1.K’th Smallest Element in Unsorted Array:

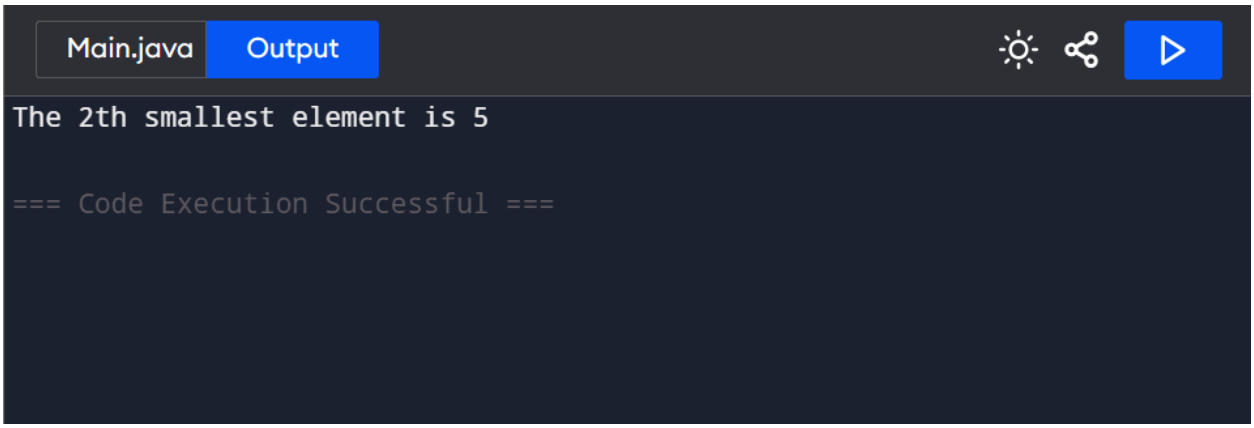
Code:

```
import java.util.Arrays;


public class Main {
   static int kthSmallest(int[] arr, int n, int k)
   {
      int max_element = arr[0];
      for (int i = 1; i < n; i++) {
         if (arr[i] > max_element) {
            max_element = arr[i];
         }
      }
      int[] freq = new int[max_element + 1];
      Arrays.fill(freq, 0);
      for (int i = 0; i < n; i++) {
         freq[arr[i]]++;
      }
      int count = 0;
      for (int i = 0; i <= max_element; i++) {
         if (freq[i] != 0) {
            count += freq[i];
            if (count >= k) {
               return i;
            }
         }
      }
      return -1;
   }
```

```
    public static void main(String[] args)
    {
        int[] arr = { 12, 3, 5, 7, 19 };
        int n = arr.length;
        int k = 2;
        System.out.println("The " + k
                           + "th smallest element is "
                           + kthSmallest(arr, n, k));
    }
}
```

OUTPUT:

The 2th smallest element is 5.



Time Complexity: O(N+max_element)

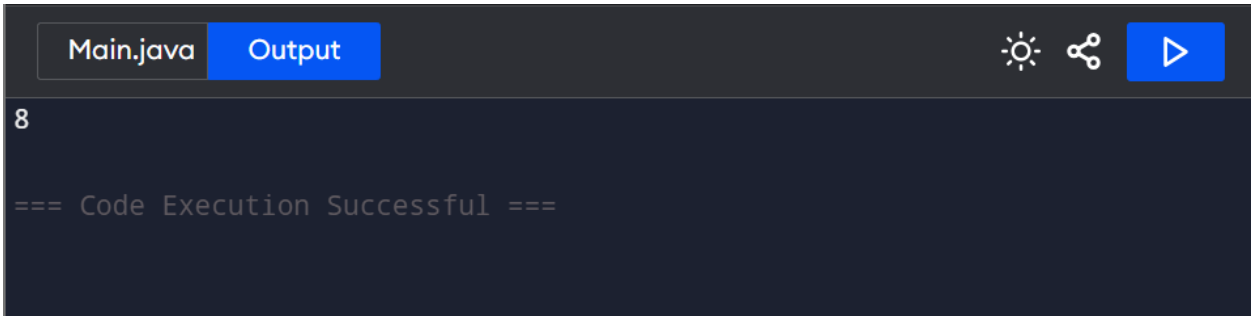
Space Complexity: O(max_element)

2. Minimize the Height:

```
import java.util.Arrays;
class Main {
   static int getMinDiff(int[] arr, int k) {
      int n = arr.length;
      Arrays.sort(arr);
      int res = arr[n - 1] - arr[0];
      for (int i = 1; i < arr.length; 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);
         res = Math.min(res, maxH - minH);
      }
      return res;
   }
   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);
   }
}
```

Output:

8



Time Complexity: O(nlogn)

Auxiliary Space: O(1)

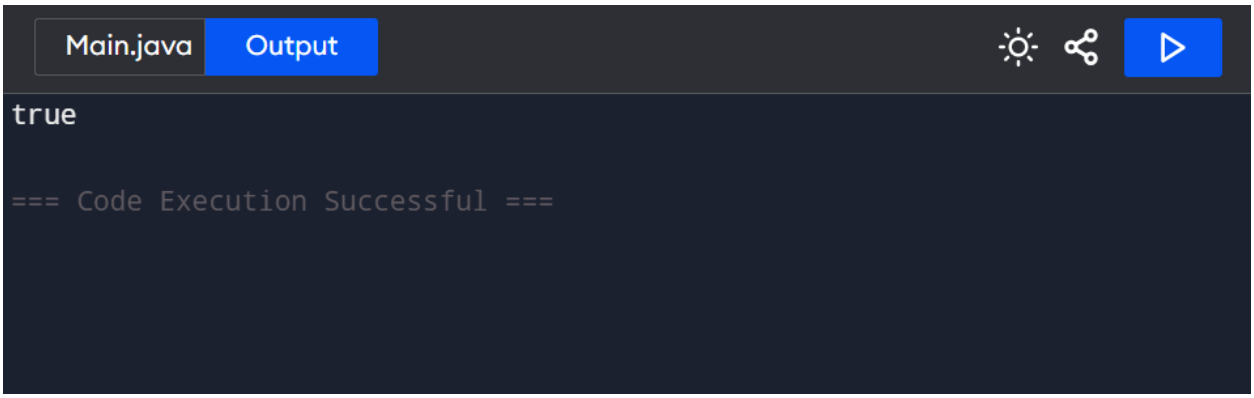
3. Valid Parentheses in an Expression

```
#include <bits/stdc++.h>
using namespace std;

bool checkMatch(char c1, char c2){
    if (c1 == '(' && c2 == ')') return true;
    if (c1 == '[' && c2 == ']') return true;
    if (c1 == '{' && c2 == '}') return true;
    return false;
}

bool ispar(string s){
```

```
    int top = -1;
    for (int i = 0; i < s.length(); ++i){
            if (top < 0 || !checkMatch(s[top], s[i])){
            ++top;
            s[top] = s[i];
        }
        else{
                --top;
        }
    }
    return top == -1;
}
int main(){
    string s = "{()}[]";
    cout << (ispar(s) ? "true" : "false") << endl;
    return 0;
}
```

Output:

True



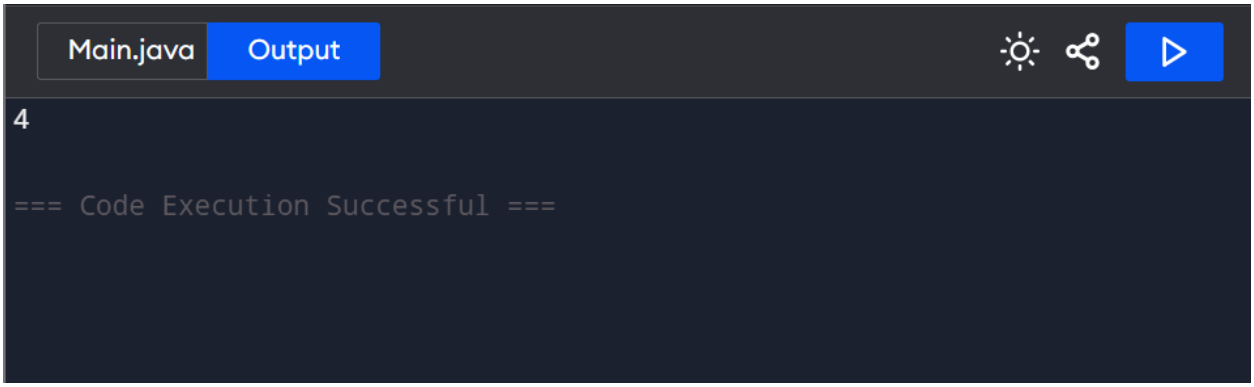
Time Complexity: O(n)

Auxiliary Space: O(n)

4. Equilibrium index of an array

```
import java.util.List;
class Solution {
   public static int equilibriumPoint(long arr[])
   {
      int n = arr.length;
      int left = 0, pivot = 0, right = 0;
      for (int i = 1; i < n; i++) {
         right += arr[i];
      }
      while (pivot < n - 1 && right != left) {
         pivot++;
         right -= arr[pivot];
         left += arr[pivot - 1];
      }
      return (left == right) ? pivot + 1 : -1;
   }
   public static void main(String[] args)
   {
      long[] arr = { 1, 7, 3, 6, 5, 6 };
      int result = equilibriumPoint(arr);
      System.out.println(result);
   }
}
```

