```java
public class MinimumCopyPasteDP {
    public int find(int number){
        int res = 0;
        for(int i=2;i<=number;i++){
            while(number%i == 0){ //check if problem can be broken into smaller problem
                res+= i; //if yes then add no of smaller problems to result. If number = 25  i = 5 then 5*5 = 25 so add 5 to results
                number=number/i; // create smaller problem
            }
        }
        return res;
    }

    public static void main(String[] args) {
        MinimumCopyPasteDP m = new MinimumCopyPasteDP();
        int n = 50;
        System.out.println("Minimum Operations: " + m.find(n));

    }
}

/******************************************************************************/
public class SubarrayProductLessThanK {

    public int find(int [] arrA, int k){

        int result=0;
        int arrSize = arrA.length;
        //start point
        for (int startPoint = 0; startPoint <arrSize ; startPoint++) {
            //group sizes
            for (int grps = startPoint; grps <=arrSize ; grps++) {
                //if start point = 1 then
                //grp size = 1 , take 10
                //grp size = 2, take 10 4
                //grp size = 3, take 10 4 2 ans so on
                int product=1;
                int noElements=0;
                String tempSubArrrays ="";
                for (int j = startPoint ; j < grps ; j++) {
                    tempSubArrrays += arrA[j] + " ";
                    product *= arrA[j];
                    noElements++;
                }
                if(product<k && noElements>0){
                    System.out.print(tempSubArrrays);
                    result++;
                    System.out.println();
                }

            }
        }
        return result;
    }

    public static void main(String[] args) {
        SubarrayProductLessThanK s = new SubarrayProductLessThanK();
        int [] nums = {10,4,2,6};
        int k = 100;
        System.out.println("Sub arrays has sum less than k=100 are: " + s.find(nums, k));
    }
}
/******************************************************************************/
public class SubArraySum {
    public int findSum(int [] arrA){

        int arrSize = arrA.length;
        int totalSum = 0;
        //start point
```

```
        for (int startPoint = 0; startPoint <arrSize ; startPoint++) {
            //group sizes
            for (int grps = startPoint; grps <=arrSize ; grps++) {
                //if start point = 1 then
                //grp size = 1 , print 1
                //grp size = 2, print 1 2
                //grp size = 3, print 1 2 3 ans so on
                for (int j = startPoint ; j < grps ; j++) {
                    totalSum += arrA[j];
                }
            }
        }
        return totalSum;
    }

    public static void main(String[] args) {
        int [] arrA = {1,2,3,4};
        int sum = new SubArraySum().findSum(arrA);
        System.out.println("Sum of elements of sub arrays is: " +  sum);
    }

}

/*********************************************************************************/
public class SlidingWindowMaximumNaive {

    public void slidingWindow(int [] nums, int k){

        for (int i = 0; i <nums.length - k ; i++) {
            int max = nums[i];
            for (int j = 1; j<=k ; j++) {
                if(nums[i+j]>max)
                    max = nums[i+j];
            }
            System.out.print(max + " ");
        }
    }

    public static void main(String[] args) {
        int [] nums = {1, 2, 3, 2, 4, 1, 5, 6, 1};
        int k = 3;
        SlidingWindowMaximumNaive s = new SlidingWindowMaximumNaive();
        s.slidingWindow(nums, k);
    }
}
/*********************************************************************************/
public class RearrangeBruteForce {

    public void rearrangeArray(int [] A, int n){
        int start = 0;
        int end  = A.length-1;
        int mid = start + (end-start)/2;
        while(start<n && mid<end) {
            int left_index = start + 1;
            int right_index = mid + 1;
            while (left_index < right_index) {
                swap(A, right_index, right_index - 1);
                right_index--;
            }
            start += 2;
            mid += 1;
        }

        for (int i = 0; i <2*n ; i++) {
            System.out.print(A[i] + " ");
        }
    }
    private void swap(int A[],int m, int n){
        int temp = A[m];
```

```
138             A[m] = A[n];
139             A[n] = temp;
140         }
141
142         public static void main(String[] args) {
143             int [] A = {1,3,5,7,9,2,4,6,8,10};
144             int n = 5;
145             new RearrangeBruteForce().rearrangeArray(A,n);
146         }
147     }
148     /*************************************************************************/
149     import java.util.Arrays;
150
151     public class removeDuplicatesUsingSorting {
152         public static String removeDuplicates(String s) {
153             char[] chars = s.toCharArray();
154             Arrays.sort(chars); // O(nlogn)
155             String sbString = new String();
156             for (int i = 1; i < chars.length; i++) {
157                 if(chars[i]!=chars[i-1])
158                     sbString +=chars[i];
159             }
160             //handle the first character
161             if(chars[0]!=chars[1])
162                 sbString = chars[0] + sbString;
163             return sbString;
164         }
165
166         public static void main(String[] args) {
167             String s = "tutorialhorizon";
168             System.out.println(removeDuplicates(s));
169         }
170     }
171     /*************************************************************************/
172     public class MedianBinary {
173
174         public float find(int [] a, int start_a, int end_a, int [] b, int start_b, int
        end_b){
175
176             if(end_a-start_a+1==2 && end_b-start_b+1==2){
177                 float x = Math.max(a[start_a],b[start_b]);
178                 float y = Math.min(a[end_a],b[end_b]);
179                 return (x+y)/2;
180             }
181
182             float median_a = getMedian(a, start_a, end_a);
183             float median_b = getMedian(b, start_b, end_b);
184
185             int mid_a = (start_a+end_a)/2;
186             int mid_b = (start_b+end_b)/2;
187             if(median_a>median_b){
188                 return find(a,start_a,mid_a,b,mid_b,end_b);
189             }else{
190                 return find(a,mid_a,end_a,b,start_b,mid_b);
191             }
192
193         }
194
195         public float getMedian(int [] x, int start, int end){
196             int size = end-start+1;
197             double median;
198             if(size%2==0){
199                 float m = x[start+(size/2)];
200                 float n = x[start+(size-1)/2];
201                 return (m+n)/2;
202             }else{
203                 return  x[start+(size-1)/2];
204             }
205         }
```

```
206
207         public static void main(String[] args) {
208             MedianBinary m = new MedianBinary();
209             int [] a = {2,6,9,10,11};
210             int [] b = {1,5,7,12,15};
211             float x = m.find(a,0,a.length-1,b,0,b.length-1);
212             System.out.println("Median of combined sorted array is: " + x);
213         }
214     }
215     /*************************************************************************/
216
217
218     public class TwoNonRepeatingXOR {
219
220         public void find(int [] arrA){
221             //xor will the xor of two non repeating elements
222             //we know that in a XOR b, any particular bit is set if that bit is set in
                either a or b
223             //we can use this to divide the array elements into two groups where each group
                will be responsible
224             // to get a and b
225             int xor = arrA[0];
226             for (int i = 1; i <arrA.length ; i++) {
227                 xor ^= arrA[i];
228             }
229             //get the right most set bit
230             int right_most_set_bit = xor & ~ (xor -1);
231
232             //divide the array elements into 2 groups
233             int a=0,b=0;
234             for (int i = 0; i <arrA.length ; i++) {
235                 int x = arrA[i];
236                 if((x & right_most_set_bit)!=0)
237                     a ^= x;
238                 else
239                     b ^= x;
240             }
241             System.out.println("Non Repeating Elements are: " + a + " and " + b);
242         }
243         public static void main(String[] args) {
244             TwoNonRepeatingXOR t = new TwoNonRepeatingXOR();
