ARRAYS
DATA
STRUCTURE

Content

1.1. Missing number in array

Statement : Given an array C of size N-1and given that there are numbers from 1 to N with one element missing, the missing number is to be found.

```
public static void main(String args[]) {
        Scanner sc = new Scanner ( System.in );
        int test = sc.nextInt();
        for(int  t = 0 ; t < test ; t++){
                int n = sc.nextInt();
                int arr[] = new int[n-1];
                for(int i = 0 ; i < n – 1 ; i++){
                        arr[i] = sc.nextInt();
                }
                int sum = n * (n + 1) / 2;
                for(int  i = 0 ; i < n – 1 ; i++){
                        sum = sum - arr[i];
                }
                System.out.println( sum );
        }
}
```

1.2 Majority Element

Statement : Find the majority element in the array. A majority element in an array A of size N is an **element that appears more than N/2 times in the array**.

```
public static void main( String args[] ) {
        Scanner sc = new Scanner (System.in);
        int test = sc.nextInt() ;
        for(int t = 0 ; t < test ; t ++){
                int n = sc.nextInt() ;
                int arr[] = new int[n];
                for(int i = 0 ; i < n ; i++){
                        arr[i] = sc.nextInt();
                }
                int ans = -1;
                Map< Integer, Integer > m = new HashMap<>() ;
                for(int i = 0 ; i < n ; i++){
                        Integer temp = m.get(arr[i]);
                        if(temp != null){
                                temp++;
                                if(temp >= n/2){
                                     ans = arr[i];
                                     break;
                                 }
                                 m.put(arr[i], temp);
                        }else{
                                m.put(arr[i], 1);
                        }
                }
                System.out.println( ans );
        }
}
```

1.3. Leaders in an array

Statement : An element of array is leader if it is greater than or equal to all the elements to its right side. Also, the rightmost element is always a leader. Your task is to find the leaders in the array.

```
class GFG {
	public static void main (String[] args) {
		Scanner sc = new Scanner(System.in);
		int test = sc.nextInt();
		for(int t=0;t<test;t++){
			int n = sc.nextInt();
			int arr[] = new int[n];
			for(int i=0;i<n;i++){
				arr[i] = sc.nextInt();
			}
			LinkedList<Integer> stack = new LinkedList<>();
			int last = arr[n-1];
			stack.push(last);
			for(int i=n-2;i>=0;i--){
				if(arr[i]>=last){
					last = arr[i];
					stack.push(last);
				}
			}
			while(!stack.isEmpty()){
				System.out.print(stack.pop()+" ");
			}
			System.out.println();
		}
	}
}
```

1.4. Equilibrium point

Statement : Given an array A of N positive numbers. The task is to find the position where equilibrium first occurs in the array. Equilibrium position in an array is a position such that the sum of elements before it is equal to the sum of elements after it.

```
public static void main (String[] args) {
        Scanner sc = new Scanner(System.in);
        int test = sc.nextInt();
        for(int t=0;t<test;t++){
                int n = sc.nextInt();
                int arr[] = new int[n];
                int rightSum = 0;
                for(int i=0;i<n;i++){
                        arr[i] = sc.nextInt();
                        rightSum += arr[i];
                }
                int leftSum = 0;
                int ans = -1;
                for(int i=0;i<n;i++){
                        rightSum -= arr[i];
                        if(leftSum==rightSum){
                                ans = i+1;
                                break;
                        }
                        leftSum += arr[i];
                }
                System.out.println(ans);
        }
}
```

1.5. Largest subarray of 0's and 1's

```
// Returns largest subarray with equal number of 0s and 1s
int maxLen(int arr[], int n){
        // Creates an empty hashMap hM
        HashMap<Integer, Integer> hM = new HashMap<Integer, Integer>();

        int sum = 0;  // Initialize sum of elements
        int max_len = 0; // Initialize result
        int ending_index = -1;
        int start_index = 0;
        for (int i = 0; i < n; i++){
                arr[i] = (arr[i] == 0) ? -1 : 1;
        }
        // Traverse through the given array
        for (int i = 0; i < n; i++){
                // Add current element to sum
                sum += arr[i];

                // To handle sum=0 at last index
                if (sum == 0){
                        max_len = i + 1;
                        ending_index = i;
                }
                // If this sum is seen before, then update max_len if required
                if (hM.containsKey(sum + n)){
                        if (max_len < i - hM.get(sum + n)){
                                max_len = i - hM.get(sum + n);
                                ending_index = i;
                        }
                }else // Else put this sum in hash table
                        hM.put(sum + n, i);
        }
        for (int i = 0; i < n; i++){
                arr[i] = (arr[i] == -1) ? 0 : 1;
        }
        int end = ending_index - max_len + 1;
        System.out.println(end + " to " + ending_index);
        return max_len;
}
```