Output:



Time Complexity: O(N)

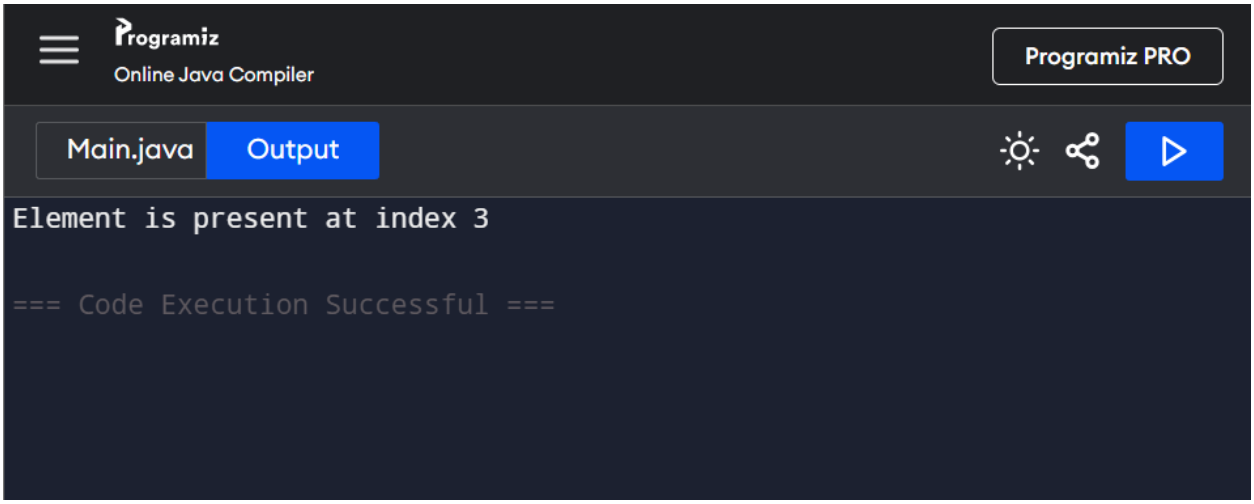
Auxiliary Space: O(1)

5. Binary Search:

```
class BinarySearch {
    int binarySearch(int arr[], int low, int high, int x)
    {
        if (high >= low) {
            int mid = low + (high - low) / 2;
            if (arr[mid] == x)
                return mid;
            if (arr[mid] > x)
                return binarySearch(arr, low, mid - 1, x);
            return binarySearch(arr, mid + 1, high, x);
        }
        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, 0, n - 1, x);
        if (result == -1)
            System.out.println(
                "Element is not present in array");
        else
            System.out.println(
                "Element is present at index " + result);
    }
}
```

OUTPUT:



Time Complexity:

Best Case: O(1)

Average Case: O(log N)

Worst Case: O(log N)

Auxiliary Space: O(1)

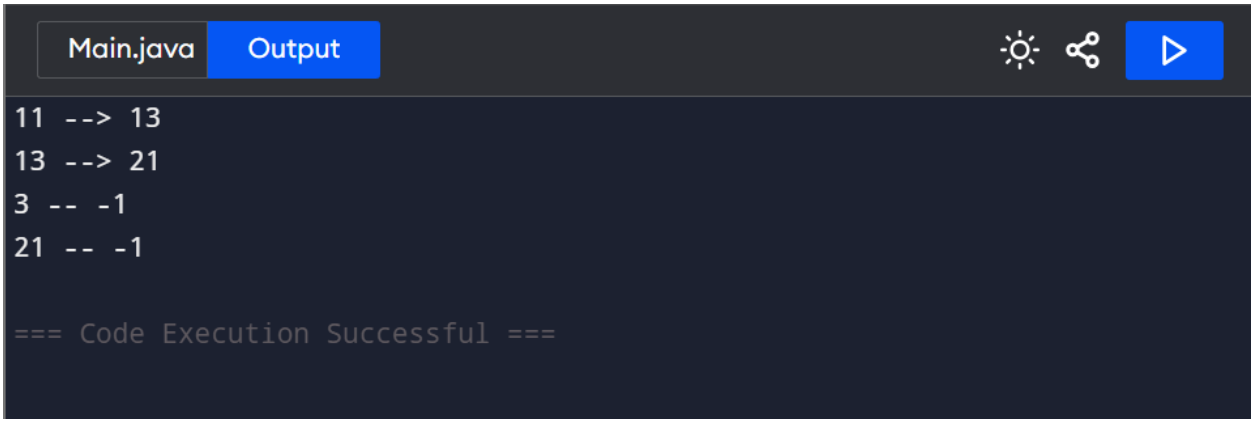
6. Next greater element:

```
public class NGE {
    static class stack {
        int top;
        int items[] = new int[100];
        void push(int x)
        {
            if (top == 99) {
                System.out.println("Stack full");
            }
            else {
                items[++top] = x;
            }
        }

        int pop()
        {
            if (top == -1) {
                System.out.println("Underflow error");
                return -1;
            }
            else {
                int element = items[top];
                top--;
                return element;
            }
        }

        boolean isEmpty()
        {
```

```
        return (top == -1) ? true : false;
    }
}
static void printNGE(int arr[], int n)
{
    int i = 0;
    stack s = new stack();
    s.top = -1;
    int element, next;
    s.push(arr[0]);
    for (i = 1; i < n; i++) {
        next = arr[i];
        if (s.isEmpty() == false) {
            element = s.pop();
            while (element < next) {
                System.out.println(element + " --> "
                                   + next);
                if (s.isEmpty() == true)
                    break;
                element = s.pop();
            }
            if (element > next)
                s.push(element);
        }
        s.push(next);
    }

    while (s.isEmpty() == false) {
        element = s.pop();
```

```
            next = -1;
            System.out.println(element + " -- " + next);
        }
    }
    public static void main(String[] args)
    {
        int arr[] = { 11, 13, 21, 3 };
        int n = arr.length;
        printNGE(arr, n);
    }
}
```

OUTPUT:



Time Complexity: O(n)

Auxiliary Space: O(n)

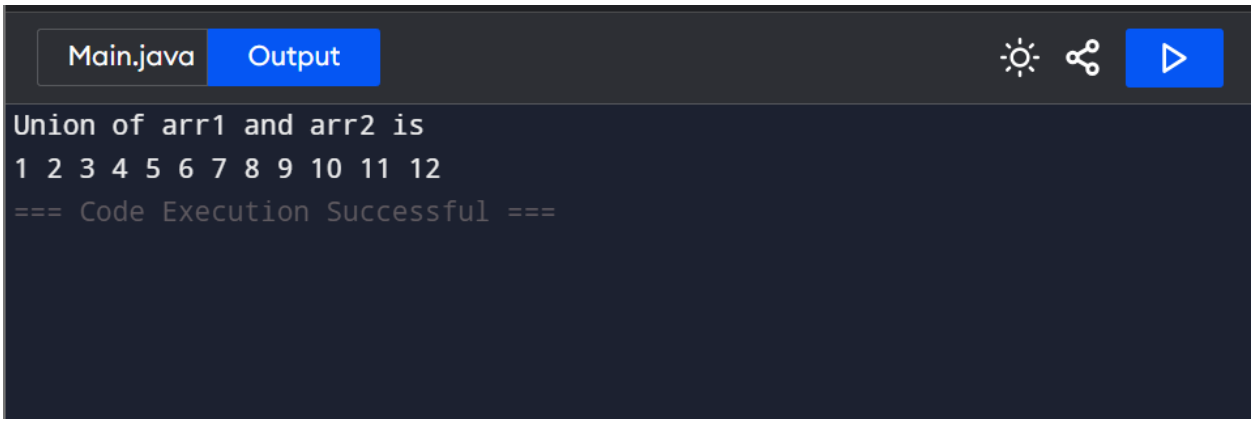
7. Union of 2 arrays with duplicate elements:

```
import java.util.*;
```

```
class Main{
static ArrayList<Integer> FindUnion(int arr1[], int arr2[], int n, int m) {
  int i = 0, j = 0;
 ArrayList<Integer > Union=new ArrayList<>();
 while (i < n && j < m) {
  if (arr1[i] <= arr2[j])
    if (Union.size() == 0 || Union.get(Union.size()-1) != arr1[i])
      Union.add(arr1[i]);
    i++;
  } else
  {
    if (Union.size() == 0 || Union.get(Union.size()-1) != arr2[j])
      Union.add(arr2[j]);
    j++;
  }
 }
 while (i < n)  {
  if (Union.get(Union.size()-1) != arr1[i])
    Union.add(arr1[i]);
  i++;
 }
 while (j < m)  {
  if (Union.get(Union.size()-1) != arr2[j])
    Union.add(arr2[j]);
  j++;
 }
 return Union;
}
public static void main(String args[]) {
```

```
  int n = 10, m = 7;
  int arr1[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
  int arr2[] = {2, 3, 4, 4, 5, 11, 12};
  ArrayList<Integer> Union = FindUnion(arr1, arr2, n, m);
  System.out.println("Union of arr1 and arr2 is ");
  for (int val: Union)
    System.out.print(val+" ");
}
}
```

OUTPUT:



Time Complexity: O(m+n)

Space Complexity : O(m+n)