245             int [] arrA = {4,5,4,5,3,2,9,3,9,8};
246             t.find(arrA);
247
248         }
249     }
250     /*************************************************************************/
251
252     import java.util.Arrays;
253     public class FindSingleElement {
254         public static void find (int [] arrA){
255             int singleElement =0;
256             for (int i = 0; i <32 ; i++) { //this is for calculating for every position in
                32 bit integer
257                 int y = (1 << i);
258                 int tempSum = 0;
259                 for (int j = 0; j <arrA.length ; j++) {
260                     if((arrA[j] & y)>=1) //if that particular bit is set for the ith
                        position, add 1 to sum (w.r.t that bit)
261                         tempSum = tempSum +1;
262                 }
263                 //if bits are not multiple of 3 then that bit belongs to the element
                    appearing single time
264                 if((tempSum%3)==1) {
265                     singleElement = singleElement | y;
266                 }
267             }
268             System.out.println("Element appearing once is: " + singleElement);
269         }
```

```
270
271     public static void main(String[] args) {
272         int arrA [] = {1, 4, 5, 6, 1, 4, 6, 1, 4, 6};
273         System.out.print(Arrays.toString(arrA));
274         find(arrA);
275     }
276
277 }
278 /*************************************************************************/
279 public class Boyer_Moore {
280     public static void find(int [] arrA) {
281         int size = arrA.length;
282         if(size==0)
283             return;
284
285         int majorityElement = arrA[0];
286         int count = 1;
287         for (int i = 1; i <size ; i++) {
288             if(majorityElement==arrA[i]){
289                 count++;
290             }else if(count==0){
291                 majorityElement = arrA[i];
292                 count = 1;
293             }else {
294                 count --;
295             }
296         }
297         //check if majorityElement is appearing more than n/2 times
298         count = 0;
299         for (int i = 0; i < size ; i++) {
300             if(arrA[i]==majorityElement){
301                 count++;
302             }
303         }
304         if (count>size/2)
305             System.out.println("(Boyer_Moore)Element appearing more than n/2 times: " +
            majorityElement);
306         else
307             System.out.println("No element appearing more than n/2 times");
308     }
309
310     public static void main(String[] args) {
311         int [] arrA = {1,3,5,5,5,5,4,1,5};
312         find(arrA);
313     }
314 }
315 /*************************************************************************/
316 public class StockSingleSellBruteForce {
317
318     public static void maxProfit(int [] prices){
319         int profit = -1;
320         int buyDateIndex = prices[0];
321         int sellDateIndex = prices[0];
322         for (int i = 0; i <prices.length ; i++) {
323             for (int j = i; j <prices.length ; j++) {
324                 if(prices[j]>prices[i] && (prices[j]-prices[i]>profit)) {
325                     profit = prices[j] - prices[i];
326                     buyDateIndex = i;
327                     sellDateIndex = j;
328                 }
329             }
330         }
331         System.out.println("Maximum Profit: " + profit + ", buy date index: " +
            buyDateIndex +
332                 ", sell date index: " + sellDateIndex);
333     }
334
335     public static void main(String[] args) {
336         int [] prices = { 200, 500, 1000, 700, 30, 400, 900, 400, 50};
```

```
337             maxProfit(prices);
338         }
339     }
340     /*************************************************************************/
341     public class MaximumSubArray {
342 
343         public int maxSubArray(int [] A){
344             if(A.length==0)//if length  = 0, return 0
345                 return 0;
346             else
347                 return maxSubArray(A, 0, A.length-1);
348         }
349 
350         public int maxSubArray(int [] A, int start, int end){
351             if(start==end){
352                 return A[start]; //if only one element, return that
353             }
354             int mid = start + (end-start)/2;
355             int leftMaxSum = maxSubArray(A, start, mid);
356             int rightMaxSum = maxSubArray(A, mid+1, end);
357             //lets calculate the part in which sub array will start in the left half and
            ends in right half
358             int sum = 0;
359             int leftMidMax =0;
360             for (int i = mid; i >=start ; i--) {
361                 sum += A[i];
362                 if(sum>leftMidMax)
363                     leftMidMax = sum;
364             }
365             sum = 0;
366             int rightMidMax =0;
367             for (int i = mid+1; i <=end ; i++) {
368                 sum += A[i];
369                 if(sum>rightMidMax)
370                     rightMidMax = sum;
371             }
372             int centerSum = leftMidMax + rightMidMax;
373             return Math.max(centerSum, Math.max(leftMaxSum, rightMaxSum));
374         }
375         public static void main(String[] args) {
376             int [] A = {-2, 12, -5, 10, -1, -6, 4};
377             MaximumSubArray m =  new MaximumSubArray();
378             System.out.println("Maximum Sub Array sum is : " + m.maxSubArray(A));
379         }
380     }
381     /*************************************************************************/
382     public class TwoRepeatingBruteForce {
383         //this solution will work even if all the numbers are not in the range of 1 to n
384         public static void twoRepeating(int [] A){
385             System.out.println("Repeated Elements: ");
386             for (int i = 0; i <A.length ; i++) {
387                 for (int j = i+1; j <A.length ; j++) {
388                     if(A[i]==A[j]){
389                         System.out.print(A[i] + " ");
390                     }
391                 }
392             }
393         }
394 
395         public static void main(String[] args) {
396             int [] A = {1,5,2,4,8,9,3,1,4,0};
397             twoRepeating(A);
398         }
399     }
400     /*************************************************************************/
401     import java.util.Arrays;
402 
403     public class CheckDuplicatesUsingSorting {
404         public void hasDuplicatesUsingSorting(int [] arrA) {
```

```
        Arrays.sort(arrA);
        for (int i = 0; i < arrA.length-1; i++) {
            if(arrA[i]==arrA[i+1]){
                System.out.println("Array has duplicates : " + arrA[i]);
            }
        }
    }
    public static void main(String[] args) {
        int a [] = {1, 6, 5, 2, 3, 3, 2};
        new CheckDuplicatesUsingSorting().hasDuplicatesUsingSorting(a);
    }
}
/*****************************************************************************/

public class SnakeSequence {

    public int getMaxSequence(int [][] matrix){

        int rows = matrix.length;
        int cols = matrix[0].length;
        int maxLenth =1;
        int maxRow = 0;
        int maxCol = 0;

        //create result matrix
        int [][] result = new int [rows][cols];

        //if no sequence is found then every cell itself is a sequence of length 1
        for (int i = 0; i <rows ; i++) {
            for (int j = 0; j <cols ; j++) {
                result[i][j] =1;
            }
        }

        for (int i = 0; i <rows ; i++) {
            for (int j = 0; j <cols ; j++) {
                if(i!=0 || j!=0){
                    //check from left
                    if(i>0 && Math.abs(matrix[i][j]-matrix[i-1][j])==1){
                        result[i][j] = Math.max(result[i][j],
                                result[i-1][j]+1);
                        if(maxLenth<result[i][j]){
                            maxLenth = result[i][j];
                            maxRow = i;
                            maxCol = j;
                        }
                    }

                    //check from top
                    if(j>0 && Math.abs(matrix[i][j]-matrix[i][j-1])==1){
                        result[i][j] = Math.max(result[i][j],
                                result[i][j-1]+1);
                        if(maxLenth<result[i][j]){
                            maxLenth = result[i][j];
                            maxRow = i;
                            maxCol = j;
                        }
                    }
                }
            }
        }

        //Now we will check the max entry in the result[][].
