HASHING SOLUTIONS

Solution 1:

Using the concept of Horizontal Distance (HD), as discussed in Top View of a Binary Tree Question in the Trees chapter.

```java
import java.util.*;

class Solution {
    static class Node {
        int data;
        int hd;
        Node left, right;

        public Node(int key) {
            this.data = key;
            this.hd = Integer.MAX_VALUE;
            this.left = this.right = null;
        }
    }

    public static void bottomViewHelper(Node root, int curr, int hd,
TreeMap<Integer, int[]> m) {
        if (root == null)
            return;

        // If node for a particular HD is not present, add to the map.
        if (!m.containsKey(hd)) {
            m.put(hd, new int[]{ root.data, curr });
        }

        // Compare height for already
        // present node at similar horizontal
        // distance
        else {
            int[] p = m.get(hd);
            if (p[1] <= curr) {
                p[1] = curr;
```

```java
                p[0] = root.data;
        }
        m.put(hd, p);
    }


    // call for left subtree
    bottomViewHelper(root.left, curr + 1, hd - 1, m);


    // call for right subtree
    bottomViewHelper(root.right, curr + 1, hd + 1, m);
}


public static void printBottomView(Node root) {
    // Map to store Horizontal Distance, Height and Data.
    TreeMap<Integer, int[]> m = new TreeMap<>();


    bottomViewHelper(root, 0, 0, m);


    // Prints the values stored by printBottomViewUtil()
    for(int val[] : m.values())
    {
        System.out.print(val[0] + " ");
    }
}


public static void main(String[] args) {
    Node root = new Node(20);
    root.left = new Node(8);
    root.right = new Node(22);
    root.left.left = new Node(5);
    root.left.right = new Node(3);
    root.right.left = new Node(4);
    root.right.right = new Node(25);
    root.left.right.left = new Node(10);
    root.left.right.right = new Node(14);

    System.out.println(
        "Bottom view of the given binary tree:");

    printBottomView(root);
```

```
        }
}
```

Solution 2:

```java
public int[] twoSum(int[] arr, int target) {
      Map<Integer, Integer> visited = new HashMap<>();
      for(int i = 0; i<arr.length; i++) {
          //diff = given target - number given at ith index
          int diff = target - arr[i];

          // check if found difference is present in the MAP list
          if(visited.containsKey(diff)) {
              //if difference in map matches with the ith index element in array
              return new int[] { i, visited.get(diff) };
          }
          //add arr element in map to match with future element if forms a pair
          visited.put(arr[i],i);
      }
      //if no matches are found
      return new int[] {0, 0};
  }
```

Solution 3 :

```java
public String frequencySort(String s) {
      HashMap<Character , Integer>map = new HashMap<>();
      for(int i=0;i<s.length();++i)
          map.put(s.charAt(i),map.getOrDefault(s.charAt(i),0)+1);

   PriorityQueue<Map.Entry<Character,Integer>> pq =
      new PriorityQueue<>((a,b)->a.getValue()== b.getValue()?
       a.getKey()-b.getKey(): b.getValue()-a.getValue());

   for(Map.Entry<Character,Integer>e:map.entrySet())  pq.add(e);
```

```java
        StringBuilder res = new StringBuilder();
    while(pq.size()!=0){
        char ch = pq.poll().getKey();
        int val = map.get(ch);
      while(val!=0){
          res.append(ch);
          val--;
      }
    }
    return res.toString();
  }
```