**Exercise No.** : 2

**Topics Covered** : Basic Math, Control Flow, Arrays, Functions, Bit Manipulation

**Date** : 09-05-2024

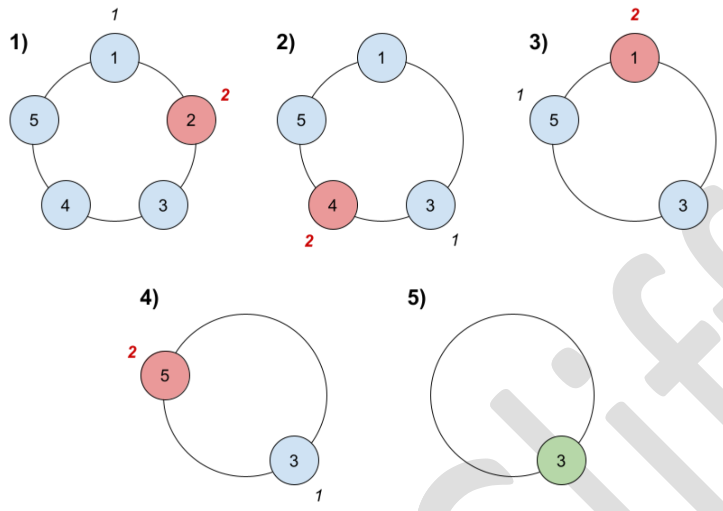
**Solve the following problems**

| Q No. | Question Detail | Level |
|---|---|---|
| 1 | **Fibonacci Sum**<br>**Problem statement** : Given a number positive number N, find value of f0 + f1 + f2 + . + fN where fi indicates ith Fibonacci number. Remember that f0 = 0, f1 = 1, f2 = 1, f3 = 2, f4 = 3, f5 = 5, Since the answer can be very large, answer modulo 1000000007 should be returned.<br><br>**Example 1:**<br>**Input:**<br>N = 3<br>**Output:**<br>4<br>**Explanation**:<br>0 + 1 + 1 + 2 = 4<br><br>**Example 2:**<br>**Input :**<br>N = 4<br>**Output :**<br>7<br>**Explanation :**<br>0 + 1 + 1 + 2 + 3 = 7<br><br>**Constraints:**<br>1 <= N <= 100000 | Medium |

| | | |
|---|---|---|
| **2** | **The kth Factor of n**<br>**Problem statement** : You are given two positive integers n and k. A factor of an integer n is defined as an integer i where n % i == 0. Consider a list of all factors of n sorted in ascending order, return the kth factor in this list or return -1 if n has less than k factors.<br><br>**Example 1:**<br>**Input:** n = 12, k = 3<br>**Output:** 3<br>**Explanation:** Factors list is [1, 2, 3, 4, 6, 12], the 3rd factor is 3.<br><br>**Example 2:**<br>**Input:** n = 7, k = 2<br>**Output:** 7<br>**Explanation:** Factors list is [1, 7], the 2nd factor is 7.<br><br>**Constraints:**<br>• 1 <= k <= n <= 1000 | Medium |
| **3** | **Find the Winner of the Circular Game**<br>**Problem Statement:** There are n friends that are playing a game. The friends are sitting in a circle and are numbered from 1 to n in clockwise order. More formally, moving clockwise from the ith friend brings you to the (i+1)th friend for 1 <= i < n, and moving clockwise from the nth friend brings you to the 1st friend.<br>The rules of the game are as follows:<br>1. Start at the 1st friend.<br>2. Count the next k friends in the clockwise direction including the friend you started at. The counting wraps around the circle and may count some friends more than once.<br>3. The last friend you counted leaves the circle and loses the game.<br>4. If there is still more than one friend in the circle, go back to step 2 starting from the friend immediately clockwise of the friend who just lost and repeat.<br>5. Else, the last friend in the circle wins the game. | Medium |

Given the number of friends, n, and an integer k, return the winner of the game.

**Example 1:**



**Input:** n = 5, k = 2
**Output:** 3
**Explanation:** Here are the steps of the game:
1) Start at friend 1.
2) Count 2 friends clockwise, which are friends 1 and 2.
3) Friend 2 leaves the circle. Next start is friend 3.
4) Count 2 friends clockwise, which are friends 3 and 4.
5) Friend 4 leaves the circle. Next start is friend 5.
6) Count 2 friends clockwise, which are friends 5 and 1.
7) Friend 1 leaves the circle. Next start is friend 3.
8) Count 2 friends clockwise, which are friends 3 and 5.
9) Friend 5 leaves the circle. Only friend 3 is left, so they are the winner.

**Example 2:**
**Input:** n = 6, k = 5
**Output**: 1
**Explanation:** The friends leave in this order: 5, 4, 6, 2, 3. The winner is friend 1.

**Constraints:**

- 1 <= k <= n <= 500

| | | |
|---|---|---|
| 4 | **Egg Drop With 2 Eggs and N Floors**<br>**Problem Statement:** You are given two identical eggs and you have access to a building with n floors labeled from 1 to n.<br>You know that there exists a floor f where 0 <= f <= n such that any egg dropped at a floor higher than f will break, and any egg dropped at or below floor f will not break.<br>In each move, you may take an unbroken egg and drop it from any floor x (where 1 <= x <= n). If the egg breaks, you can no longer use it. However, if the egg does not break, you may reuse it in future moves.<br>Return the minimum number of moves that you need to determine with certainty what the value of f is.<br><br>**Example 1:**<br>**Input**: n = 2<br>**Output:** 2<br>**Explanation:** We can drop the first egg from floor 1 and the second egg from floor 2.<br>If the first egg breaks, we know that f = 0.<br>If the second egg breaks but the first egg didn't, we know that f = 1.<br>Otherwise, if both eggs survive, we know that f = 2.<br><br>**Example 2:**<br>**Input:** n = 100<br>**Output:** 14<br>**Explanation:** One optimal strategy is:<br>- Drop the 1st egg at floor 9. If it breaks, we know f is between 0 and 8. Drop the 2nd egg starting from floor 1 and going up one at a time to find f within 8 more drops. Total drops is 1 + 8 = 9.<br>- If the 1st egg does not break, drop the 1st egg again at floor 22. If it breaks, we know f is between 9 and 21. Drop the 2nd egg starting from floor 10 and going up one at a time to find f within 12 more drops. Total drops is 2 + 12 = 14.<br>- If the 1st egg does not break again, follow a similar process | Medium |

| | | |
|---|---|---|
| | dropping the 1st egg from floors 34, 45, 55, 64, 72, 79, 85, 90, 94, 97, 99, and 100.<br><br>**Constraints:**<br>• 1 <= n <= 1000 | |
| **5** | **Airplane Seat Assignment Probability**<br>**Problem Statement :** n passengers board an airplane with exactly n seats. The first passenger has lost the ticket and picks a seat randomly. But after that, the rest of the passengers will:<br>• Take their own seat if it is still available, and<br>• Pick other seats randomly when they find their seat occupied<br>Return the probability that the nth person gets his own seat.<br><br>**Example 1:**<br>**Input**: n = 1<br>**Output**: 1.00000<br>**Explanation:** The first person can only get the first seat.<br><br>**Example 2:**<br>**Input**: n = 2<br>**Output:** 0.50000<br>**Explanation:** The second person has a probability of 0.5 to get the second seat (when first person gets the first seat).<br><br>**Constraints:**<br>• 1 <= n <= 10^5 | Medium |
| **6** | **Closest Divisor**<br><br>**Problem Statement:** Given an integer num, find the closest two integers in absolute difference whose product equals num + 1 or num + 2.<br>Return the two integers in any order.<br><br>**Example 1:**<br>**Input:** num = 8<br>**Output:** [3,3] | Medium |

| | | |
|---|---|---|
| | **Explanation:** For num + 1 = 9, the closest divisors are 3 & 3, for num + 2 = 10, the closest divisors are 2 & 5, hence 3 & 3 is chosen.<br><br>**Example 2:**<br>**Input:** num = 123<br>**Output:** [5,25]<br><br>**Constraints:**<br>• 1 <= num <= 10^9 | |
| **7** | **Special Numbers**<br>**Problem statement** : Number is a special number if it's digits only consist 0, 1, 2, 3, 4 or 5. Given a number N and we have to find N-th Special Number.<br><br>**Example 1:**<br>**Input:**<br>N = 6<br>**Output**: 5<br>**Explanation:** First 6 numbers are ( 0, 1, 2, 3, 4, 5 )<br><br>**Example 2:**<br>**Input:**<br>N = 7<br>**Output:** 10<br>**Explanation:** First 7 numbers are ( 0, 1, 2, 3, 4, 5, 10 ) | Medium |
| **8** | **NCr**<br>**Problem statement** : Given two integers n and r, find nCr. Since the answer may be very large, calculate the answer modulo 10^9+7.<br><br>**Example 1:**<br>**Input:** n = 3, r = 2<br>**Output:** 3<br>**Explaination:** 3C2 = 3.<br><br>**Example 2:** | Medium |

