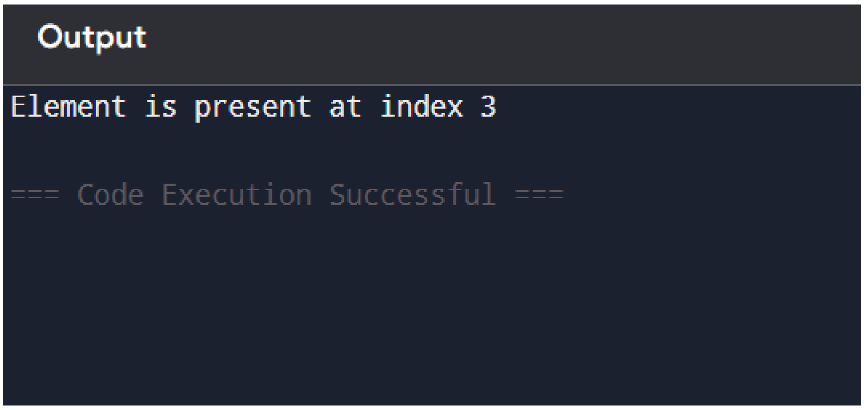
BINARY SEARCH

```
import java.io.*;
class BinarySearch{
         int binarysearch(int arr, int x){
                int low=0, int high=arr.length-1;
                while(low<=high){
                      int mid=low+(high-low)/2;
                      if(arrr[mid]==x){
                             return mid;
                      }
                      if(arr[mid]<x){
                             low=mid+1;
                      }
                      else{
                            high=mid-1;
                     }
               }
               return -1;
        }
  public static void main(String args[])
  {
    BinarySearch ob = new BinarySearch();
    int arr[] = { 2, 3, 4, 10, 40 };
    int n = arr.length;
    int x = 10;
    int result = ob.binarySearch(arr, x);
    if (result == -1)
      System.out.println(
        "Element is not present in array");
    else
```

```
            System.out.println("Element is present at "
                                + "index " + result);
   }
}
```



```
Element is present at index 3

=== Code Execution Successful ===
```

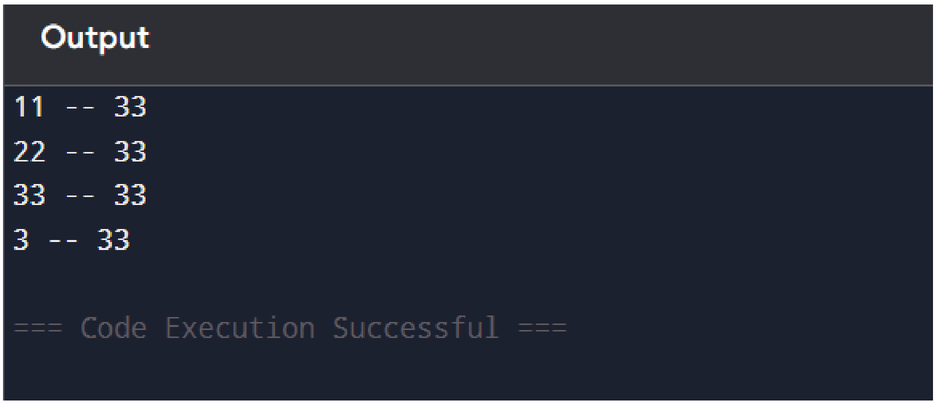
Time complexity: O(logn)

Space complexity: O(1)

Next Greater Element (NGE) for every element in given Array

```
import java.io.*;
class NGE{
            static void nge(int arr[], int n){
                        int i, j;
                        int next=-1;
                        for( i=0;i<n;i++){
                              for(j=i+1 ; j<n; j++){
                                      if(arr[i]<arr[j]){
                                              next=arr[j];
                                      }
                              }
                              System.out.println(arr[i] + " -- " + next);
                        }
```

```
        }
        public static void main(String[] args){
                int arr[]={11, 22, 33, 3};
                int n = arr.length;
                nge(arr, n);
        }
}
```