1.6. Subarray with given sum

```
class GFG {
	public static void main (String[] args) {
		Scanner sc = new Scanner(System.in);
		int test = sc.nextInt();
		for(int t=0;t<test;t++){
			int n = sc.nextInt();
			int sum = sc.nextInt();
			int arr[] = new int[n];
			for(int i=0;i<n;i++){
				arr[i] = sc.nextInt();
			}
			int start = 0,temp = arr[0];
			int i;
			for(i=1;i<=n;i++){
				while(temp>sum && start<i-1)
					temp = temp - arr[start++];

				if(temp==sum)
					break;

				if(i<n)
					temp = temp + arr[i];
			}
			if(temp!=sum){
				System.out.println(-1);
			}else{
				System.out.println(start + 1 + " " + i);
			}
		}
	}
}
```

1.7. Sort an array of 0s, 1s and 2s

```
class GFG {
	public static void main (String[] args) {
		Scanner sc = new Scanner(System.in);
		int test = sc.nextInt();
		for(int t = 0 ; t < test ; t++){
			int n = sc.nextInt();
			int temp[] = new int[3];
			for(int i = 0 ; i < n ; i++){
				int no = sc.nextInt();
				temp[no]++;
			}
			int arr[] = new int[n];
			int k = 0;
			for(int i = 0 ; i < 3 ; i++){
				int j = 0;
				while(j < temp[i]){
					arr[k++] = i;
					j++;
				}
			}
			for(int i = 0 ; i < n ; i++){
				System.out.print(arr[i]+" ");
			}
			System.out.println();
		}
	}
}
```

1.8. Inversion of array

1. simple (o(n^2))

```
public static void main (String[] args) {
        Scanner sc = new Scanner(System.in);
        int test = sc.nextInt();
        for(int t=0;t<test;t++){
                int n = sc.nextInt();
                int arr[] = new int[n];
                for(int i=0;i<n;i++){
                        arr[i] = sc.nextInt();
                }
                int count = 0;
                for(int i=0;i<n-1;i++){
                        for(int j=i+1;j<n;j++){
                                if(arr[i]>arr[j])
                                        count++;
                        }
                }
                System.out.println(count);
        }
}
```

2. Using Merge Sort (o(nlogn))

```
class GFG {
        static long mergeCount(int arr[],int l,int m,int r){
                int i = 0,j = 0,k = l;
                long swaps = 0;
                int left[] = Arrays.copyOfRange(arr,l,m+1);
                int right[] = Arrays.copyOfRange(arr,m+1,r+1);
                while ( i < left.length && j < right.length){
                        if(left[ i ] <= right[ j ]){
                                arr[k++] = left[i++];
                        }else{
                                arr[k++] = right[j++];
                                swaps += (m+1)-(l+i);
                        }
                }
                while(i < left.length){
```

```
                arr[k++] = left[i++];
            }
            while(j<right.length){
                arr[k++] = right[j++];
            }
            return swaps;
        }
        static long mergeAndCount(int arr[],int l,int r){
            long count = 0;
            if( l < r ){
                int m = (l+r)/2;
                    count += mergeAndCount(arr,l,m);
                    count += mergeAndCount(arr,m+1,r);
                    count += mergeCount(arr,l,m,r);
            }
            return count;
        }
        public static void main (String[] args) {
            Scanner sc = new Scanner(System.in);
            int test = sc.nextInt();
            for(int t=0;t<test;t++){
                int n = sc.nextInt();
                int arr[] = new int[n];
                for(int i=0;i<n;i++){
                    arr[i] = sc.nextInt();
                }
                System.out.println(mergeAndCount(arr,0,n-1));
            }
        }
}
```

1.9. Kadane's Algorithm

Statement : Given an array arr of N integers. Find the contiguous sub-array with maximum sum.

```
public static void main(String args[]) {
        Scanner sc = new Scanner (System.in);
        int test = sc.nextInt();
        for(int t=0;t<test;t++){
                int n = sc.nextInt();
                int arr[] = new int[n];
                for(int i=0;i<n;i++){
                        arr[i] = sc.nextInt();
                }
                 int max_so_far = arr[0];
                 int curr_max = arr[0];
                 for(int i=1;i<n;i++){
                         if(curr_max+arr[i]>arr[i]){
                                 curr_max = curr_max+arr[i];
                         }else{
                                 curr_max = arr[i];
                         }
                         if(max_so_far<curr_max){
                                 max_so_far = curr_max;
                          }
                 }
                 System.out.println(max_so_far);
        }
}
```

1.10. Stock buy and sell

Statement : The cost of stock on each day is given in an array **A[]** of size **N**. Find all the days on which you buy and sell the stock so that in between those days your profit is maximum.

```
// This function finds the buy sell
// schedule for maximum profit
void stockBuySell(int price[], int n) {
        // Prices must be given for at least two days
        if (n == 1)
                return;
        // Traverse through given price array
        int i = 0;
        while (i < n - 1) {
                // Find Local Minima
                // Note that the limit is (n-2) as we are
                // comparing present element to the next element
                while ((i < n - 1) && (price[i + 1] <= price[i]))
                        i++;

                // If we reached the end, break
                // as no further solution possible
                if (i == n - 1)
                        break;

                // Store the index of minima
                int buy = i++;

                // Find Local Maxima
                // Note that the limit is (n-1) as we are
                // comparing to previous element
                while ((i < n) && (price[i] >= price[i - 1]))
                        i++;

                // Store the index of maxima
                int sell = i - 1;

                System.out.println( buy + " " + sell ) ;
        }
}
```

1.11. Kth smallest element nlogn

Statement : Given an array **arr[]** and a number **K** where K is smaller than size of array, the task is to find the **Kth smallest** element in the given array. It is given that all array elements are distinct.

```
class GFG {
	static void merge(int arr[],int l,int m,int r){
		int left[] = new int[m-l+1];
		int right[] = new int[r-m];
		for(int i=0;i<m-l+1;i++)
			left[i] = arr[l+i];
		for(int i=0;i<r-m;i++)
			right[i] = arr[m+1+i];
		int i = 0,j = 0,k = l;
		while(i<left.length && j<right.length){
			if(left[i]<=right[j])
				arr[k++] = left[i++];
			else
				arr[k++] = right[j++];
		}
		while(i<left.length)
			arr[k++] = left[i++];
		while(j<right.length)
			arr[k++] = right[j++];
	}
	static void mergeSort(int arr[],int l,int r){
		if(l<r){
			int m = (l+r)/2;
			mergeSort(arr,l,m);
			mergeSort(arr,m+1,r);
			merge(arr,l,m,r);
		}
	}
	public static void main (String[] args) {
		Scanner sc = new Scanner(System.in);
		int test = sc.nextInt();
		for(int t=0;t<test;t++){
			int n = sc.nextInt();
			int arr[] = new int[n];
			for(int i=0;i<n;i++)
				arr[i] = sc.nextInt();
```

```
                    mergeSort(arr,0,n-1);
                    int k = sc.nextInt();
                    System.out.println(arr[k-1]);
                }
        }
}
```