| | | |
|---|---|---|
| | **Input**: n = 2, r = 4<br>**Output:** 0<br>**Explaination**: r is greater than n.<br><br>**Constraints:**<br>1 ≤ n ≤ 1000<br>1 ≤ r ≤ 800 | |
| **9** | **Void of diamond**<br>**Problem statement** : You are given an integer ‘N’, ‘N’ will always be an odd integer. Your task is to print a pattern with the following description:<br>1. The pattern will consist of ‘N’ lines.<br>2. The pattern will consist of ‘ ‘ (space) and ‘*’ characters only.<br>3. The pattern will be a “Void of Diamond” pattern.<br>4. A “Void of Diamond” pattern is a pattern ‘N’ * ‘N’ cells and ‘ ‘ characters make a diamond shape and ‘*’ fill all other points.<br>5. For a better understanding of the “Void of Diamond” pattern refer to example and sample input-output.<br><br>**For example:**<br>If ‘N’ is 5 then the pattern will be-<br><br>`*****`<br>`** **`<br>`*   *`<br>`** **`<br>`*****`<br>**Sample Input 1:**<br>2<br>5<br>3<br>**Sample Output 1:**<br>`*****`<br>`** **`<br>`*   *`<br>`** **` | Medium |

*Little practice is worth more than a ton of theory*

```
*****

***
* *
***
```

**Explanation of Sample Input 1:**

**Test Case 1:**

Given ‘N’ = 5

We will print the pattern as the description of the “Void of Diamond” pattern.

**Test Case 2:**

Given ‘N’ = 3

There will be only 1 ‘ ‘ space in the diamond pattern.

**Sample Input 2:**

```
2
7
9
```

**Sample Output 2:**

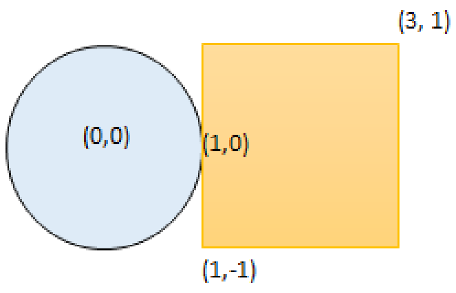
```
*******
*** ***
**   **
*     *
**   **
*** ***
*******

*********
**** ****
***   ***
**     **
*       *
**     **
***   ***
**** ****
*********
```

|  |  |  |
|---|---|---|
|  | **Explanation of Sample Input 2:**<br>**Test Case 1:**<br>Given 'N' = 7<br>The pattern is printed such that ' ' making a diamond and '*' filling other points.<br><br>**Test Case 2:**<br>Given 'N' = 9<br>Created a 9*9 grid and all space cells make a diamond pattern. |  |
| **10** | **Pyramid star pattern**<br><br>**Problem Statement:** Print a pattern of stars in the shape of a right-angled triangle which resembles a pyramid , with the base increasing in size as shown below. The number of rows in the triangle should be determined by user input.<br><br>**Example:**<br>**Input 1:** N=5<br>**Output 1:**<br>`*`<br>`* *`<br>`* * *`<br>`* * * *`<br>`* * * * *`<br>**Constraints:**<br>1<=N<= 10 | Medium |
| **11** | **Triangle of numbers**<br>**Problem statement :** You are given a pattern. Now you need to print the same pattern for any given 'N' number of rows.<br>For example, Pattern for 'N' = 4 will be.<br>1<br>232<br>34545<br>4567654 | Medium |