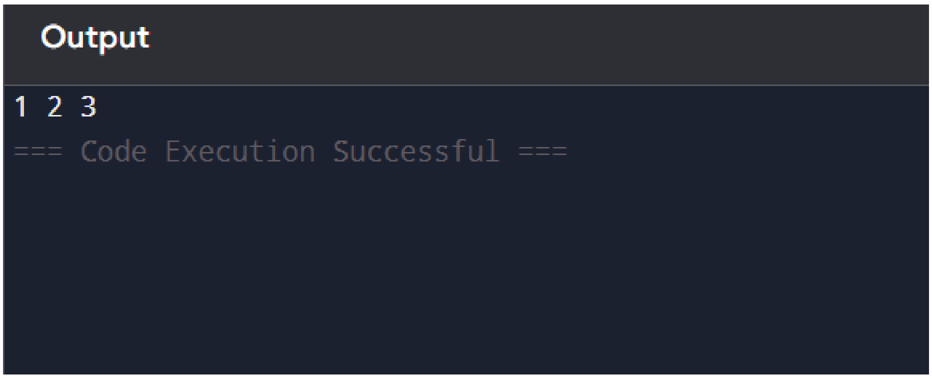
Output

```
11 -- 33
22 -- 33
33 -- 33
3 -- 33

=== Code Execution Successful ===
```

Time Complexity: O(n2)
Auxiliary Space: O(1)

Union of Two Arrays

```
import java.util.HashSet;

import java.util.ArrayList;


class Union{
        static ArrayList<Integer> findUnion(int[] a, int[]  b){
                 HashSet<Integer>  set=new HashSet<>();
                 for (int num:a){
                      set.add(num);
                  }
                for (int num:b){
                      set.add(num);
                  }
                 ArrayList<Integer> result=new ArrayList<>();
for(int i:set){
          result.add(i);
}
return result;
}
public static void main(String[] args) {
    int[] a = {1, 2, 3, 2, 1};
    int[] b = {3, 2, 2, 3, 3, 2};

    ArrayList<Integer> result = findUnion(a, b);

    for (int num : result)
      System.out.print(num + " ");
  }
}
```



Output

```
1 2 3
=== Code Execution Successful ===
```

Time Complexity: O(n + m)

Space complexity: O(n + m)

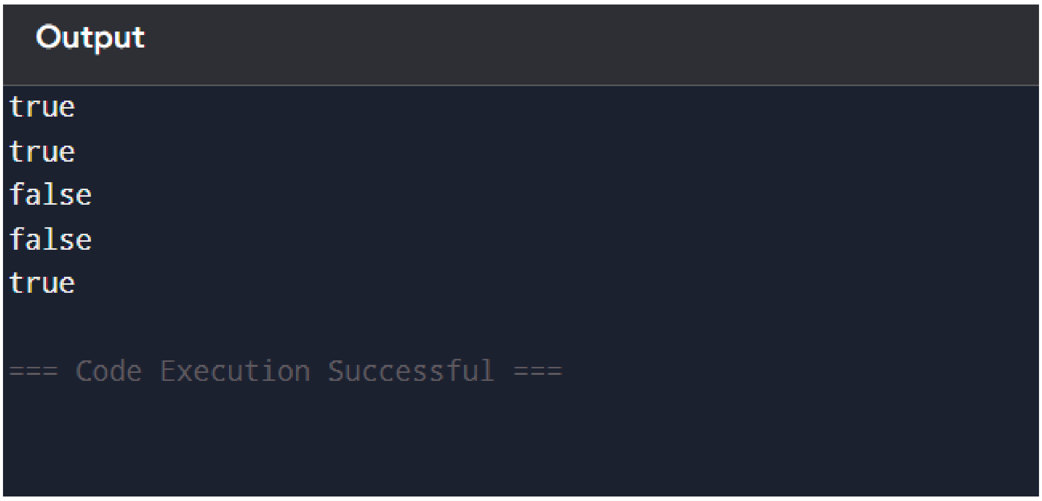
VALID PARENTHESES

```
import java.util.Stack;
```

```
class Main {
    public boolean isValid(String s) {
        Stack<Character> st = new Stack<>();

        for (char c : s.toCharArray()) {
            if (c == '[') {
                st.push(']');
            } else if (c == '{') {
                st.push('}');
            } else if (c == '(') {
                st.push(')');
            } else if (st.isEmpty() || st.pop() != c) {
                return false;
            }
        }
        return st.isEmpty();
    }

    public static void main(String[] args) {
        Main sol = new Main();
        System.out.println(sol.isValid("()"));
        System.out.println(sol.isValid("()[]{}"));
        System.out.println(sol.isValid("(]"));
        System.out.println(sol.isValid("([)]"));
        System.out.println(sol.isValid("{[]}"));
    }
}
```

