Class Notes 1

- ▶ Python
- ▶ JavaScript
- ▼ Java

💡 Question 1

Given an array of size N. The task is to find the maximum and the minimum element of the array using the minimum number of comparisons.
Examples:
Input: arr[] = {3, 5, 4, 1, 9}
Output: Minimum element is: 1
Maximum element is: 9

TC : O(n)
SC: O(n)

```
// Java program of above implementation
// Java program of above implementation
public class MinMax {
  static int[] getMinMax(int arr[], int n) {
    int i;
int max =0, min =0;
    /*If there is only one element then return it as min and max both
*/
    if (n == 1) {
      max = arr[0];
      min = arr[0];
      return new int[]{max,min};
    }

    /* If there are more than one elements, then initialize min
  and max*/
    if (arr[0] > arr[1]) {
      max = arr[0];
      min = arr[1];
    } else {
      max = arr[1];
      min = arr[0];
    }

    for (i = 2; i < n; i++) {
      if (arr[i] > max) {
      max = arr[i];
      } else if (arr[i] < min) {
        min = arr[i];
      }
    }

    return new int[]{max,min};
  }

  /* Driver program to test above function */
  public static void main(String args[]) {
    int arr[] = {1000, 11, 445, 1, 330, 3000};
    int arr_size = 6;
    int[] minmax = getMinMax(arr, arr_size);
    System.out.printf("\nMinimum element is %d", minmax[1]);
    System.out.printf("\nMaximum element is %d", minmax[0]);

  }
}
```

💡 Question 2

You are given an array prices where prices[i] is the price of a given stock on the ith day.
You want to maximize your profit by choosing a single day to buy one stock and choosing a different day in the future to sell that stock.
Return the maximum profit you can achieve from this transaction. If you cannot achieve any profit, return 0.
Example :
Input: prices = [7,1,5,3,6,4]
Output: 5
Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5.
Note that buying on day 2 and selling on day 1 is not allowed because you must buy before you sell.
Solution:

```
public class Solution {
    public int maxProfit(int prices[]) {
        int minprice = Integer.MAX_VALUE;
        int maxprofit = 0;
        for (int i = 0; i < prices.length; i++) {
            if (prices[i] < minprice)
                minprice = prices[i];
            else if (prices[i] - minprice > maxprofit)
                maxprofit = prices[i] - minprice;
        }
        return maxprofit;
    }
}
```

Question 3

Given an integer array nums, find a subarray that has the largest product, and return the product.
The test cases are generated so that the answer will fit in a 32-bit integer.
Example:
Input: nums = [2,3,-2,4]
Output: 6
Explanation: [2,3] has the largest product 6.

```
class Solution {
    public int maxProduct(int[] nums) {
        if (nums.length == 0) return 0;

        int max_so_far = nums[0];
        int min_so_far = nums[0];
        int result = max_so_far;

        for (int i = 1; i < nums.length; i++) {
            int curr = nums[i];
            int temp_max = Math.max(curr, Math.max(max_so_far * curr,
min_so_far * curr));
            min_so_far = Math.min(curr, Math.min(max_so_far * curr, m
in_so_far * curr));

            max_so_far = temp_max;

            result = Math.max(max_so_far, result);
        }

        return result;
    }
}
```

Question 4

Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] such that i != j, i != k, and j != k, and nums[i] + nums[j] + nums[k] == 0.
Notice that the solution set must not contain duplicate triplets.
Example:
Input: nums = [-1,0,1,2,-1,-4]
Output: [[-1,-1,2],[-1,0,1]]
Explanation:
nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0.
nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0.
nums[0] + nums[3] + nums[4] = (-1) + 2 + (-1) = 0.
The distinct triplets are [-1,0,1] and [-1,-1,2].
Notice that the order of the output and the order of the triplets does not matter.

```
class Solution {
    public List<List<Integer>> threeSum(int[] nums) {
        Arrays.sort(nums);
        List<List<Integer>> res = new ArrayList<>();
        for (int i = 0; i < nums.length && nums[i] <= 0; ++i)
            if (i == 0 || nums[i - 1] != nums[i]) {
                twoSumII(nums, i, res);
            }
        return res;
    }
    void twoSumII(int[] nums, int i, List<List<Integer>> res) {
        int lo = i + 1, hi = nums.length - 1;
        while (lo < hi) {
            int sum = nums[i] + nums[lo] + nums[hi];
            if (sum < 0) {
                ++lo;
            } else if (sum > 0) {
                --hi;
            } else {
                res.add(Arrays.asList(nums[i], nums[lo++], nums[hi-
-]));

                while (lo < hi && nums[lo] == nums[lo - 1])
                    ++lo;
            }
        }
    }
}
```

Question 5

Given an integer array nums and an integer k, return the kth largest element in the array. Note that it is the kth largest element in the sorted order, not the kth distinct element.
Example 1:
Input: nums = [3,2,1,5,6,4], k = 2
Output: 5

```
class Solution {
    public int findKthLargest(int[] nums, int k) {
        // init heap 'the smallest element first'
        PriorityQueue<Integer> heap =
            new PriorityQueue<Integer>((n1, n2) -> n1 - n2);

        // keep k largest elements in the heap
```

```
        for (int n: nums) {
          heap.add(n);
          if (heap.size() > k)
            heap.remove;
        }

        // output
        return heap.remove();
    }
}
```

Question 6

Given an integer array nums and an integer k, return the kth smallest element in the array. Note that it is the kth smallest element in the sorted order, not the kth distinct element.
Example 1:
Input: nums = [3,2,1,5,6,4], k = 2
Output: 2

```
public class KthSmallestElement {
    public static int findKthSmallest(int[] nums, int k) {
        PriorityQueue<Integer> pq = new PriorityQueue<>();

        for (int num : nums) {
            pq.add(num);

            if (pq.size() > k) {
                pq.remove();
            }
        }

        return pq.peek();
    }

    public static void main(String[] args) {
        int[] nums = {3, 1, 5, 2, 4};
        int k = 2;
        int kthSmallest = findKthSmallest(nums, k);
        System.out.println("The " + k + "th smallest element is: " +
kthSmallest);
    }
}
```