| | | |
|---|---|---|
| | **Sample Input1 :**<br>1<br>5<br>**Sample Output1 :**<br>1<br>232<br>34543<br>4567654<br>567898765<br>**Explanation of Sample Input 1:**<br>**For test case 1:**<br>We print the given pattern for the given 5 rows where each row prints spaces initially for each row decrementing for each consecutive row and printing the values corresponding to each row according to the given pattern.<br><br>**Sample Input2 :**<br>1<br>4<br>**Sample Output2 :**<br>1<br>232<br>34545<br>4567654<br>**Explanation of Sample Input 2:**<br>**For test case 1:**<br>We print the given pattern for the given 4 rows where each row prints spaces initially for each row decrementing for each consecutive row and printing the values corresponding to each row according to the given pattern. | |
| **12** | **Circle and Rectangle Overlapping**<br>**Problem statement** : You are given a circle represented as (radius, xCenter, yCenter) and an axis-aligned rectangle represented as (x1, y1, x2, y2), where (x1, y1) are the coordinates of the bottom-left corner, and (x2, y2) are the coordinates of the top-right corner of the rectangle. | Medium |

Return true if the circle and rectangle are overlapped otherwise return false. In other words, check if there is any point (xi, yi) that belongs to the circle and the rectangle at the same time.

**Example 1:**

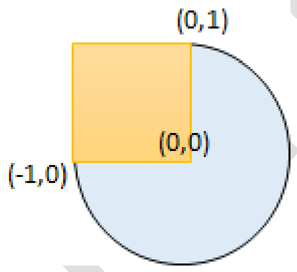


**Input:** radius = 1, xCenter = 0, yCenter = 0, x1 = 1, y1 = -1, x2 = 3, y2 = 1

**Output**: true

**Explanation:** Circle and rectangle share the point (1,0).

**Example 2**:



**Input:** radius = 1, xCenter = 1, yCenter = 1, x1 = 1, y1 = -3, x2 = 2, y2 = -1

**Output**: false

**Constraints**:

- $1 <= radius <= 2000$
- $-10^4 <= xCenter, yCenter <= 10^4$
- $-10^4 <= x1 < x2 <= 10^4$
- $-10^4 <= y1 < y2 <= 10^4$

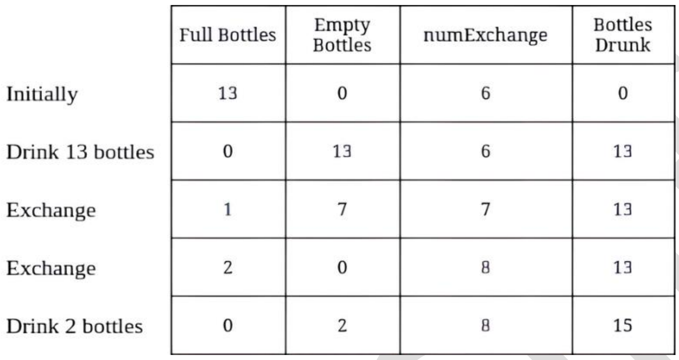
| | | |
|---|---|---|
| **13** | **Oneful Pairs** | Medium |

| | | |
|---|---|---|
| | **Problem statement** : Chef defines a pair of positive integers (*a*,*b*) to be a Oneful PairOneful Pair, if<br>a+*b*+(*a*⋅*b*)=111<br>For example, (1,55)(1,55) is a Oneful PairOneful Pair,<br>since 1+55+(1⋅55)=56+55=1111+55+(1⋅55)=56+55=111<br>.<br>But (1,56)(1,56) is not a Oneful PairOneful Pair,<br>since 1+56+(1⋅56)=57+56=113≠1111+56+(1⋅56)=57+56<br>=113 not equal to 111.<br>**Input 1:** 1 55<br>**Output 1:** Yes<br><br>**Explanation:**<br>(1,55)(1,55) is a Oneful PairOneful Pair,<br>since 1+55+(1⋅55)=56+55=1111+55+(1⋅55)=56+55=111.<br><br>**Input 2**: 1 56<br>**Output 2:** No<br><br>**Explanation:**<br>(1,56)(1,56) is not a Oneful PairOneful Pair,<br>since 1+56+(1⋅56)=57+56=113≠1111+56+(1⋅56)=57+56=113 not equal to 111<br>**Costraints**<br>• 1≤*a*,*b*≤1000 | |
| **14** | **Water Bottles II**<br><br>**Problem statement** : You are given two integers numBottles and numExchange.<br>numBottles represents the number of full water bottles that you initially have. In one operation, you can perform one of the following operations:<br>Drink any number of full water bottles turning them into empty bottles.<br>Exchange numExchange empty bottles with one full water bottle. Then, increase numExchange by one. | Medium |

Note that you cannot exchange multiple batches of empty bottles for the same value of numExchange. For example, if numBottles == 3 and numExchange == 1, you cannot exchange 3 empty water bottles for 3 full bottles.

