

ASSIGNMENT -5 (ADVANCED PROGRAMMING)

Prince Sharma – 22BCS14846

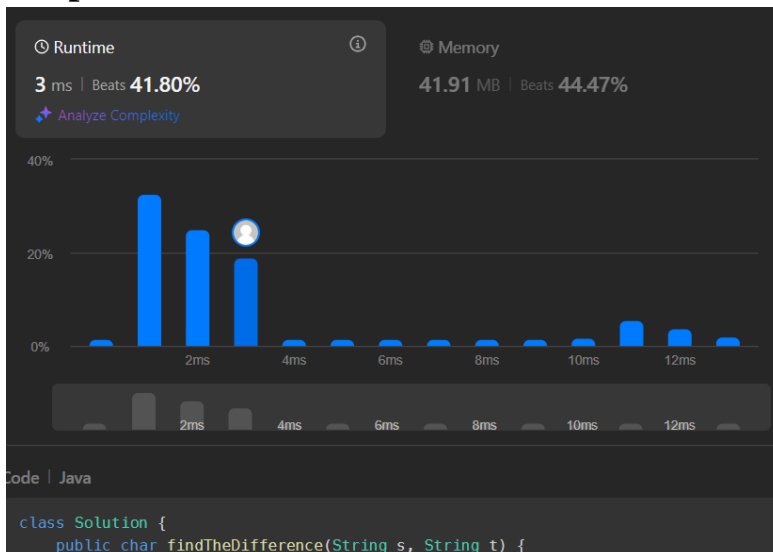
Section-22BCS_IOT-605

1. Problem 1: Find the Difference.

2. Implementation/Code:

```
class Solution {  
    public char findTheDifference(String s, String t) {  
        int ssum = 0;  
        int tsum = 0;  
        for(int i=0;i<s.length();i++)  
        { ssum = ssum + (int)s.charAt(i); }  
        for(int i=0;i<t.length();i++) {  
            tsum = tsum + (int)t.charAt(i); }  
        int value = tsum - ssum;  
        return (char)value;  
    }  
}
```

3. Output:

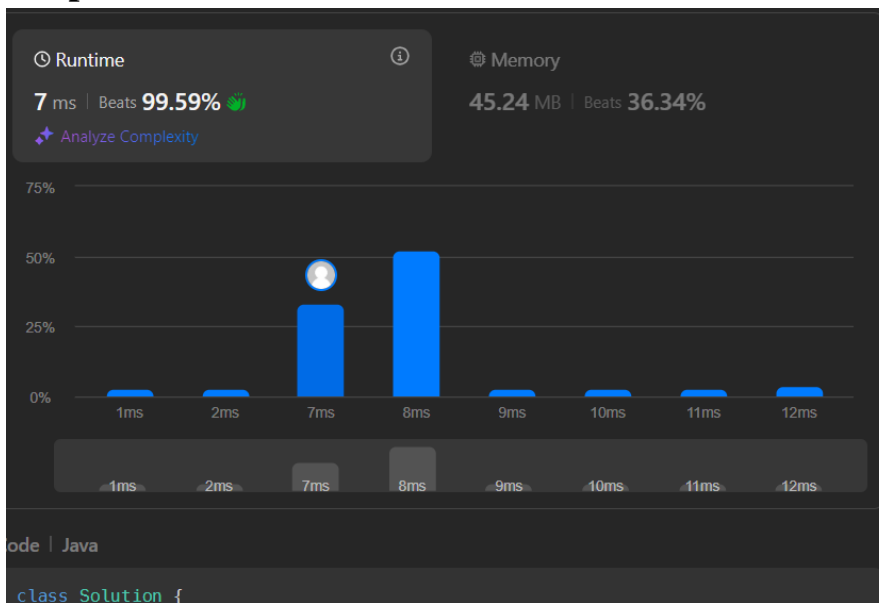


1. Problem 2: Largest Perimeter Triangle

2. Implementation/Code:

```
class Solution {  
    public int largestPerimeter(int[] nums) {  
        Arrays.sort(nums);  
        for(int i = nums.length-1; i>1; i--){  
            if(nums[i] < nums[i-1] + nums[i-2])  
                return nums[i] + nums[i-1]+ nums[i-2];  
        }  
        return 0;  
    }  
}
```