        System.out.println("Max Snake Sequence : " + maxLenth);
        printPath(matrix, result, maxLenth, maxRow, maxCol);
        return 0;
    }

    public void printPath(int [][] matrix, int [][] result, int maxLength, int maxRow,
```

```
        int maxCol){
            int len =  maxLength;
            while(maxLength>=1){
                System.out.print(" - " + matrix[maxRow][maxCol]);
                if(maxRow>0 && Math.abs(result[maxRow-1][maxCol]-result[maxRow][maxCol])==1){
                    maxRow--;
                }else if(maxCol>0 &&
                Math.abs(result[maxRow][maxCol-1]-result[maxRow][maxCol])==1){
                    maxCol--;
                }
                maxLength--;
            }
        }


        public static void main(String[] args) {
            int arrA [][] = {{1, 2, 1, 2},
                             {7, 7, 2, 5},
                             {6, 4, 3, 4},
                             {1, 2, 2, 5}};
            SnakeSequence snakeSequence = new SnakeSequence();
            snakeSequence.getMaxSequence(arrA);
        }
    }
    /**************************************************************************/
    public class NoOfPathObstruction {

        public int count(int [][] arrA, int row, int col){
            //base case
            //check if last cell is reached since after that only one path
            if(row==arrA.length-1 && col==arrA.length-1){
                return 1;
            }
            int right =0;
            int down = 0;
            if(row!=arrA.length-1 && arrA[row+1][col]!=-1){
                right = count(arrA, row+1, col);
            }
            if(col!=arrA.length-1 && arrA[row][col+1]!=-1){
                down = count(arrA, row, col+1);
            }
            return right + down;
        }

        public int countDP(int [][] arrA){
            int result [][] = arrA;

            for (int i = 1; i <result.length ; i++) {
                for (int j = 1; j <result.length ; j++) {
                    if(result[i][j]!=-1){
                        result[i][j]=0;
                        if(result[i-1][j]>0)
                            result[i][j]+=result[i-1][j];
                        if(result[i][j-1]>0)
                            result[i][j]+=result[i][j-1];
                    }
                }
            }

            return result[arrA.length-1][arrA.length-1];
        }

        public static void main(String[] args) {
            int arrA [][] = {{1,1,1},{1,-1,1},{1,-1,1}};
            NoOfPathObstruction noOfPaths = new NoOfPathObstruction();
            System.out.println("No Of Path (Recursion):- " +noOfPaths.count(arrA,0,0));
            System.out.println("No Of Path (DP):- " +noOfPaths.countDP(arrA));
        }
    }
```

```java
/*****************************************************************************/
public class Diagonals {

    public static void print(int [][] a){

        //print first half
        int row =0;
        int col;

        while(row<a.length){
            col =0;
            int rowTemp = row;
            while(rowTemp>=0){
                System.out.print(a[rowTemp][col] + " ");
                rowTemp--;
                col++;
            }
            System.out.println();
            row++;
        }

        //print second half
        col = 1;

        while(col<a.length){
            int colTemp = col;
            row = a.length-1;
            while(colTemp<=a.length-1){
                System.out.print(a[row][colTemp] + " ");
                row--;
                colTemp++;
            }
            System.out.println();
            col++;
        }

    }

    public static void main(String[] args) {
        int [][] a = {{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16}};
        print(a);

    }
}
/*****************************************************************************/
public class MaximumSubArray {
    // Kadane algorithm
    public int kandane(int[] arrA) {
        int max_end_here = 0;
        int max_so_far = 0;
        for (int i = 0; i < arrA.length; i++) {
            max_end_here += arrA[i];
            if (max_end_here < 0) {
                max_end_here = 0;
            }
            if (max_so_far < max_end_here) {
                max_so_far = max_end_here;
            }
        }
        return max_so_far;
    }

    // Below modification will allow the program to work even if all the
    // elements in the array are negative
    public int KandaneModify(int[] arrA) {
        int max_end_here = arrA[0];
        int max_so_far = arrA[0];
        for(int i=1;i<arrA.length;i++){
            max_end_here = Math.max(arrA[i], max_end_here+arrA[i]);
```

```
            max_so_far = Math.max(max_so_far,max_end_here);
        }
        return max_so_far;
    }

    public static void main(String args[]) {
        int arrA[] = { 1, 2, -3, -4, 2, 7, -2, 3 };
        MaximumSubArray i = new MaximumSubArray();
        System.out.println("Maximum subarray is  " + i.kandane(arrA));
        int arrB[] = { -2, -3, -4, -2, -7, -2, -3,-11 };
        System.out.println("Maximum Subarray when all elements are negative : " +
        i.KandaneModify(arrB));
    }
}

/*****************************************************************************/
public class SmallestRangeInKList {

    public int size;
    public HeapNode[] Heap;
    public int position;
    static int gMax;
    static int gMin;
    int currMax; //tracks the max entry in the heap
    int range = Integer.MAX_VALUE;

    public SmallestRangeInKList(int k) {
        this.size = k;
        Heap = new HeapNode[k + 1]; // size+1 because index 0 will be empty
        position = 0;
        Heap[0] = new HeapNode(0, -1); // put some junk values at 0th index node
    }

    public int merge(int[][] A, int k, int n) {
        int nk = n * k;
        int count = 0;
        int[] ptrs = new int[k];
        // create index pointer for every list.
        for (int i = 0; i < ptrs.length; i++) {
            ptrs[i] = 0;
        }
        for (int i = 0; i < k; i++) {
            insert(A[i][ptrs[i]], i); // insert the element into heap

        }
        while (count < nk) {
            HeapNode h = extractMin(); // get the min node from the heap.
            int min = h.data; // this is min among all the values in the heap
            if (range > currMax - min) { // check if current difference > range
                gMin = min;
                gMax = currMax;
                range = gMax - gMin;
            }
            ptrs[h.listNo]++; // increase the particular list pointer
            if (ptrs[h.listNo] < n) { // check if list is not burns out
                insert(A[h.listNo][ptrs[h.listNo]], h.listNo); // insert the
                                                               // next element
                                                               // from the list
            } else {
                return range; // if any of this list
                              // burns out, return range
            }
            count++;
        }
        return range;
    }

    public void insert(int data, int listNo) {
        // keep track of max element entered in Heap till now
```

```
678             if (data != Integer.MAX_VALUE && currMax < data) {
679                 currMax = data;
680             }
681             if (position == 0) { // check if Heap is empty
682                 Heap[position + 1] = new HeapNode(data, listNo); // insert the first
683                                                                  // element in
684                                                                  // heap
685                 position = 2;
686             } else {
687                 Heap[position++] = new HeapNode(data, listNo);// insert the element
688                                                               // to the end
689                 bubbleUp(); // call the bubble up operation
690             }
691         }
692 
693         public HeapNode extractMin() {
694             HeapNode min = Heap[1]; // extract the root
695             Heap[1] = Heap[position - 1]; // replace the root with the last element
696                                          // in
697                                          // the heap
698             Heap[position - 1] = null; // set the last Node as NULL
699             position--; // reduce the position pointer
700             sinkDown(1); // sink down the root to its correct position
701             return min;
702         }
703 
704         public void sinkDown(int k) {
705             int smallest = k;
706             // check which is smaller child , 2k or 2k+1.
707             if (2 * k < position && Heap[smallest].data > Heap[2 * k].data) {
708                 smallest = 2 * k;
709             }
710             if (2 * k + 1 < position && Heap[smallest].data > Heap[2 * k + 1].data) {
711                 smallest = 2 * k + 1;
712             }
713             if (smallest != k) { // if any if the child is small, swap
714                 swap(k, smallest);
715                 sinkDown(smallest); // call recursively
716             }
717 
718         }
719 
720         public void swap(int a, int b) {
721             // System.out.println("swappinh" + mH[a] + " and " + mH[b]);
722             HeapNode temp = Heap[a];
723             Heap[a] = Heap[b];
724             Heap[b] = temp;
725         }
726 
727         public void bubbleUp() {
728             int pos = position - 1; // last position
729             while (pos > 0 && Heap[pos / 2].data > Heap[pos].data) { // check if its
730                                                                      // parent is
731                 // greater.
732                 HeapNode y = Heap[pos]; // if yes, then swap
733                 Heap[pos] = Heap[pos / 2];
734                 Heap[pos / 2] = y;
735                 pos = pos / 2; // make pos to its parent for next iteration.
736             }
737         }
738 
739         public static void main(String[] args) {
740             // TODO Auto-generated method stub
741             int[][] A = new int[3][];
742             A[0] = new int[] { 3, 10, 15, 24 };
743             A[1] = new int[] { 0, 1, 2, 20 };
744             A[2] = new int[] { 1, 18, 21, 30 };
745 
746             SmallestRangeInKList m = new SmallestRangeInKList(A.length);
```

```
747             int rng = m.merge(A, A.length, A[0].length);
748             System.out.println("Smallest Range is: " + rng + " from " + gMin
749                     + " To " + gMax);
750         }
751
752     }
753
754     class HeapNode {
755         int data;
756         int listNo;
757
758         public HeapNode(int data, int listNo) {
759             this.data = data;
760             this.listNo = listNo;
761         }
762     }
763     /*************************************************************************/
764     import java.util.Comparator;
765     import java.util.PriorityQueue;
766
767     public class MaxRevenueTickets {
768
769         PriorityQueue<Integer> pq;
770
771         // we will create a max heap
772         public MaxRevenueTickets(int length) {
773             pq = new PriorityQueue<>(length, new Comparator<Integer>() {
774
775                 @Override
776                 public int compare(Integer o1, Integer o2) {
777                     // TODO Auto-generated method stub
778                     return o2 - o1;
779                 }
780             });
781         }
782
783         public int calculate(int[] windowsTickets, int tickets) {
784
785             int revenue = 0;
786             // insert the all the elements of an array into the priority queue
787             for (int i = 0; i < windowsTickets.length; i++) {
788                 pq.offer(windowsTickets[i]);
789             }
790
791             while (tickets > 0) {
792                 int ticketPrice = pq.poll();
793                 revenue += ticketPrice;
794                 pq.offer(--ticketPrice);
795                 tickets--;
796             }
797             return revenue;
798         }
799
800         public static void main(String[] args) {
801             int[] windowsTickets = { 5, 1, 7, 10, 11, 9 };
802             int noOfTickets = 5;
803             MaxRevenueTickets mx = new MaxRevenueTickets(windowsTickets.length);
804             System.out.println("Max revenue generated by selling " + noOfTickets
805                     + " tickets: " + mx.calculate(windowsTickets, noOfTickets));
806
807         }
808     }
809     /*************************************************************************/
810     import java.util.*;
811     public class MergeKSortedArrays {
812
813         public int size;
814         public HeapNode[] Heap;
815         public int position;
```

```
816         int[] result;
817
818         public MergeKSortedArrays(int k) {
819             this.size = k;
820             Heap = new HeapNode[k + 1]; // size+1 because index 0 will be empty
821             position = 0;
822             Heap[0] = new HeapNode(0, -1); // put some junk values at 0th index node
823         }
824
825         public int[] merge(int[][] A, int k, int n) {
826             int nk = n * k;
827             result = new int[nk];
828             int count = 0;
829             int[] ptrs = new int[k];
830             // create index pointer for every list.
831             for (int i = 0; i < ptrs.length; i++) {
832                 ptrs[i] = 0;
833             }
834             for (int i = 0; i < k; i++) {
835                 if (ptrs[i] < n) {
836                     insert(A[i][ptrs[i]], i); // insert the element into heap
837                 } else {
838                     insert(Integer.MAX_VALUE, i); // if any of this list burns out, insert
                        +infinity
839                 }
840
841             }
842             while (count < nk) {
843                 HeapNode h = extractMin(); // get the min node from the heap.
844                 result[count] = h.data; // store node data into result array
845                 ptrs[h.listNo]++; // increase the particular list pointer
846                 if (ptrs[h.listNo] < n) { // check if list is not burns out
847                     insert(A[h.listNo][ptrs[h.listNo]], h.listNo); // insert the next
                        element from the list
848                 } else {
849                     insert(Integer.MAX_VALUE, h.listNo); // if any of this list burns out,
                        insert +infinity
850                 }
851                 count++;
852             }
853             return result;
854         }
855
856         public void insert(int data, int listNo) {
857             if (position == 0) { // check if Heap is empty
858                 Heap[position + 1] = new HeapNode(data, listNo); // insert the first
                    element in heap
859                 position = 2;
860             } else {
861                 Heap[position++] = new HeapNode(data, listNo);// insert the element to the
                    end
862                 bubbleUp(); // call the bubble up operation
863             }
864         }
865
866         public HeapNode extractMin() {
867             HeapNode min = Heap[1]; // extract the root
868             Heap[1] = Heap[position - 1]; // replace the root with the last element in the
                heap
869             Heap[position - 1] = null; // set the last Node as NULL
870             position--; // reduce the position pointer
871             sinkDown(1); // sink down the root to its correct position
872             return min;
873         }
874
875         public void sinkDown(int k) {
876             int smallest = k;
877             // check which is smaller child , 2k or 2k+1.
878             if (2 * k < position && Heap[smallest].data > Heap[2 * k].data) {
```

```
            smallest = 2 * k;
        }
        if (2 * k + 1 < position && Heap[smallest].data > Heap[2 * k + 1].data) {
            smallest = 2 * k + 1;
        }
        if (smallest != k) { // if any if the child is small, swap
            swap(k, smallest);
            sinkDown(smallest); // call recursively
        }

    }

    public void swap(int a, int b) {
        // System.out.println("swappinh" + mH[a] + " and " + mH[b]);
        HeapNode temp = Heap[a];
        Heap[a] = Heap[b];
        Heap[b] = temp;
    }

    public void bubbleUp() {
        int pos = position - 1; // last position
        while (pos > 0 && Heap[pos / 2].data > Heap[pos].data) { // check if its parent
        is greater.
            HeapNode y = Heap[pos]; // if yes, then swap
            Heap[pos] = Heap[pos / 2];
            Heap[pos / 2] = y;
            pos = pos / 2; // make pos to its parent for next iteration.