1.12. Number of pairs

```
class Test
{
	// Function to return count of pairs with x as one element
	// of the pair. It mainly looks for all values in Y[] where
	// x ^ Y[i] > Y[i] ^ x
	static int count(int x, int Y[], int n, int NoOfY[])
	{
		// If x is 0, then there cannot be any value in Y such that
		// x^Y[i] > Y[i]^x
		if (x == 0) return 0;

		// If x is 1, then the number of pais is equal to number of
		// zeroes in Y[]
		if (x == 1) return NoOfY[0];

		// Find number of elements in Y[] with values greater than x
		// getting upperbound of x with binary search
		int idx = Arrays.binarySearch(Y, x);
		int ans;
		if(idx < 0){
			idx = Math.abs(idx+1);
			ans = Y.length - idx;
		}else{
			while (idx<n && Y[idx]==x) {
				idx++;
			}
			ans = Y.length - idx;
		}

		// If we have reached here, then x must be greater than 1,
		// increase number of pairs for y=0 and y=1
		ans += (NoOfY[0] + NoOfY[1]);

		// Decrease number of pairs for x=2 and (y=4 or y=3)
		if (x == 2) ans -= (NoOfY[3] + NoOfY[4]);

		// Increase number of pairs for x=3 and y=2
		if (x == 3) ans += NoOfY[2];
```

```
        return ans;
    }

    // Function to returns count of pairs (x, y) such that
    // x belongs to X[], y belongs to Y[] and x^y > y^x
    static int countPairs(int X[], int Y[], int m, int n) {
        // To store counts of 0, 1, 2, 3 and 4 in array Y
        int NoOfY[] = new int[5];
        for (int i = 0; i < n; i++)
            if (Y[i] < 5)
                NoOfY[Y[i]]++;

        // Sort Y[] so that we can do binary search in it
        Arrays.sort(Y);

        int total_pairs = 0; // Initialize result

        // Take every element of X and count pairs with it
        for (int i=0; i<m; i++)
            total_pairs += count(X[i], Y, n, NoOfY);

        return total_pairs;
    }

    // Driver method
    public static void main(String args[])
    {
        int X[] = {2, 1, 6};
        int Y[] = {1, 5};

        System.out.println("Total pairs = " + countPairs(X, Y, X.length,
        Y.length));
    }
}
```

1.13. Rearrange Array Alternately

Statement : Given a sorted array of positive integers. Your task is to rearrange the array elements alternatively i.e first element should be max value, second should be min value, third should be second max, fourth should be second min and so on...

```
public static void main (String[] args) {
        Scanner sc = new Scanner (System.in);
        int test = sc.nextInt();
        for(int t=0;t<test;t++){
                int n = sc.nextInt();
                int temp[] = new int[n];
                for(int i=0;i<n;i++){
                        temp[i] = sc.nextInt();
                }
                 StringBuffer sb = new StringBuffer();

                 for(int i=0,j=n-1,z=0;i<j;)
                 {
                        sb.append(temp[j--]+" ");
                        sb.append(temp[i++]+" ");
                 }

                if(n%2!=0)
                        sb.append(temp[temp.length/2]);

                System.out.println(sb);
        }
}
```

1.14. Minimum Platforms

Statement : Given arrival and departure times of all trains that reach a railway station. Your task is to find the minimum number of platforms required for the railway station so that no train waits.
Note: Consider that all the trains arrive on the same day and leave on the same day. Also, arrival and departure times will not be same for a train, but we can have arrival time of one train equal to departure of the other.

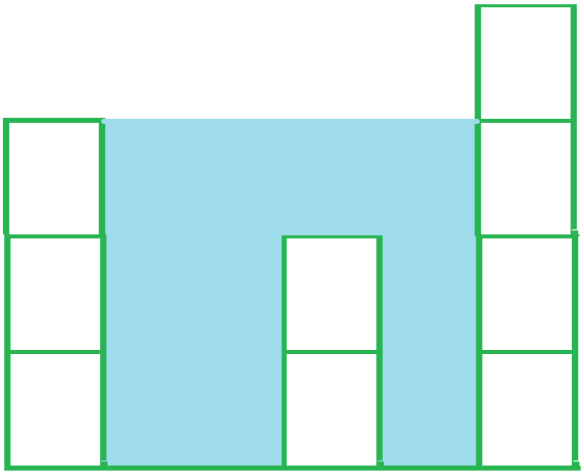
In such cases, **we need different platforms,** i.e at any given instance of time, **same platform can not be used for both departure of a train and arrival of another.**

```
static int findPlatform(int arr[], int dep[], int n) {
        // Sort arrival and departure arrays
        Arrays.sort(arr);
        Arrays.sort(dep);

        // plat_needed indicates number of platforms needed at a time
        int plat_needed = 1, result = 1, i = 1, j = 0;

        // Similar to merge in merge sort to process all events in sorted order
        while (i < n && j < n){
                // If next event in sorted order is arrival, increment count of platforms
                needed
                if (arr[i] <= dep[j]){
                        plat_needed++;
                        i++;

                        // Update result if needed
                        if (plat_needed > result)
                                result = plat_needed;
                }

                // Else decrement count of platforms needed
                else{
                        plat_needed--;
                        j++;
                }
        }
        return result;
}
```

1.15. Trapping Rain Water

Statement : Given an array **arr[]** of **N** non-negative integers representing height of blocks at index i as **Ai** where the width of each block is 1. Compute how much water can be trapped in between blocks after raining.

```
static int trappingWater(int arr[], int n) {

        int left[] = new int[n];
        int right[] = new int[n];
        //fill left array
        left[0] = arr[0];
        for(int i=1;i<n;i++){
                left[i] = Math.max(left[i-1],arr[i]);
        }
        //fill right array
        right[n-1] = arr[n-1];
        for(int i=n-2;i>=0;i--){
                right[i] = Math.max(right[i+1],arr[i]);
        }
        int water = 0;
        for(int i=0;i<n;i++){
                water += Math.min(left[i],right[i]) - arr[i];
        }
        return water;
}
```



Bars for input {3, 0, 0, 2, 0, 4}
Total trapped water = 3 + 3 + 1 + 3 = 10

1.16. Boolean Matrix Problem

Statement : Given a boolean matrix **mat[M][N]** of size **M** X **N**, modify it such that if a matrix cell **mat[i][j]** is **1** (or true) then make all the cells of ith row and jth column as **1**.

```
public static void main(String args[]) {
        Scanner sc = new Scanner (System.in);
        int test = sc.nextInt();
        for(int t=0;t<test;t++){
                int n = sc.nextInt();
                int m = sc.nextInt();
                int mat[][] = new int[n][m];
                for(int i=0;i<n;i++){
                        for(int j=0;j<m;j++){
                                mat[i][j] = sc.nextInt();
                        }
                }
                int row[] = new int[n];
                int col[] = new int[m];
                for(int i=0;i<n;i++){
                        for(int j=0;j<m;j++){
                                if(mat[i][j]==1){
                                        row[i] = 1;
                                        col[j] = 1;
                                }
                        }
                }
                for(int i=0;i<n;i++){
                        for(int j=0;j<m;j++){
                                if(row[i]==1 || col[j]==1)
                                        System.out.print(1 + " ");
                                else
                                        System.out.print(0 + " ");
                        }
                        System.out.println();
                }
        }
}
```

1.17. Merge Without Extra Space

```
int nextGap( int gap ){
        if ( gap < = 1)
                return 0;
        return (gap/2)+(gap%2);
}
public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        int test = sc.nextInt();
        for ( int t = 0 ; t <test ; t ++){
                int n = sc.nextInt();
                int m = sc.nextInt();
                int first[] = new int[n];
                int sec[] = new int[m];
                for ( int i = 0 ; i < n ; i++){
                        first[i] = sc.nextInt();
                }
                for(int i=0;i<m;i++){
                        sec[i] = sc.nextInt();
                }
                //processing
                int i,j,gap = n+m;
                for ( gap = nextGap ( gap ) ; gap > 0 ; gap = nextGap ( gap ) ) {
                        for ( i = 0 ; i + gap < n ; i ++){
                                if ( first [ i ] > ( first [ i + gap ] ) ) {
                                        int temp = first[i];
                                        first[i] = first[i+gap];
                                        first[i+gap] = temp;
                                }
                        }
                        for ( j = gap > n ? gap – n : 0 ; i < n && j < m ; i + + , j + +){
                                if ( first [ i ] >  sec  [ j ] ){
                                        int temp = first[i];
                                        first[i] = sec[j];
                                        sec[j] = temp;
                                }
                        }
                        if(j < m){
                                for ( j = 0 ; j + gap < m ; j ++){
                                        if ( sec [ j ] > (sec [ j + gap ] ) ) {
```

```
                                        int temp = sec[j];
                                        sec[j] = sec[j+gap];
                                        sec[j+gap] = temp;
                                    }
                                }
                            }
                        }
                        //print
                        for ( int i = 0 ; i < n ; i ++){
                            System.out.print( first[i] + “  “);
                        }
                        for(int i = 0 ; i < m ; i + +){
                            System.out.print( sec[i] + " " );
                        }
                        System.out.println();
                    }
                    return 0;
}
```

1.18. Binary Search

```
int bin_search(int A[], int left, int right,  int k)
{
        if(left <= right){
                int mid = left + ( right - left + 1 ) / 2;
                if(A[mid] == k){
                        return mid;
                }else if(k < A[mid]){
                        return bin_search(A, left, mid - 1, k);
                }else{
                        return bin_search(A, mid + 1, right, k);
                }
        }
        return -1;
}
```