3. Output:

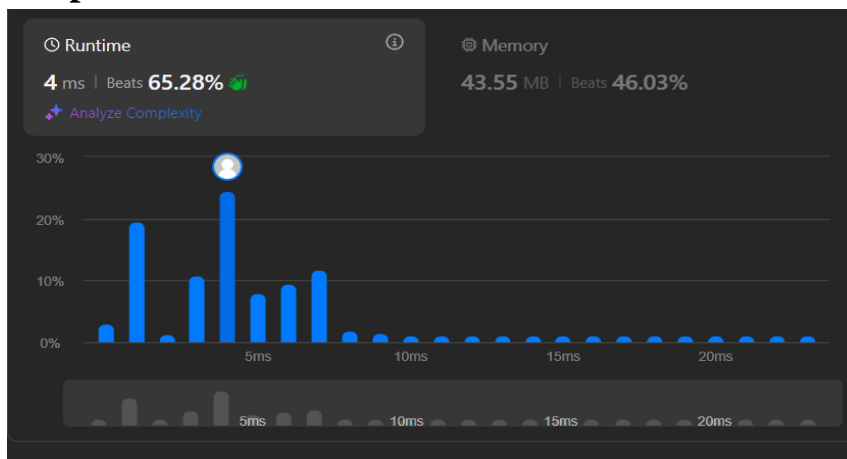


1. Problem 3: Third Maximum Number

2. Implementation/code:

```
class Solution {
    public int thirdMax(int[] nums) {
        Integer max1 = null;
        Integer max2 = null;
        Integer max3 = null;
        for (Integer n : nums) {
            if (n.equals(max1) || n.equals(max2) || n.equals(max3)) continue;
            if (max1 == null || n > max1) {
                max3 = max2;
                max2 = max1;
                max1 = n;
            } else if (max2 == null || n > max2) {
                max3 = max2;
                max2 = n;
            } else if (max3 == null || n > max3) {
                max3 = n; } }
        return max3 == null ? max1 : max3; } }
```

3. Output:



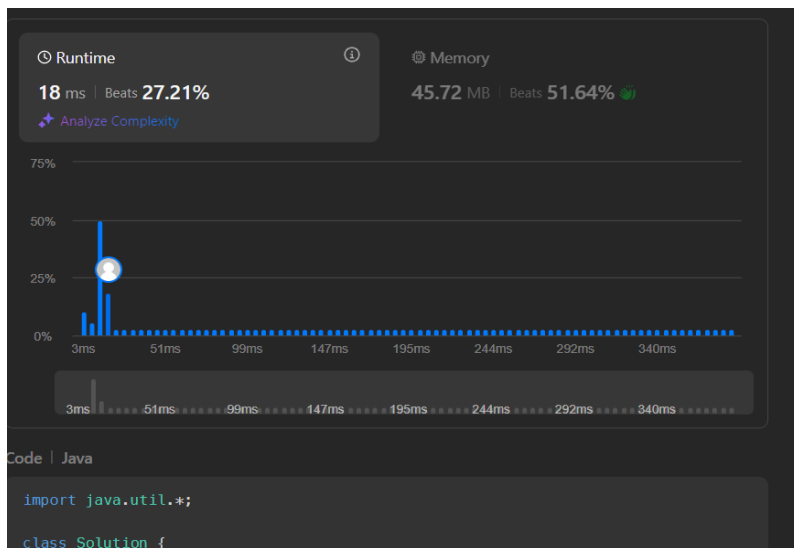
1. Problem 4: Sort Characters By Frequency

