);
System.out.println("following is depth first Traversal" + (starting from vertex 2)");
g.DFS(2);
}
}
```

BFS Traversals

```
import java.io.*;
import java.util.*;
Public class BFS {
  Private int V;
  Private LinkedList<Integer> adj[];
  BFS(int v) {
  V = v;
  adj = new LinkedList[v];
  for (int i=0; i< v; ++i)
```

```
    adj[i] = new LinkedList();
  }
  Void addEdge(int v, int w){
    adj[v].add(w);
  }
  Void BFS(int s){
  boolean visited[] = new boolean[v];
  LinkedList<Integer> queue = new LinkedList<Integer>();
  visited[s] = true;
  queue.add(s);
  while(queue.size()!=0){
    s = queue.poll();
  System.out.print(s+" ");
  Iterator<Integer> i = adj[s].listIterator();
  while(i.hasNext()){
    int n = i.next();
    if(!visited[n]){
      visited[n] = true;
      queue.add[n];
    }
  }
}
}
  Public static void main(String[] args){
    Graph g = new Graph(4);
    g.addEdge(0,1);
    g.addEdge(0,2);
    g.addEdge(1,2);
    g.addEdge(2,0);
    g.addEdge(2,3);
    g.addEdge(3,3);
System.out.println("following is Breadth first Traversal")+("Starting from
      Vertex 2)");  g.Bfs;  }  }
```

Prims Algorithms.

```
import java.io.*;
import java.lang.*;
import java.util.*;

Public class Prims {
 private static final int V = 5;
  int min = Integer.MAX_VALUE, min_index = -1;
  for (int v = 0; v<V; v++)
     if (primset [v] == false && Key [v] < min ) {
         min = key [v];
         min_index = v;
      }
  return min_index;
  }
  void printprims (int Parent [], int graph [][]) {
  System.out.println ("Edge \t weight");
  for (int i = 1; i< v; i++) {
  System.out.println (Parent [i] + " " +i+ "\t" + graph [i].
                      Parent [i]]);
  }
  void primMst (int graph [ ] [ ])
      int parent [] = new int [v];
      int key [ ] = new int [v];
       Boolean mst set [ ] = new Boolean [v];
   for (int i=0; i<v ; i++) {
   Key [i] = Integer.MAX_VALUE;
   mstset [i] = false;
  }
```

```
key[0] = 0;
Parent[0] = -1;

for (int count = 0; count < V-1; count++) {
    int u = minKey(key, mstSet);
    mstSet[u] = true;
    for (int v=0; v<V; v++) {
        if (graph[u][v] != 0 && mstSet[v] == false
            && graph[u][v] < key[v]) {
            Parent[v] = u;
            key[v] = graph[u][v];
        }
    }
    PrimsMST(parent, graph);
}

Public static void main (String[] args) {
    Prims t = new Prims();
    int graph[][] = new int[][] { {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} };
    t.primMST(graph);
    }
}
```

Ans:] KRUSKAL's Algorithm.

```
import Java.io.*;
import Java.lang.*;
import Java.util.*;
Public class Graph {
  class Edges implements Comparable<Edge> {
    int src, dest, weight;
  Public int CompareTo (Edges compareEdge) {
    return this.weight - compareEdge.weight;
    }
  };
class Subset {
  int parent, rank;
  };
int V, E;
Edge Edge [];
Graph (int v, int e)
 {
   V = v;
   E = e;
  edge = new Edge [E];
 for (int i=0; i<e; ++i)
    edge [i] = new Edge ();
   }
int find ( Subset subsets [], int i)
 {
   if (Subsets [i].parent != i)
      subsets [i].parent = find (Subsets, Subsets [i].parent);
   return Subsets [i].parent;
}
```

```
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++;
    }
}
void KruskalMST() {
    Edge result[] = new Edge[v];
    int e = 0;
    int i = 0;
    for (i=0; i<v; ++i) {
        result[i] = new Edge();
        Arrays.sort(edge);
    Subset subsets[] = new Subset[v];
    for (i=0; i<v; ++i)
    subsets[i] = new Subset();
    for (int v=0; v<V; ++v) {
        subsets[v].parent = v;
        subsets[v].rank = 0;
    }
    i = 0;
    while (e < v-1)
    int x = find (subsets, next_edge.src);
    int y = find (subsets, next_edge.dest);
```

```
System.out.println("following are the edges in" + "the
                    constructed MST");
  int minimum cost = 0;
  for (i=0; i<e; ++i) {
  System.out.println(result[i].src + "....." + result
      [i].dest + "==" result[i].weight);
    Minimum cost + = result[i].weight;
  }
System.out.println("Minimum Cost Spanning Tree" + minimum Cost);
Public Static void main (String [] args) {
  int V = 4;
  int E = 5;
  Graph graph = new Graph (V, E);
  graph.edge[0].src = 0;
  graph.edge[0].dest = 1;
  graph.edge[0].weight = 10;
  graph.edge[1].src = 0;
  graph.edge[1].dest = 3;
  graph.edge[1].weight = 5;
  graph.edge[2].src = 2;
  graph.edge[2].dest = 3;
  graph.edge[2].weight = 4;
  graph.edge[3].src = 0
  Graph.edge[3].dest = 3
  graph.edge[3].weight = 5;
  graph.KruskalMST();
}
```

Dijkstra's Algorithm

```
import. Java. util. *;
import Java. io. *;
import Java. lang. *;