TIME COMPLEXITY:O(n)

SPACE COMPLEXITY:O(n)

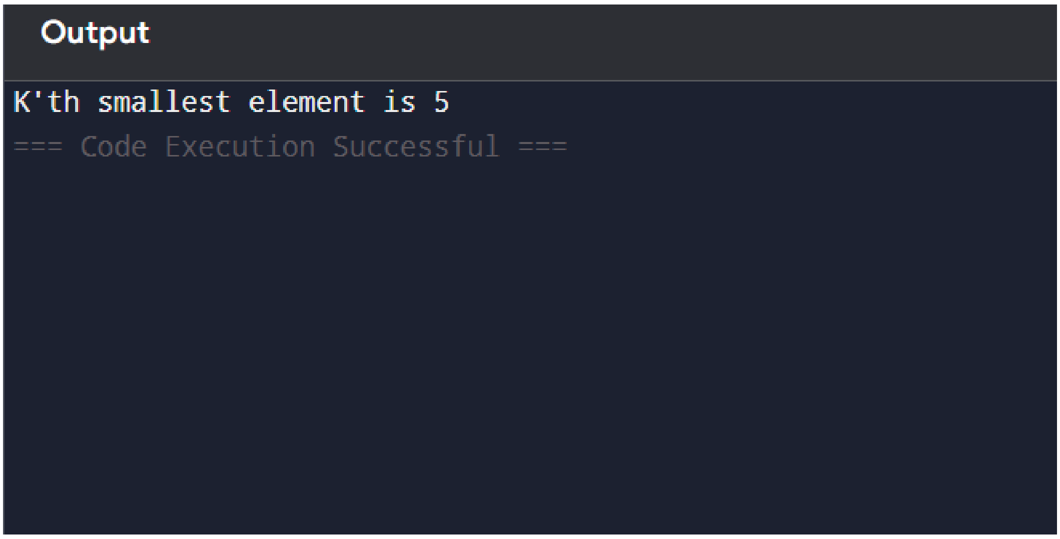


```
true
true
false
false
true

=== Code Execution Successful ===
```

K’th Smallest Element in Unsorted Array

```
import java.util.Arrays;

import java.util.Collections;

class Small{

            public static int kthSmallest(Integer[] arr, int k){

                         Arrays.sort(arr);

                         return arr[k-1];

            }

            public static void main(String[] args)

             {

                        Integer arr[] = new Integer[] { 12, 3, 5, 7, 19 };

                        int k = 2;

                        System.out.print("K'th smallest element is "+ kthSmallest(arr, k));

  }

}
```

Time Complexity: O(N log N)
Auxiliary Space: O(1)



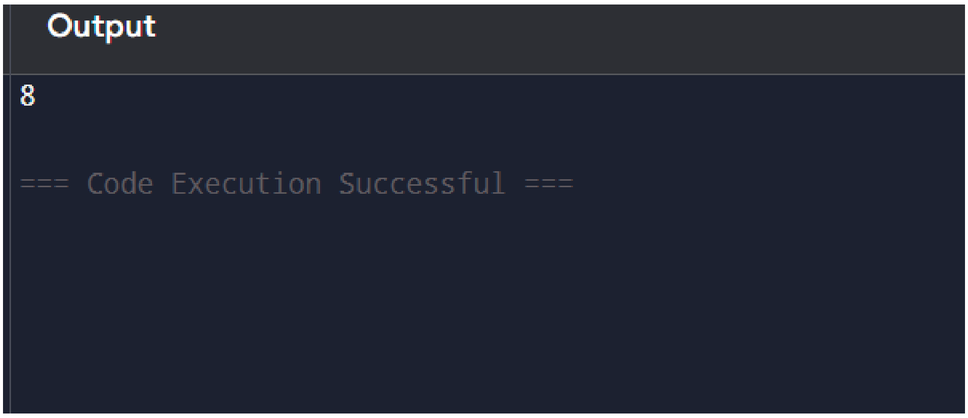
```
K'th smallest element is 5
=== Code Execution Successful ===
```

Minimize the maximum difference between the heights

```
import java.util.Arrays;
class minidiffheight{
    static int getMinDiff(int[] arr, int k){
                 int n=arr.length;
                Arrays.sort(arr);
                int result=arr[n-1]-arr[0];
                for(int i=0; i<n; i++){
                            if(arr[i]-k < 0){
                                     continue;
                            }
                            int minH=Math.min(arr[0]+k , arr[i]-k);
                            int maxH=Math.max(arr[i-1]+k , arr[n-1]-k);
                            result=Math.min(result, maxH-minH);
                 }
                return result;
      }
 public static void main(String[] args) {
       int k = 6;
       int[] arr = {12, 6, 4, 15, 17, 10};
```

```
        int ans = getMinDiff(arr, k);

        System.out.println(ans);

    }

}
```

Time Complexity: O(nlogn)
Auxiliary Space: O(1)

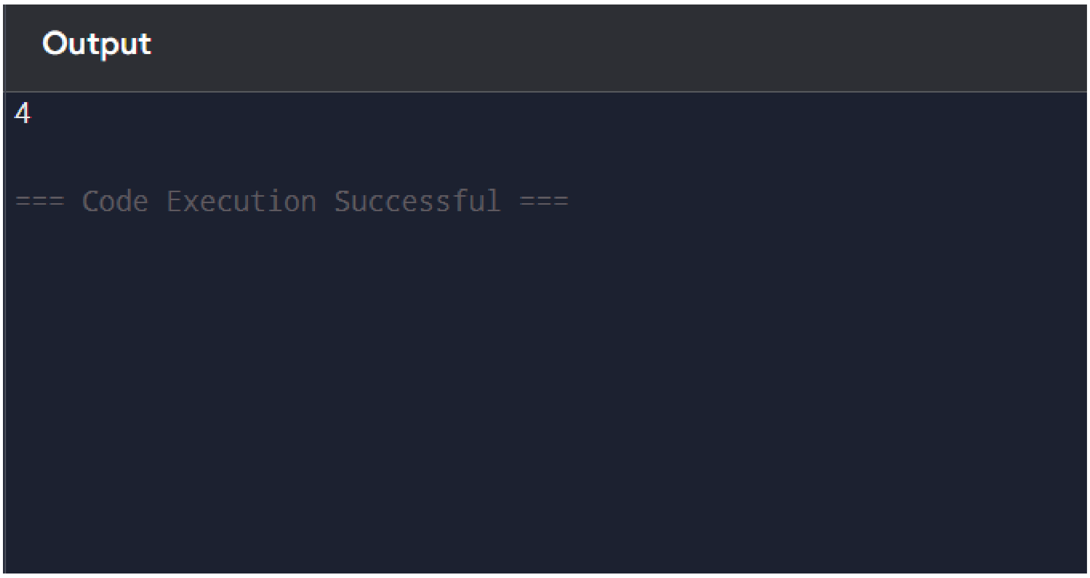


Output

```
8

=== Code Execution Successful ===
```

Equilibrium index of an array

```
import java.util.*;


public class Main {

    public static int equilibriumPoint(long[] arr)

    {

        int n = arr.length;

        long leftsum, rightsum;

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

            leftsum = 0;

            for (int j = 0; j < i; j++)

                leftsum += arr[j];
            rightsum = 0;
            for (int j = i + 1; j < n; j++)
                rightsum += arr[j];
            if (leftsum == rightsum)
                return i + 1;
        }
        return -1;
    }

    public static void main(String[] args)
    {
        long[] arr = { -7, 1, 5, 2, -4, 3, 0 };
        System.out.println(equilibriumPoint(arr));
    }
}
```



```
4

=== Code Execution Successful ===
```

Time Complexity: O(N^2)
Auxiliary Space: O(1)