2. Implementation/code:

```
import java.util.*;

class Solution {
    public String frequencySort(String s) {
        Map<Character, Integer> frequencyMap = new HashMap<>();
        for (char c : s.toCharArray()) {
            frequencyMap.put(c, frequencyMap.getOrDefault(c, 0) + 1); }
        PriorityQueue<Character> maxHeap = new PriorityQueue<>((
            (a, b) -> frequencyMap.get(b) - frequencyMap.get(a) ));
        maxHeap.addAll(frequencyMap.keySet());
        StringBuilder result = new StringBuilder();
        while (!maxHeap.isEmpty()) {
            char c = maxHeap.poll();
            result.append(String.valueOf(c).repeat(frequencyMap.get(c))); }
        return result.toString();
    }
}
```

3. Output:

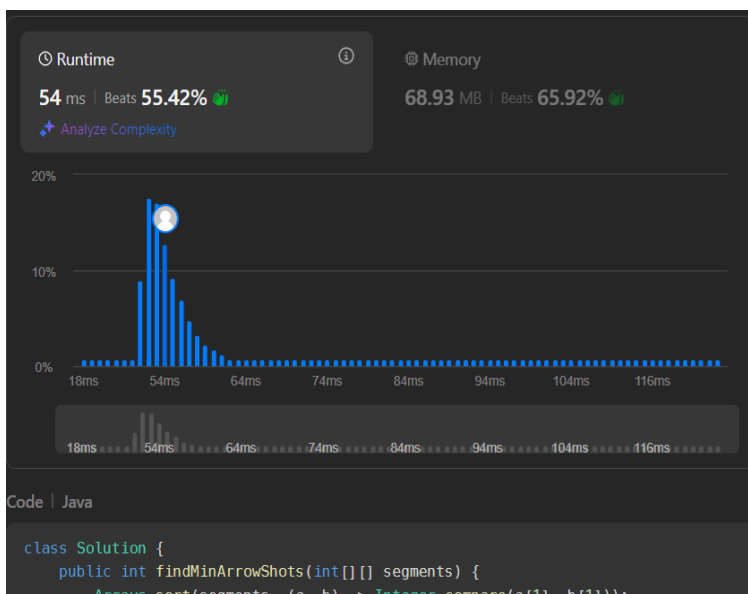


1. Problem 5: Minimum Number of Arrows to Burst Balloons

2. Implementation/Code:

```
class Solution {  
    public int findMinArrowShots(int[][] segments) {  
        Arrays.sort(segments, (a, b) -> Integer.compare(a[1], b[1]));  
        int ans = 0, arrow = 0;  
        for (int i = 0; i < segments.length; i++) {  
            if (ans == 0 || segments[i][0] > arrow) {  
                ans++;  
                arrow = segments[i][1];  
            }  
        }  
        return ans;  
    }  
}
```