Public class DijkstraAlgo {
Static final int total vertex = 9;
int minimumDistance (int distance [], Boolean Spset []) {
int m = Integer. MAX_VALUE, m_index = -1
for (int vx = 0; vx < total vertex; vx++) {
 {
 if (spset [vx] = 0; vx < total vertex; vx++)
 {
if (spset [vx] == false && distance [vx] <= m)
 {
 m = distance [vx];
 m_index = vx;
 }
}
return m_index;
 }
 void printSolution (int distance [] int n)
{
System.out.println ("The shortest distance from Source 0th Node to all
        other Nodes are:");
for (int j = 0; j < n; j++)
System.out.println ("To" + j + " the shortest distance :: " + distance [j]);
}
Void dijkstra (int graph [][], int s)
 int distance [] = new int [total vertex];
for (int j = 0; j < total vertex; j++) {
   distance [j] = Integer. MAX_VALUE;
   Spset [j] = false; }
```

```
distance [s] = 0;
for (int cnt = 0; cnt < totalVertex - 1; cnt++) {
int ux = minimumDistance (distance, spset);
spset [ux] = true;
for (int vx = 0; vx < totalVertex; vx++) {
if (!spset [vx] && graph [ux] [vx] != -1 && distance [ux] !=
Integer.MAX_VALUE && distance [ux] + graph [ux][vx] <
distance [vx]) {
distance [vx] = distance [ux] + graph [ux] [vx]; }
printSolution (distance, totalVertex); }
public static void main (String args []) {
int graph [] [] = new int [] [] { {-1, 3, -1, -1, -1, -1, 7, -1},
{3, -1, 7, -1, -1, -1, 10, 4};
{-1, 7, -1, 6, -1, 2, -1, -1, 1};
{-1, -1, 6, -1, 8, 13, -1, -1, 3};
{-1, -1, -1, 8, -1, 9, -1, -1, -1};
{-1, -1, 2, 13, 9, -1, 4, -1, 5};
{-1, -1, -1, -1, -1, 4, -1, 2, 5};
{7, 10, -1, -1, -1, -1, 2, -1, 6};
{-1, 4, 1, 3, -1, 5, 5, 6, -1}};
DijkstraAlgo obj = new DijkstraExample ();
obj.dijkstra (graph, 0);
}
}
```

Interval Scheduling Problems

```
import Java.util.Arrays;
import Java.util.Comparator;
class Job {
    int start, int finish, int Profit;
  Job (int start, int finish, int profit) {
      this.start = start;
      this.finish = finish;
      this.profit = profit;
    }
}
class JobComparator implements Comparator<Job>
  {
  Public int compare (Job a, Job b) {
    return a.finish < b.finish ? -1 a.finish == b.finish ? 0 : 1;
      }
  }
Public class Weighed Interval scheduling {
  static public int binarysearch (Job Jobs [], int index) {
      int lo = 0; hi = index-1;
      while (lo <= hi)
        {
        int mid = (lo+ho)/2;
      if (Jobs [mid].finish <= Jobs [index].start)
        {
        if (Jobs (mid+1).finish <= Jobs [index]. start)
          lo = mid+1;
        else
      }   return mid;
    else
      }  hi = mid-1;
  return -1;
}
```

```
static public int schedule (Job Jobs [])
  Array.sort (Jobs, new JobComparator());
    int n = Jobs.length;
    int table [] = new int [n];
    table [0] = Jobs [0].profit;
    for (int i = 1; i < n; i++) {
      int inclProf = Jobs [i].profit;
      int l = binarySearch (Jobs, i);
      if (l != -1)
        inclProft = table [l];
      table [i] = Math.max (inclProf, table [i-1]); }
      return table [n-1];
  }

  Public static void main (String [] args) {
    Job Jobs [] = {new Job (1,2,50), new Job (3,5,20), new Job (6,19,100),
                   new Job (2,100,200)};
    System.out.println ("Optimal Profit is = "+schedule (Jobs));
  }
}
```

Interval Partioning Problem

```
import Java.util.ArrayList;
import Java.util.Comparator;
Public class Interval Partioning Problem {
  Static class pair {
      int first, second;
    Public Pair (int f, int s)
      this.first = f;
      this.second = s; } }
Public static int minimizeSegment (ArrayList<Pair> A, Pair x) {
final Comparator<Pair> arrayComparator = new Comparator<Interval
  Partioning Problem.Pair>() {
Public int compare (Pair o1, Pair o2) {
return Integer.Compare (o1.first, o2.first);
  };
  A.Sort (arrayComparator);
  A.add (new Pair (Integer.MAX_VALUE, Integer.MIN_VALUE));
  int start = X.first;
  int end = X.first - 1;
  int cnt = 0;
  for (int i=0; i<A.size(); ) {
  if (A.get(i).first <= start) {
   end = Math.max (A.get (i++).second, end); }
  else {
    start = end;
    ++cnt;
   if (A.get(i).first > end) || end >= x.second) {
   break;
  }
 }
}
```

```
        if (end < x.second) {
            return -1;
        }

    Public static void main (String args[]) {
     Arraylist <Pair> = new Arraylist <Pair> ();
     A.add (new Pair (1,3));
     A.add (new Pair (2,4));
     A.add (new Pair (2,10));
     A.add (new Pair (2,3));
     A.add (new Pair (1,1));
     Pair x = new Pair (1,10);
     System.out.Println (minimize Segment (A, x));
    }
}
```