3RD OCTOBER ASSIGNMENT

Q1. Given an integer array, find the kth largest element using a priority queue.

```java
import java.util.PriorityQueue;

public int findKthLargest(int[] nums, int k) {
    PriorityQueue<Integer> minHeap = new PriorityQueue<>();

    for (int num : nums) {
        minHeap.add(num);
        if (minHeap.size() > k) {
            minHeap.poll(); // Remove the smallest element if the size exceeds k
        }
    }

    return minHeap.poll();
}
```

Q2. Given n ropes of different lengths, connect them into a single rope with minimum cost. Assume that the cost to connect two ropes is the same as the sum of their lengths.

```java
import java.util.PriorityQueue;

public int connectRopes(int[] ropes) {
    PriorityQueue<Integer> minHeap = new PriorityQueue<>();

    for (int rope : ropes) {
        minHeap.add(rope);
    }
```

```
    int cost = 0;

    while (minHeap.size() > 1) {
        int first = minHeap.poll();
        int second = minHeap.poll();
        int combined = first + second;
        cost += combined;
        minHeap.add(combined);
    }

    return cost;
}
```

Q3. Given an array of string 'words' and an integer k, return the k most frequent strings sorted by frequency and lexicographical order.

```java
import java.util.*;

public List<String> kMostFrequentWords(String[] words, int k) {
    Map<String, Integer> wordCount = new HashMap<>();

    for (String word : words) {
        wordCount.put(word, wordCount.getOrDefault(word, 0) + 1);
    }

    List<String> uniqueWords = new ArrayList<>(wordCount.keySet());

    Collections.sort(uniqueWords, (a, b) -> {
        if (wordCount.get(a).equals(wordCount.get(b))) {
            return a.compareTo(b);
        }
        return wordCount.get(b) - wordCount.get(a);
    });
```

```
    return uniqueWords.subList(0, k);
}
```

Q4. Find the weight of the last remaining stone after smashing the heaviest stones.

```java
import java.util.PriorityQueue;

public int lastStoneWeight(int[] stones) {
    PriorityQueue<Integer> maxHeap = new PriorityQueue<>((a, b) -> b - a);

    for (int stone : stones) {
        maxHeap.add(stone);
    }

    while (maxHeap.size() > 1) {
        int x = maxHeap.poll();
        int y = maxHeap.poll();

        if (x != y) {
            maxHeap.add(Math.abs(x - y));
        }
    }

    return maxHeap.isEmpty() ? 0 : maxHeap.poll();
}
```

Q5. Given a string s, rearrange the characters so that any two adjacent characters are not the same. Return any possible rearrangement or return "" if not possible.

```
java
Copy code
import java.util.*;

public String rearrangeString(String s) {
```

```
    Map<Character, Integer> charCount = new HashMap<>();
    for (char c : s.toCharArray()) {
        charCount.put(c, charCount.getOrDefault(c, 0) + 1);
    }

    PriorityQueue<Character> maxHeap = new PriorityQueue<>((a, b) ->
charCount.get(b) - charCount.get(a));
    maxHeap.addAll(charCount.keySet());

    Queue<Character> queue = new LinkedList<>();
    StringBuilder result = new StringBuilder();

    while (!maxHeap.isEmpty()) {
        char current = maxHeap.poll();
        result.append(current);
        charCount.put(current, charCount.get(current) - 1);
        queue.add(current);

        if (queue.size() < 2) {
            continue;
        }

        char last = queue.poll();
        if (charCount.get(last) > 0) {
            maxHeap.add(last);
        }
    }

    return result.length() == s.length() ? result.toString() : "";
}
```

Q6. Find the k pairs with the smallest sums from two sorted arrays.

```
java
Copy code
import java.util.*;
```

```
public List<int[]> kSmallestPairs(int[] nums1, int[] nums2, int k) {
    List<int[]> result = new ArrayList<>();
    if (nums1.length == 0 || nums2.length == 0) {
        return result;
    }

    PriorityQueue<int[]> minHeap = new PriorityQueue<>(
        (a, b) -> (nums1[a[0]] + nums2[a[1]]) - (nums1[b[0]] + nums2[b[1]]));

    for (int i = 0; i < nums1.length && i < k; i++) {
        minHeap.add(new int[]{i, 0});
    }

    while (k > 0 && !minHeap.isEmpty()) {
        int[] pair = minHeap.poll();
        result.add(new int[]{nums1[pair[0]], nums2[pair[1]]});

        if (pair[1] < nums2.length - 1) {
            minHeap.add(new int[]{pair[0], pair[1] + 1});
        }

        k--;
    }

    return result;
}
```

Q7. Find the maximum score in a stone-picking game with three piles of stones.

```
java
Copy code
public int maxScore(int a, int b, int c) {
    int[] piles = new int[] {a, b, c};
    Arrays.sort(piles);

    int score = 0;
```

```
    while (piles[0] < piles[2]) {
        piles[0]++;
        piles[2]--;
        Arrays.sort(piles);
        score++;
    }

    return score;
}
```