3. Output:



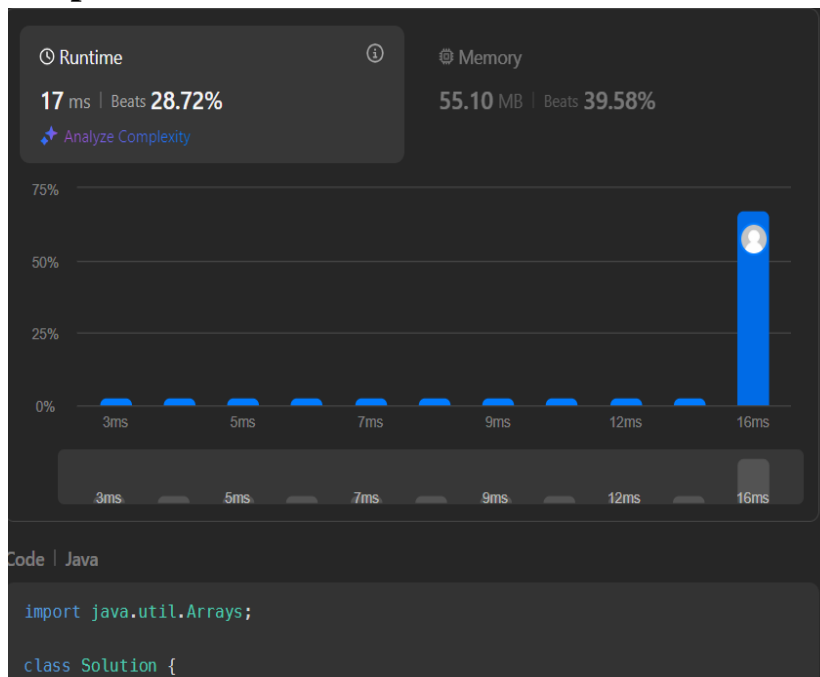
1. Problem 6: Boats to Save People

2. Implementation/Code:

```
import java.util.Arrays;
class Solution {
    public int numRescueBoats(int[] people, int limit) {
        Arrays.sort(people);
        int left = 0, right = people.length - 1;
        int boats = 0;
        while (left <= right) {
            if (people[left] + people[right] <= limit) {
                left++;
            }
            right--;
            boats++; }
        return boats; }}

```

3. Output:



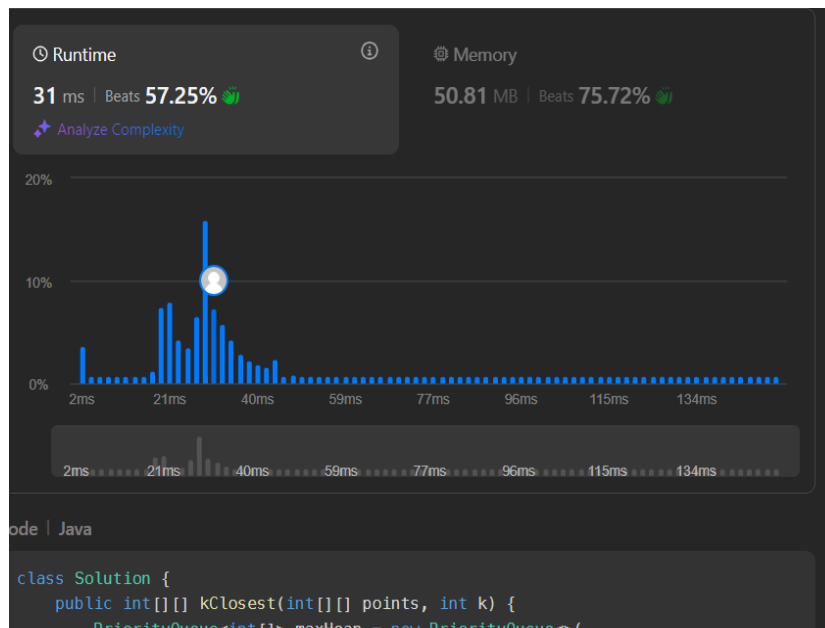
1. Problem 7: K Closest Points to Origin

2. Implementation/Code:

```
class Solution {
    public int[][] kClosest(int[][] points, int k) {
        PriorityQueue<int[]> maxHeap = new PriorityQueue<>(
            (a, b) -> Integer.compare((b[0] * b[0] + b[1] * b[1]), (a[0] * a[0] +
a[1] * a[1])) );

        for (int[] point : points) {
            maxHeap.add(point);
            if (maxHeap.size() > k) {
                maxHeap.poll(); } }
        int[][] result = new int[k][2];
        for (int i = 0; i < k; i++) {
            result[i] = maxHeap.poll(); }
        return result; } }
```

3. Output:



1. Problem 8: Reduce Array Size to The Half

2. Implementation/Code:

```
import java.util.*;

class Solution {
    public int minSetSize(int[] arr) {
        Map<Integer, Integer> freq = new HashMap<>();
        for (int num : arr) freq.put(num, freq.getDefault(num, 0) + 1);
        List<Integer> counts = new ArrayList<>(freq.values());
        counts.sort(Collections.reverseOrder());
        int res = 0, cnt = 0, half = arr.length / 2;
        for (int num : counts) {
            cnt += num;
            res++;
            if (cnt >= half) break; }
        return res; }}

```

3. Output:

