

### Day-4 Practice SUM

**Name:** *Rishi Aathithya CN*

**Reg no:**22IT086

#### **1.Kth Smallest element**

**Solution:**

```
import java.util.PriorityQueue;

public class KthSmallest {

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

        PriorityQueue<Integer> minheap=new PriorityQueue<>();

        for(int num:arr) {

            minheap.add(num);

        }

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

            minheap.poll();

        }

        return minheap.poll();

    }

    public static void main(String[] args) {

        int[] arr = {7, 10, 4, 3, 20, 15};

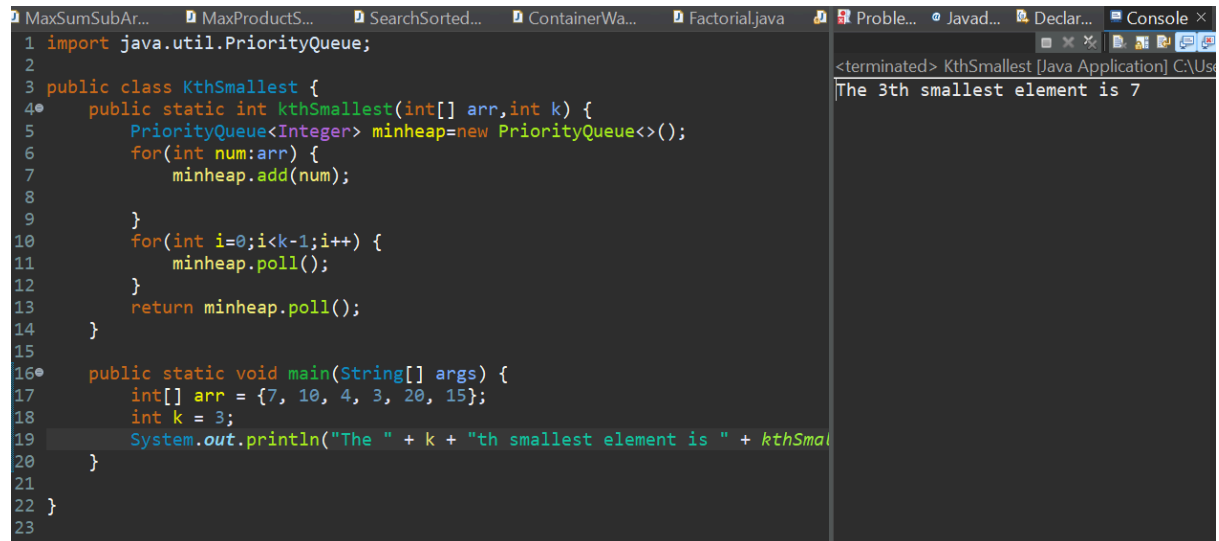
        int k = 3;

        System.out.println("The " + k + "th smallest element is " + kthSmallest(arr, k));

    }

}
```

Output:



```
1 import java.util.PriorityQueue;
2
3 public class KthSmallest {
4     public static int kthSmallest(int[] arr, int k) {
5         PriorityQueue<Integer> minheap = new PriorityQueue<>();
6         for (int num : arr) {
7             minheap.add(num);
8         }
9         for (int i = 0; i < k - 1; i++) {
10             minheap.poll();
11         }
12         return minheap.poll();
13     }
14 }
15
16 public static void main(String[] args) {
17     int[] arr = {7, 10, 4, 3, 20, 15};
18     int k = 3;
19     System.out.println("The " + k + "th smallest element is " + kthSmallest(arr, k));
20 }
21
22 }
```

Time complexity:  $O(n \log n)$

Space complexity:  $O(n)$

## 2. Minimize the height II

Solution:

```
import java.util.Arrays;

public class MinHeightDifference {

    public static int getMinDiff(int[] arr, int k) {

        int n = arr.length;

        if (n == 1) {
            return 0;
        }

        Arrays.sort(arr);

        int mindiff = arr[n - 1] - arr[0];

        for (int i = 1; i < n; i++) {
            if (arr[i] - k < 0) continue;

            int min = Math.min(arr[0] + k, arr[i] - k);

            int max = Math.max(arr[i - 1] + k, arr[n - 1] - k);

            mindiff = Math.min(mindiff, max - min);
        }
    }
}
```

```

    }

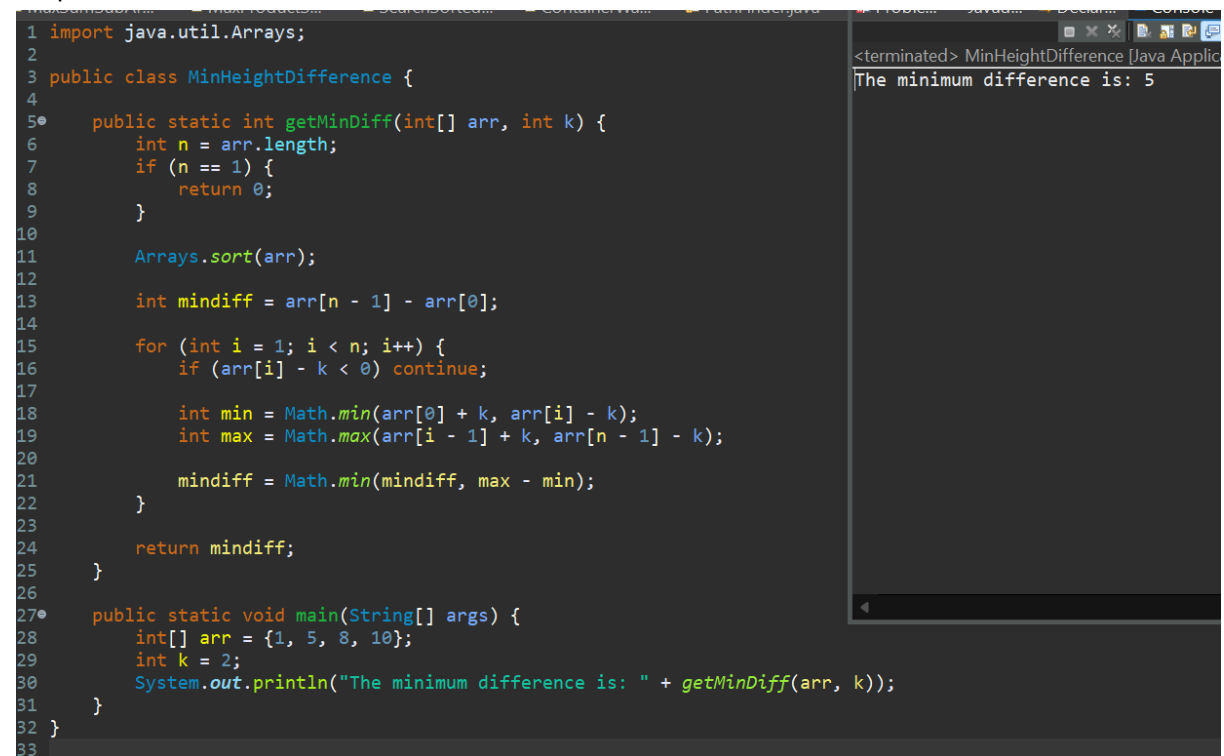
    return mindiff;
}

public static void main(String[] args) {
    int[] arr = {1, 5, 8, 10};
    int k = 2;

    System.out.println("The minimum difference is: " + getMinDiff(arr, k));
}
}

```

Output:



```

1 import java.util.Arrays;
2
3 public class MinHeightDifference {
4
5     public static int getMinDiff(int[] arr, int k) {
6         int n = arr.length;
7         if (n == 1) {
8             return 0;
9         }
10
11         Arrays.sort(arr);
12
13         int mindiff = arr[n - 1] - arr[0];
14
15         for (int i = 1; i < n; i++) {
16             if (arr[i] - k < 0) continue;
17
18             int min = Math.min(arr[0] + k, arr[i] - k);
19             int max = Math.max(arr[i - 1] + k, arr[n - 1] - k);
20
21             mindiff = Math.min(mindiff, max - min);
22         }
23
24         return mindiff;
25     }
26
27     public static void main(String[] args) {
28         int[] arr = {1, 5, 8, 10};
29         int k = 2;
30         System.out.println("The minimum difference is: " + getMinDiff(arr, k));
31     }
32 }
33

```

The console output shows: <terminated> MinHeightDifference [Java Applic  
The minimum difference is: 5

Time complexity:  $O(n \log n)$

Space complexity:  $O(n)$

### 3. Parenthesis Checker

Solution:

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

```

class ParanthesisCheck{

    static boolean isParenthesisBalanced(String s) {

        Stack<Character> st=new Stack<>();

        for(char it:s.toCharArray()){

            if(it=='(' || it=='[' || it=='{'){

                st.push(it);

            }

            else{

                if(st.isEmpty()){

                    return false;

                }

                char ch=st.pop();

                if(ch=='(' && it==')' ||

                ch=='[' && it==']' ||

                ch=='{' && it=='}'){

                    continue;

                }

                return false;

            }

        }

        return st.isEmpty();

    }

    public static void main (String args[]) {

        String s="(){}[]";

        System.out.println(isParenthesisBalanced(s));

    }

}

```

Output:

```
1 import java.util.Stack;
2
3 class ParanthesisCheck{
4
5     static boolean isParenthesisBalanced(String s) {
6         Stack<Character> st=new Stack<>();
7         for(char it:s.toCharArray()){
8             if(it=='(' || it=='[' || it=='{'){
9                 st.push(it);
10            }
11            else{
12                if(st.isEmpty()){
13                    return false;
14                }
15                char ch=st.pop();
16                if(ch=='(' && it==')' ||
17                   ch=='[' && it==']' ||
18                   ch=='{' && it=='}'){
19                    continue;
20                }
21                return false;
22            }
23        }
24        return st.isEmpty();
25    }
26
27    public static void main (String args[]) {
28        String s="(){}[]";
29        System.out.println(isParenthesisBalanced(s));
30    }
31 }
```

Time complexity:  $O(n)$

Space complexity:  $O(n)$

#### 4.Equilibrium Point

**Solution:**

```
class EquilibriumPoint {

    public static int equilibriumPoint(int arr[]) {

        int n=arr.length;

        if(n==1){

            return 1;

        }

        int totalsum=0;

        for(int num:arr){

            totalsum+=num;
```

```

    }

    int left=0;

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

        int right=totalsum-left-arr[i];

        if(left==right){

            return i+1;

        }

        left+=arr[i];

    }

    return -1;

}

public static void main(String args[]) {

    int[] arr= {1,3,5,2,2};

    System.out.println("The equilibrium point is:"+equilibriumPoint(arr));

}

}

```

Output:

```

1
2 class EquilibriumPoint {
3
4     public static int equilibriumPoint(int arr[]) {
5         int n=arr.length;
6         if(n==1){
7             return 1;
8         }
9         int totalsum=0;
10        for(int num:arr){
11            totalsum+=num;
12        }
13        int left=0;
14        for(int i=0;i<n;i++){
15            int right=totalsum-left-arr[i];
16            if(left==right){
17                return i+1;
18            }
19            left+=arr[i];
20        }
21        return -1;
22    }
23
24 }
25
26 public static void main(String args[]) {
27     int[] arr= {1,3,5,2,2};
28     System.out.println("The equilibrium point is:"+equilibriumPoint(arr));
29 }

```

<terminated> EquilibriumPoint [Java Application]  
The equilibrium point is:3

Time complexity:  $O(n)$

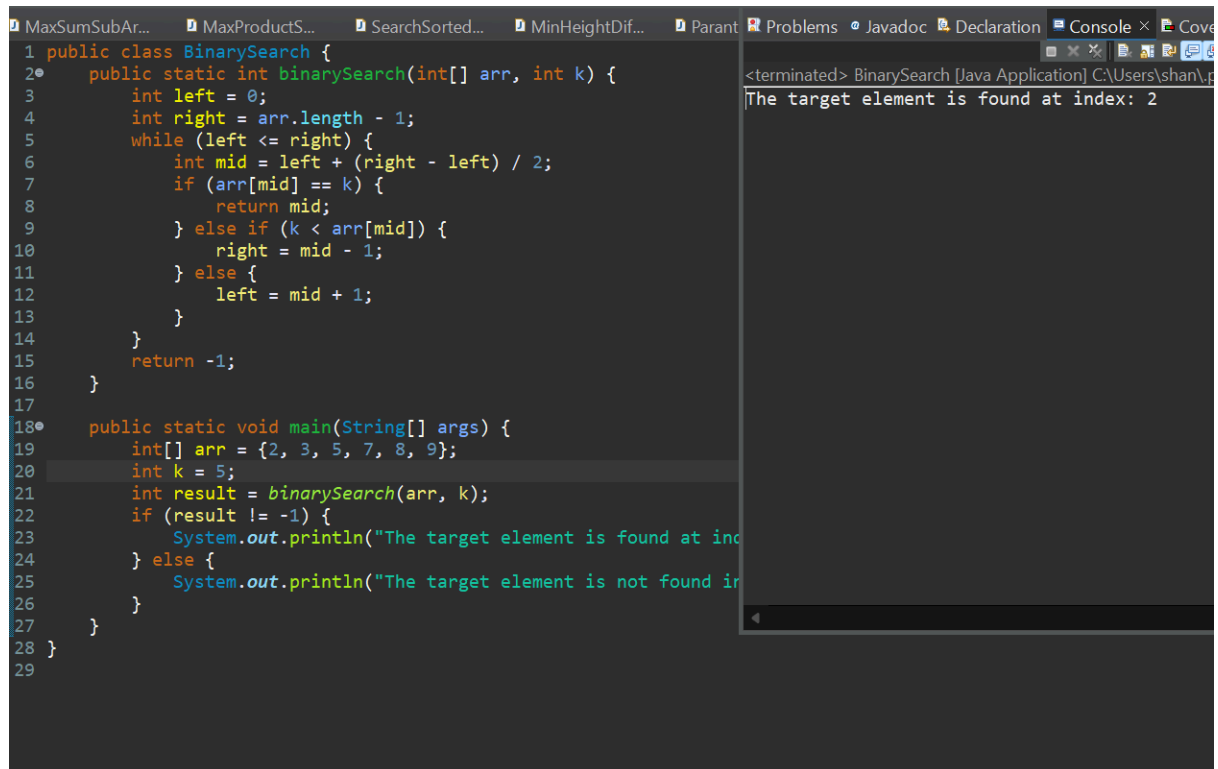
Space complexity:  $O(1)$

## 5.Binary Search

Solution:

```
public class BinarySearch {  
    public static int binarySearch(int[] arr, int k) {  
        int left = 0;  
        int right = arr.length - 1;  
        while (left <= right) {  
            int mid = left + (right - left) / 2;  
            if (arr[mid] == k) {  
                return mid;  
            } else if (k < arr[mid]) {  
                right = mid - 1;  
            } else {  
                left = mid + 1;  
            }  
        }  
        return -1;  
    }  
  
    public static void main(String[] args) {  
        int[] arr = {2, 3, 5, 7, 8, 9};  
        int k = 5;  
        int result = binarySearch(arr, k);  
        if (result != -1) {  
            System.out.println("The target element is found at index: " + result);  
        } else {  
            System.out.println("The target element is not found in the array.");  
        }  
    }  
}
```

Output:

A screenshot of an IDE window with a dark theme. The main editor shows a Java file named 'BinarySearch.java'. The code implements a binary search algorithm. It has a 'binarySearch' method that takes an array and a target value, and a 'main' method that tests it with the array {2, 3, 5, 7, 8, 9} and target 5. The console on the right shows the output: 'The target element is found at index: 2'.

```
1 public class BinarySearch {
2     public static int binarySearch(int[] arr, int k) {
3         int left = 0;
4         int right = arr.length - 1;
5         while (left <= right) {
6             int mid = left + (right - left) / 2;
7             if (arr[mid] == k) {
8                 return mid;
9             } else if (k < arr[mid]) {
10                right = mid - 1;
11            } else {
12                left = mid + 1;
13            }
14        }
15        return -1;
16    }
17
18    public static void main(String[] args) {
19        int[] arr = {2, 3, 5, 7, 8, 9};
20        int k = 5;
21        int result = binarySearch(arr, k);
22        if (result != -1) {
23            System.out.println("The target element is found at index: " + result);
24        } else {
25            System.out.println("The target element is not found in the array");
26        }
27    }
28 }
29
```

<terminated> BinarySearch [Java Application] C:\Users\shan\p...  
The target element is found at index: 2

Time complexity:  $O(\log n)$

Space complexity:  $O(1)$

## 6.Next Greater Element

Solution:

```
import java.util.Scanner;
```

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

```
public class NextGreaterElement {

    public static void nextGreater(int[] arr) {

        Stack<Integer> stack=new Stack<>();

        int[] nge=new int[arr.length];

        for(int i=arr.length-1;i>=0;i--) {

            while (!stack.isEmpty() && stack.peek() <= arr[i]) {

                stack.pop();

            }

            nge[i] = stack.isEmpty() ? -1 : stack.peek();

        }

    }

}
```



```

        stack.push(arr[i]);
    }
    for (int i = 0; i < arr.length; i++) {
        System.out.println(arr[i] + " --> " + nge[i]);
    }
}

```

```

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.println("Enter the elements of the array separated by spaces:");
    String input = scanner.nextLine();
    String[] elements = input.split(" ");
    int[] arr = new int[elements.length];
    for (int i = 0; i < elements.length; i++) {
        arr[i] = Integer.parseInt(elements[i]);
    }
    System.out.println("Next Greater Elements:");
    nextGreater(arr);

}
}

```

Output:

```
1 import java.util.Scanner;
2 import java.util.Stack;
3
4 public class NextGreaterElement {
5     public static void nextGreater(int[] arr) {
6         Stack<Integer> stack=new Stack<>();
7         int[] nge=new int[arr.length];
8
9         for(int i=arr.length-1;i>=0;i--) {
10             while (!stack.isEmpty() && stack.peek() <= arr[i]) {
11                 stack.pop();
12             }
13             nge[i] = stack.isEmpty() ? -1 : stack.peek();
14             stack.push(arr[i]);
15         }
16         for (int i = 0; i < arr.length; i++) {
17             System.out.println(arr[i] + " --> " + nge[i]);
18         }
19     }
20
21
22
23 public static void main(String[] args) {
24     Scanner scanner = new Scanner(System.in);
25     System.out.println("Enter the elements of the array separated by space");
26     String input = scanner.nextLine();
27     String[] elements = input.split(" ");
28     int[] arr = new int[elements.length];
29     for (int i = 0; i < elements.length; i++) {
30         arr[i] = Integer.parseInt(elements[i]);
31     }
32     System.out.println("Next Greater Elements:");
33     nextGreater(arr);
34 }
35 }
```

<terminated> NextGreaterElement [Java Application] C:\User...  
Enter the elements of the array separated by space  
4 5 2 25  
Next Greater Elements:  
4 --> 5  
5 --> 25  
2 --> 25  
25 --> -1

Time complexity:  $O(n)$

Space complexity:  $O(n)$

## 7.Union of Two Array with Duplicate

**Solution:**

```
import java.util.HashSet;
```

```
public class UnionTwoArray {

    public static int unionTwoArray(int[] a,int[] b) {

        HashSet<Integer> hash=new HashSet<>();

        for(int num:a) {

            hash.add(num);

        }

        for(int num:b) {

            hash.add(num);

        }

        return hash.size();

    }

}
```

```

public static void main(String[] args) {

    int[] a= {1,2,3,4,5};

    int[] b= {1,2,3,6};

    System.out.println("The Union of two Array with duplicate is:"+unionTwoArray(a,b));

}

}

```

Output:

The screenshot shows an IDE with a Java file named `UnionTwoArray.java`. The code defines a class `UnionTwoArray` with two methods: `unionTwoArray` and `main`. The `unionTwoArray` method uses a `HashSet` to find the union of two arrays. The `main` method initializes two arrays, `a` and `b`, and prints the result of `unionTwoArray(a, b)`. The output window on the right shows the message: `The Union of two Array with duplicate is:6`.

```

import java.util.HashSet;

public class UnionTwoArray {
    public static int unionTwoArray(int[] a,int[] b) {
        HashSet<Integer> hash=new HashSet<>();
        for(int num:a) {
            hash.add(num);
        }
        for(int num:b) {
            hash.add(num);
        }
        return hash.size();
    }

    public static void main(String[] args) {
        int[] a= {1,2,3,4,5};
        int[] b= {1,2,3,6};
        System.out.println("The Union of two Array with duplicate

    }

}

```

<terminated> UnionTwoArray [Java Application] C:\Users\shan\p  
The Union of two Array with duplicate is:6

Time complexity:  $O(n+m)$

Space complexity:  $O(n+m)$