        }
    }

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        int[][] A = new int[5][];
        A[0] = new int[] { 1, 5, 8, 9 };
        A[1] = new int[] { 2, 3, 7, 10 };
        A[2] = new int[] { 4, 6, 11, 15 };
        A[3] = new int[] { 9, 14, 16, 19 };
        A[4] = new int[] { 2, 4, 6, 9 };
        MergeKSortedArrays m = new MergeKSortedArrays(A.length);
        int[] op = m.merge(A, A.length, A[0].length);
        System.out.println(Arrays.toString(op));
    }

}

// Every Node will store the data and the list no from which it belongs
class HeapNode {
    int data;
    int listNo;

    public HeapNode(int data, int listNo) {
        this.data = data;
        this.listNo = listNo;
    }
}
/*****************************************************************************/
import java.util.*;
public class FixIndexes {
    public static int[] fix(int[] A) {
        for (int i = 0; i < A.length; i++) {
            if (A[i] != -1 && A[i] != i) {
                int x = A[i];
                while (A[x] != -1 && A[x] != x) { // check if desired place is not vacate
                    int y = A[x]; // store the value from desired place
                    A[x] = x; // place the x to its correct position
                    x = y; // now y will become x, now search the place for x
                }
                A[x] = x; // place the x to its correct position
                if(A[i]!=i){//check if while loop hasn't set the correct value at A[i]
```

```
                    A[i] = -1; // if not then put -1 at the vacated place
                }

            }
        }
        return A;
    }

    public static void main(String[] args) {
        int A[] = { -1, -1, 6, 1, 9, 3, 2, -1, 4, -1 };
        System.out.println("Fixed Indexed Array " + Arrays.toString(fix(A)));
        int B[] = { 19, 7, 0, 3, 18, 15, 12, 6, 1, 8, 11, 10, 9, 5, 13, 16, 2,
                14, 17, 4, };
        System.out.println("Fixed Indexed Array " + Arrays.toString(fix(B)));

    }

}
/*****************************************************************************/
public class minHeap {
    public int capacity;
    public int [] mH;
    public int currentSize;
    public minHeap(int capacity){
        this.capacity=capacity;
        mH = new int [capacity+1];
       currentSize =0;
    }
    public void createHeap(int [] arrA){
        if(arrA.length>0){
            for(int i=0;i<arrA.length;i++){
                insert(arrA[i]);
            }
        }
    }
    public void display(){
        for(int i=1;i<mH.length;i++){
            System.out.print(" " + mH[i]);
        }
        System.out.println("");
    }
    public void insert(int x) {
        if(currentSize==capacity){
            System.out.println("heap is full");
            return;
        }
        currentSize++;
        int idx = currentSize;
        mH[idx] = x;
        bubbleUp(idx);
    }

    public void bubbleUp(int pos) {
        int parentIdx = pos/2;
        int currentIdx = pos;
        while (currentIdx > 0 && mH[parentIdx] > mH[currentIdx]) {

            swap(currentIdx,parentIdx);
            currentIdx = parentIdx;
            parentIdx = parentIdx/2;
        }
    }

    public int extractMin() {
        int min = mH[1];
        mH[1] = mH[currentSize];
        mH[currentSize] = 0;
        sinkDown(1);
        currentSize--;
```

```
1016            return min;
1017        }
1018
1019        public void sinkDown(int k) {
1020            int smallest = k;
1021            int leftChildIdx = 2 * k;
1022            int rightChildIdx = 2 * k+1;
1023            if (leftChildIdx < heapSize() && mH[smallest] > mH[leftChildIdx]) {
1024                smallest = leftChildIdx;
1025            }
1026            if (rightChildIdx < heapSize() && mH[smallest] > mH[rightChildIdx]) {
1027                smallest = rightChildIdx;
1028            }
1029            if (smallest != k) {
1030
1031                swap(k, smallest);
1032                sinkDown(smallest);
1033            }
1034        }
1035
1036        public void swap(int a, int b) {
1037            int temp = mH[a];
1038            mH[a] = mH[b];
1039            mH[b] = temp;
1040        }
1041        public boolean isEmpty() {
1042            return currentSize == 0;
1043        }
1044
1045        public int heapSize(){
1046            return currentSize;
1047        }
1048
1049        public static void main(String args[]){
1050            int arrA [] = {3,2,1,7,8,4,10,16,12};
1051            System.out.print("Original Array : ");
1052            for(int i=0;i<arrA.length;i++){
1053                System.out.print("  " + arrA[i]);
1054            }
1055            minHeap m = new minHeap(arrA.length);
1056            System.out.print("\nMin-Heap : ");
1057            m.createHeap(arrA);
1058            m.display();
1059            System.out.print("Extract Min :");
1060            for(int i=0;i<arrA.length;i++){
1061                System.out.print("  " + m.extractMin());
1062            }
1063        }
1064    }
1065    /*************************************************************************/
1066    public class MagicIndex {
1067        // perform modified binary search
1068        public int search(int[] A, int start, int end) {
1069            if (start <= end) {
1070                int mid = (start + end) / 2;
1071                if (mid == A[mid]) // check for magic index.
1072                    return mid;
1073                if (mid > A[mid]) { // If mid>A[mid] means fixed point might be on
1074                                    // the right half of the array
1075                    return search(A, mid + 1, end);
1076                } else {// If mid<A[mid] means fixed point might be on
1077                    // the left half of the array
1078                    return search(A, start, mid - 1);
1079                }
1080            }
1081            return -1;
1082        }
1083
1084        public static void main(String[] args) {
```

```
        // TODO Auto-generated method stub
        int[] A = { -1, 0, 1, 2, 4, 10 };
        MagicIndex m = new MagicIndex();
        System.out.println("Magic index " + m.search(A, 0, A.length - 1));
    }

}
/*****************************************************************************/
public class AllCombinations {

    static int[] B = { 1, 2, 3 };

    public void combinations(int[] A, int x) {
        if (x == A.length - 1) {
            A[x] = 0; // last digit, don't select it
            printArray(A); // print the set
            A[x] = 1; //// last digit, select it
            printArray(A);
            return;
        }
        A[x] = 0;   //either you will not select this digit
        combinations(A, x + 1);
        A[x] = 1;  //either you will select this digit
        combinations(A, x + 1);
    }

    public void printArray(int[] A) {
        boolean isNULL = true;
        System.out.print("{");
        for (int i = 0; i < B.length; i++) {
            if (A[i] == 1) {
                System.out.print(B[i] + "");
                isNULL = false;
            }
        }
        if (isNULL == false) {
            System.out.print("}");
            System.out.print("  ");
        }

        if (isNULL) {
            System.out.print("Empty");
            System.out.print("} ");
        }
    }

    public static void main(String[] args) {
        AllCombinations a = new AllCombinations();
        int[] A = new int[B.length];
        a.combinations(A, 0);

    }

}
/*****************************************************************************/
import java.util.Stack;

public class TrackMaxInStack {

    // objective here is to keep track of maximum value in a stack of integers
    // create another another Stack which will keep track of maximum
    Stack<Integer> main = new Stack<>();
    Stack<Integer> track = new Stack<>();

    public void insert(int x) {
        if (main.isEmpty()) { // if stack is empty, insert the number in both
                              // stacks
            main.add(x);
            track.add(x);
```

```
        } else {
            // check if number in Stack(track) is bigger than x
            // which ever is bigger, insert it into Stack

            int a = track.peek();
            track.add(Math.max(a, x));
            main.add(x); // insert it into main stack.
        }
    }

    public int remove() {
        if (!main.isEmpty()) { // pop the top elements
            track.pop();
            return main.pop();
        }
        return 0;
    }

    public int getMax() {
        return track.peek();
    }

    public static void main(String[] args) {
        TrackMaxInStack i = new TrackMaxInStack();
        i.insert(4);
        i.insert(2);
        i.insert(14);
        i.insert(1);
        i.insert(18);
        System.out.println("Max Element is " + i.getMax());
        System.out.println("Removing Element " + i.remove());
        System.out.println("Max Element is " + i.getMax());
    }

}
/*****************************************************************************/
public class PrintValidParentheses {

    public static void Validparentheses(int openP, int closeP, String string) {
        if (openP == 0 && closeP == 0) // mean all opening and closing in
                                       // string,
                                       // print it
            System.out.println(string);
        if (openP > closeP) // means closing parentheses is more than open ones
            return;
        if (openP > 0)
            Validparentheses(openP - 1, closeP, string + "("); // put ( and
                                                               // reduce
                                                               // the count by
                                                               // 1
        if (closeP > 0)
            Validparentheses(openP, closeP - 1, string + ")"); // put ) and
                                                               // reduce
                                                               // the count by
                                                               // 1
    }

    public static void printParentheses(int n) {
        Validparentheses(n / 2, n / 2, "");
    }

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        int n = 4;
        printParentheses(n);
    }
}
/*****************************************************************************/
```

```java
import java.util.Arrays;

public class CountingSort {

    public int[] sort(int[] A) {
        int[] Result = new int[A.length + 1];
        int[] Count = new int[A.length + 1];

        for (int i = 0; i < Count.length; i++) {
            Count[i] = 0; // put count for every element as 0
        }
        // Count[] will store the counts of each integer in the given array
        for (int i = 0; i < A.length; i++) {
            int x = Count[A[i]];
            x++;
            Count[A[i]] = x;
        }
        // • Update the Count[] so that each index will store the sum till
        // previous step. (Count[i]=Count[i] + Count[i-1]).
        // Now updated Count[] array will reflect the actual position of each
        // integer in Result[].
        for (int i = 1; i < Count.length; i++) {
            Count[i] = Count[i] + Count[i - 1];
        }
        // • Now navigate the input array taking one element at a time,
        // Count[input[i]] will tell you the index position of input[i] in
        // Result[]. When you do that, decrease the count in Count[input[i]] by
        // 1.
        for (int i = A.length - 1; i >= 0; i--) {
            int x = Count[A[i]];
            Result[x] = A[i];
            x--;
            Count[A[i]] = x;
        }
        return Result;

    }

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        int input[] = { 2, 1, 4, 5, 7, 1, 1, 8, 9, 10, 11, 14, 15, 3, 2, 4 };
        System.out.println("Orginal Array " + Arrays.toString(input));
        CountingSort c = new CountingSort();
        int[] B = c.sort(input);
        System.out.println("Sorted Array : " + Arrays.toString(B));

    }

}

/*****************************************************************************/
public class CheckArrayHasConsecutiveNos {
    public Boolean WihtOutAuxArray(int [] arrA){
        //this method with work if numbers are non negative
        int max = findMax(arrA);
        int min = findMin(arrA);
        if(arrA.length!=max-min+1) return false;
        for(int i = 0;i<arrA.length;i++){
            arrA[i] = arrA[i]-min+1;
        }
        for(int i = 0;i<arrA.length;i++){
            int x  = Math.abs(arrA[i]);
            if(arrA[x-1]>0){
                arrA[x-1] = arrA[x-1]*-1;
            }
            else{
                return false;
            }
        }
```

```
            return true;
        }
        public Boolean withAuxArray(int [] arrA){
            // this method with work even if numbers are negative
            int []  aux = new int [arrA.length];
            int max = findMax(arrA);
            int min = findMin(arrA);
            if(arrA.length!=max-min+1) return false;
            for(int i = 0;i<arrA.length;i++){
                arrA[i] = arrA[i]-min;
                aux[i] = 0;
            }
            for(int i = 0;i<arrA.length;i++){
                if(aux[arrA[i]]==0){
                    aux[arrA[i]]=1;
                }
                else{
                    return false;
                }
            }
            //If we have reached till here means , we satisfied all the requirements
            return true;
        }
        public int findMax(int [] arrA){
    // find the maximum in array
            int max = arrA[0];
            for(int i = 1;i<arrA.length;i++){
                if(max<arrA[i]){
                    max = arrA[i];
                }
            }
            return max;
        }
        public int findMin(int [] arrA){
    // find the minimum in array

            int min = arrA[0];
            for(int i = 1;i<arrA.length;i++){
                if(min>arrA[i]){
                    min = arrA[i];
                }
            }
            return min;
        }
        public static void main (String[] args) throws java.lang.Exception
        {
            int [] arrA = {21,24,22,26,23,25};
            CheckArrayHasConsecutiveNos i = new CheckArrayHasConsecutiveNos();
            System.out.println(i.withAuxArray(arrA));
            int [] arrB = {11,10,12,14,13};
            System.out.println(i.WihtOutAuxArray(arrB));
            int [] arrC = {11,10,14,13};
            System.out.println(i.WihtOutAuxArray(arrC));

        }
    }
    /******************************************************************************/
    public class MinimumSubArraySum {

        public void minSubArray(int[] arrA, int x) {
            int start = 0;
            int ansEnd = 0;
            int ansStart = 0;
            int currSum = 0;
            int minLen = arrA.length;
            for (int i = 0; i <= arrA.length; i++) {
                while (currSum > x) {
                    currSum = currSum - arrA[start];
                    if (i - start <= minLen) {
```

```
                    minLen = (i - start);
                    ansEnd = i;
                    ansStart = start;
                }
                start++;
            }
            if (i < arrA.length) {
                currSum = currSum + arrA[i];
            }
        }
        System.out.println("Minimum length of subarray to get " + x + " is : "
                + minLen);
        System.out.print("SubArray is:");
        for (int i = ansStart; i < ansEnd; i++) {
            System.out.print("   " + arrA[i]);
        }
    }

    public static void main(String[] args) throws java.lang.Exception {
        int[] arrA = { 1, 10, 3, 40, 18 };
        MinimumSubArraySum i = new MinimumSubArraySum();
        i.minSubArray(arrA, 50);
    }
}
/*****************************************************************************/
public class RearragePostiveNegativeAlternatively {
    public void rerrange(int[] arrA) {
        int pivot = 0;
        int left = 0;
        int right = arrA.length - 1;
        while (right > left) {
            while (arrA[left] < 0 && left < right)
                left++;
            while (arrA[right] > 0 && left < right)
                right--;
            if (left < right) {

                int temp = arrA[left];
                arrA[left] = arrA[right];
                arrA[right] = temp;
                left++;
                right--;
            }
        }
        // At the point all the negative elements on the left half and
        // positive elements on the right half of the array
        // swap the every alternate element in the left half (negative
        // elements) with the elements in the right (positive elements)
        left = 1;
        int high = 0;
        while (arrA[high] < 0)
            high++;
        right = high;
        while (arrA[left] < 0 && right < arrA.length) {
            int temp = arrA[left];
            arrA[left] = arrA[right];
            arrA[right] = temp;
            left = left + 2;
            right++;
        }
        for (int i = 0; i < arrA.length; i++) {
            System.out.print("  " + arrA[i]);
        }
    }

    public static void main(String[] args) throws java.lang.Exception {
        int[] arrA = { 1, 2, -3, -4, -5, 6, -7, -8, 9, 10, -11, -12, -13, 14 };
        RearragePostiveNegativeAlternatively i = new
        RearragePostiveNegativeAlternatively();
```

```
1429            i.rerrange(arrA);
1430        }
1431    }
1432    /*************************************************************************/
1433    public class MaximumDistance {
1434        public int findMaxDistance(int[] arrA) {
1435            int[] Lmin = new int[arrA.length];
1436            int[] Rmax = new int[arrA.length];
1437            int leftMinIndex = 0;
1438            int leftMinValue = arrA[0];
1439            int rightMaxValue = arrA[arrA.length - 1];
1440            int rightMaxIndex = arrA.length - 1;
1441
1442            // traverse the main array and fill the Lmin array with the index
1443            // position which has the minimum value so far
1444            for (int i = 0; i < arrA.length; i++) {
1445                if (leftMinValue > arrA[i]) {
1446                    leftMinIndex = i;
1447                    leftMinValue = arrA[i];
1448                }
1449                Lmin[i] = leftMinValue;
1450            }
1451            // for(int i=0;i<Lmin.length;i++){
1452            // System.out.print("  " + Lmin[i]);
1453            // }
1454            // System.out.println("");
1455            // traverse the main array backwords and fill the Rmax array with the
1456            // index position which has the maximun value so far
1457            for (int i = arrA.length - 1; i >= 0; i--) {
1458                if (rightMaxValue < arrA[i]) {
1459                    rightMaxIndex = i;
1460                    rightMaxValue = arrA[i];
1461                }
1462                Rmax[i] = rightMaxValue;
1463            }
1464            // for(int i=0;i<Rmax.length;i++){
1465            // System.out.print("  " + Rmax[i]);
1466            // }
1467            System.out.println("");
1468            // Initialize distance_so_far = -1
1469            int distance_so_far = -1;
1470            int i = 0, j = 0;
1471            // Then check if (Rmax[i]-Lmin[i])>distance_so_far then distance_so_far
1472            // = Rmax[i]-Lmin[i]
1473            while (i < arrA.length && j < arrA.length) {
1474                if (Lmin[i] < Rmax[j]) {
1475                    if ((j - i > distance_so_far)) {
1476                        distance_so_far = j - i;
1477
1478                    }
1479                    j++;
1480                } else {
1481                    i++;
1482                }
1483            }
1484            return distance_so_far;
1485        }
1486
1487        public static void main(String args[]) {
1488            int[] arrA = { 12, 3, 1, 5, 6, 4, 10, 9, 8, 0 };
1489            MaximumDistance m = new MaximumDistance();
1490            int x = m.findMaxDistance(arrA);
1491            System.out.println("Max(j-i) where j>i and A[j]>A[i] is : " + x);
1492        }
1493    }
1494    /*************************************************************************/
1495    //Objective is to find the element in an array
1496    ///which occurs more than n/k times
1497    public class NbyKTimes {
```

```
1498
1499        public void findElement(int[] arrA, int k) {
1500            Elements[] emts = new Elements[k - 1];
1501            for (int j = 0; j < emts.length; j++) {
1502                emts[j] = new Elements(0, 0);
1503            }
1504            for (int i = 0; i < arrA.length; i++) {
1505                int index = found(emts, arrA[i]);
1506                if (index >= 0) {
1507                    // means element found in Elements array
1508                    // just increase its count
1509                    emts[index].count++;
1510                } else {
1511                    addtoArray(emts, arrA[i]);
1512                }
1513            }//
1514                // now check if any of the elements in the Elements array appears
1515                // more than n/k times
1516            for (int i = 0; i < emts.length; i++) {
1517                int cnt = 0;
1518                for (int j = 0; j < arrA.length; j++) {
1519                    if (arrA[j] == emts[i].element) {
1520                        cnt++;
1521                    }
1522                }
1523                if (cnt > (arrA.length / k)) {
1524                    System.out.println(emts[i].element + " appears more than n/"
1525                            + k + " times, Actual count is " + cnt);
1526                }
1527            }
1528        }
1529
1530        public void addtoArray(Elements[] emts, int x) {
1531            // check is array is full or not
1532            for (int j = 0; j < emts.length; j++) {
1533                if (emts[j].count == 0) {// find the empty place and add it
1534                    emts[j].element = x;
1535                    return;
1536                }
1537            }
1538            // if we have reached here means array is full
1539            // reduce the counter of every element
1540            for (int j = 0; j < emts.length; j++) {
1541                emts[j].count--;
1542            }
1543        }
1544
1545        // This found function will check whether element already exist or not
1546        // if yes then return its index else return -1
1547        public int found(Elements[] emts, int x) {
1548            for (int j = 0; j < emts.length; j++) {
1549                if (emts[j].element == x) {
1550                    return j;
1551                }
1552            }
1553            return -1;
1554        }
1555
1556        public static void main(String args[]) {
1557            int[] arrA = { 2, 2, 4, 4, 3, 5, 3, 4, 4, 6, 4, 3, 3, 8 };
1558            NbyKTimes n = new NbyKTimes();
1559            n.findElement(arrA, 4);
1560        }
1561    }
1562
1563    class Elements {
1564        int element;
1565        int count;
1566
```

```
1567        public Elements(int element, int count) {
1568            this.element = element;
1569            this.count = count;
1570        }
1571    }
1572
1573    /*************************************************************************/
1574    public class SortedArrayToBST {
1575        public BSTNode convert(int [] arrA, int start, int end){
1576            if(start>end){
1577                return null;
1578            }
1579            int mid = (start + end)/2;
1580            BSTNode root = new BSTNode(arrA[mid]);
1581            root.left = convert(arrA, start, mid-1);
1582            root.right =convert(arrA, mid+1, end);
1583            return root;
1584        }
1585        public void displayTree(BSTNode root){
1586            if(root!=null){
1587                displayTree(root.left);
1588                System.out.print(" " + root.data);
1589                displayTree(root.right);
1590            }
1591        }
1592        public static void main(String args[]){
1593            int [] arrA = {2,3,6,7,8,9,12,15,16,18,20};
1594            SortedArrayToBST s = new SortedArrayToBST();
1595            BSTNode x = s.convert(arrA, 0, arrA.length-1);
1596            System.out.println("Tree Display : ");
1597            s.displayTree(x);
1598        }
1599    }
1600    class BSTNode{
1601        int data;
1602        BSTNode left;
1603        BSTNode right;
1604        public BSTNode(int data){
1605            this.data = data;
1606            left = null;
1607            right = null;
1608        }
1609    }
1610    /*************************************************************************/
1611    public class PrintAllPathIn2DArray {
1612
1613        int rowCount;
1614        int colCount;
1615        int[][] arrA;
1616
1617        public PrintAllPathIn2DArray(int arrA[][]) {
1618            this.arrA = arrA;
1619            rowCount = arrA.length;
1620            colCount = arrA[0].length;
1621        }
1622
1623        public void printAll(int currentRow, int currentColumn, String path) {
1624            if (currentRow == rowCount - 1) {
1625                for (int i = currentColumn; i < colCount; i++) {
1626                    path += "-" + arrA[currentRow][i];
1627                }
1628                System.out.println(path);
1629                return;
1630            }
1631            if (currentColumn == colCount - 1) {
1632                for (int i = currentRow; i <= rowCount - 1; i++) {
1633                    path += "-" + arrA[i][currentColumn];
1634                }
1635                System.out.println(path);
```

```
            return;
        }
        path = path + "-" + arrA[currentRow][currentColumn];
        printAll(currentRow + 1, currentColumn, path);
        printAll(currentRow, currentColumn + 1, path);
    //  printAll(currentRow + 1, currentColumn + 1, path);
    }

    public static void main(String args[]) {
        int[][] a = { { 1, 2, 3 }, { 4, 5, 6 } };
        PrintAllPathIn2DArray p = new PrintAllPathIn2DArray(a);
        p.printAll(0, 0, "");
    }

}
/*****************************************************************************/
public class RearrageArrayPositiveNegative {
    int[] arrA;

    public RearrageArrayPositiveNegative(int[] arrA) {
        this.arrA = arrA;
    }

    public void divideGroups(int low, int high) {
        if (low >= high)
            return;
        int mid = (low + high) / 2;
        divideGroups(low, mid);
        divideGroups(mid + 1 , high);
        merge(low, mid, high);

    }

    public void merge(int low, int mid, int high) {
        int l = low;
        int k = mid + 1;
        while (l <= mid && arrA[l] <= 0)
            l++;
        while (k <= high && arrA[k] <= 0)
            k++;
        reverse(l, mid);
        reverse(mid + 1, k - 1);
        reverse(l, k - 1);
    }

    public void reverse(int x, int y) {
        while (y > x) {
            int temp = arrA[x];
            arrA[x] = arrA[y];
            arrA[y] = temp;
            x++;
            y--;
        }
    }

    public void display() {
        for (int i = 0; i < arrA.length; i++) {
            System.out.print(" " + arrA[i]);
        }
    }

    public static void main(String args[]) {
        int[] a = { 1, -2, -3, -4, 5, -6, 7, -8, 9, -10, -11, -12, 20 };
        RearrageArrayPositiveNegative r = new RearrageArrayPositiveNegative(a);
        System.out.print("Input : ");
        r.display();
        r.divideGroups(0, a.length - 1);
        System.out.println("");
        System.out.print("ReArranged Output : ");
```

```
            r.display();
        }
    }
    /*****************************************************************************/
    public class LongestPrefixSequence {
        private String[] arrA;

        public LongestPrefixSequence(String[] arrA) {
            this.arrA = arrA;
        }

        public String findPrefix() {
            int resultLen = arrA[0].length();
            int curr;
            for (int i = 1; i < arrA.length; i++) {
                curr = 0;
                while (curr < resultLen && curr < arrA[i].length()
                        && arrA[0].charAt(curr) == arrA[i].charAt(curr)) {
                    curr++;
                }
                resultLen = curr;
            }
            return arrA[0].substring(0, resultLen);
        }

        public static void main(String args[]) {
            String x = "Sumit Summation Summit Sum";
            String[] arrA = x.split(" ");
            LongestPrefixSequence lp = new LongestPrefixSequence(arrA);
            System.out.println("Original String : " + x);
            System.out.println("Common Prefix is : " + lp.findPrefix());
        }
    }
    /*****************************************************************************/
    public class Print2DArrayInSpiral {

        public int arrA[][] = { { 1, 2, 3, 4, 5 }, { 18, 19, 20, 21, 6 },
                { 17, 28, 29, 22, 7 }, { 16, 27, 30, 23, 8 },
                { 15, 26, 25, 24, 9 }, { 14, 13, 12, 11, 10 } };

        public int printSpiral(int row_S, int row_E, int col_S, int col_E,
                boolean reverse, boolean rowPrint) {

            if (row_S > row_E && col_S > col_E) {
                return 1;
            }
            if (rowPrint == false) {
                if (reverse == false) {
                    for (int i = col_S; i <= col_E; i++) {
                        System.out.print(" " + arrA[row_S][i]);
                    }
                }
                row_S++;
                rowPrint = true;
                reverse = false;
            }
            if (rowPrint == true) {
                if (reverse == false) {
                    for (int i = row_S; i <= row_E; i++) {
                        System.out.print(" " + arrA[i][col_E]);
                    }
                }
                col_E--;
                rowPrint = false;
                reverse = true;
            }
            if (rowPrint == false) {
                if (reverse == true) {
                    for (int i = col_E; i >= col_S; i--) {
```

```
                    System.out.print(" " + arrA[row_E][i]);
                }
            }
            row_E--;
            rowPrint = true;
            reverse = true;
        }
        if (rowPrint == true) {
            if (reverse == true) {
                for (int i = row_E; i >= row_S; i--) {
                    System.out.print(" " + arrA[i][col_S]);
                }
            }
            col_S++;
            rowPrint = false;
            reverse = false;
        }
        printSpiral(row_S, row_E, col_S, col_E, reverse, rowPrint);
        return 0;
    }

    public static void main(String args[]) {
        Print2DArrayInSpiral p = new Print2DArrayInSpiral();
        p.printSpiral(0, 5, 0, 4, false, false);
    }

}
/*****************************************************************************/
public class MergeSort {

    private int arrSize;
    private int [] arrAux;
    private int [] arrInput;

    public MergeSort(int [] arrInput){
        this.arrInput = arrInput;
        arrSize = arrInput.length;
        arrAux = new int [arrSize];
    }

    public int[] mergeSorting(){
        sort(0,arrSize-1);
        return arrInput;

    }
    public void sort(int low, int high){
        if(low<high){
            int mid = low+((high-low))/2;
            sort(low,mid);
            sort(mid+1,high);
            merge(low, mid, high);
        }

    }
    public void merge(int low, int mid, int high){
        //copy the entire array in the Auxilary array
        for(int i=low;i<=high;i++){
            arrAux[i] = arrInput[i];
        }
        int i = low;
        int j = mid+1;
        int k = low;

        while(i<=mid && j<=high){
            if(arrAux[i]<=arrAux[j]){
                arrInput[k]=arrAux[i];
                i++;
            }
            else{
```

```
                arrInput[k]=arrAux[j];
                j++;
            }
            k++;
        }
        while(i<=mid){
            arrInput[k]=arrAux[i];
            i++;
            k++;
        }
        while(j<=high){
            arrInput[k]=arrAux[j];
            j++;
            k++;
        }
    }

    public void displayArray(int [] b){
        for(int i=0;i<b.length;i++){
            System.out.print(" " + b[i]);
        }
    }

    public static void main(String[] args){
        int [] a = {2,1,6,3,9,4,5,10};
        MergeSort m = new MergeSort(a);
        int [] b = m.mergeSorting();
        m.displayArray(b);

    }

}
/*********************************************************************************/
public class BinarySearch {
    private int [] arrA;
    private int number;

    public BinarySearch(int [] arrA){
        this.arrA = arrA;
    }
    public Boolean Search(int low,int high, int number){
        if(low>high){
            return false;
        }
        int mid = low + ((high - low) / 2);
        if(arrA[mid]==number)return true;
        else if (arrA[mid]>number) return Search(low,mid-1,number);
        else return Search(mid+1,high,number);
    }

    public static void main(String args[]){
        int [] a = {2,5,8,10,14,44,77,78,99};
        int number = 99;
        BinarySearch b = new BinarySearch(a);
        System.out.println("The "+ number + " present in array a ??? :" + b.Search(0,
        a.length-1, number));
        number = 76;
        System.out.println("The "+ number + " present in array a ??? :" + b.Search(0,
        a.length-1, number));

    }

}
/*********************************************************************************/
```