Return the maximum number of water bottles you can drink.

**Example 1:**



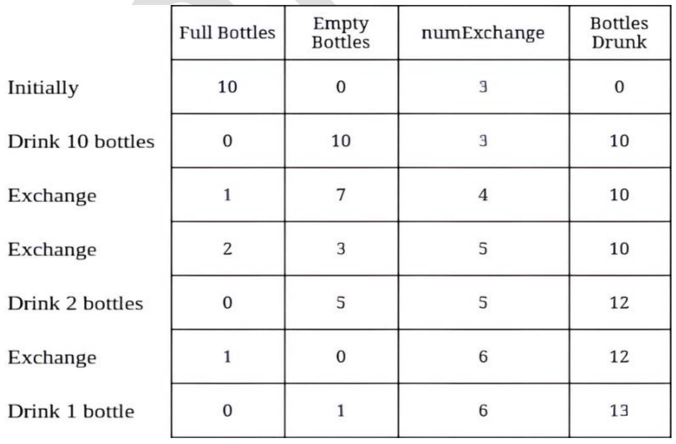
| | Full Bottles | Empty Bottles | numExchange | Bottles Drunk |
|---|---|---|---|---|
| Initially | 13 | 0 | 6 | 0 |
| Drink 13 bottles | 0 | 13 | 6 | 13 |
| Exchange | 1 | 7 | 7 | 13 |
| Exchange | 2 | 0 | 8 | 13 |
| Drink 2 bottles | 0 | 2 | 8 | 15 |

**Input:** numBottles = 13, numExchange = 6

**Output:** 15

**Explanation:** The table above shows the number of full water bottles, empty water bottles, the value of numExchange, and the number of bottles drunk.

**Example 2:**



| | Full Bottles | Empty Bottles | numExchange | Bottles Drunk |
|---|---|---|---|---|
| Initially | 10 | 0 | 3 | 0 |
| Drink 10 bottles | 0 | 10 | 3 | 10 |
| Exchange | 1 | 7 | 4 | 10 |
| Exchange | 2 | 3 | 5 | 10 |
| Drink 2 bottles | 0 | 5 | 5 | 12 |
| Exchange | 1 | 0 | 6 | 12 |
| Drink 1 bottle | 0 | 1 | 6 | 13 |

**Input:** numBottles = 10, numExchange = 3

**Output**: 13

**Explanation:** The table above shows the number of full water bottles, empty water bottles, the value of numExchange, and the number of bottles drunk.

| | | |
|---|---|---|
| | **Constraints:**<br>1 <= numBottles <= 100<br>1 <= numExchange <= 100 | |
| 15 | **Decode XORed Permutation**<br>**Problem Statement:** There is an integer array perm that is a permutation of the first n positive integers, where n is always odd. It was encoded into another integer array encoded of length n - 1, such that encoded[i] = perm[i] XOR perm[i + 1]. For example, if perm = [1,3,2], then encoded = [2,1].Given the encoded array, return *the original array* perm. It is guaranteed that the answer exists and is unique.<br>**Example 1:**<br>**Input:** encoded = [3,1]<br>**Output:** [1,2,3]<br>**Explanation:** If perm = [1,2,3], then encoded = [1 XOR 2,2 XOR 3] = [3,1]<br><br>**Example 2:**<br>**Input:** encoded = [6,5,4,6]<br>**Output:** [2,4,1,5,3]<br><br>**Constraints**:<br>• 3 <= n < 10^5<br>• n is odd.<br>encoded.length == n – 1 | Medium |
| 16 | **Single Number II**<br>**Problem Statement:** Given an integer array nums where every element appears three times except for one, which appears exactly once. Find the single element and return it.You must implement a solution with a linear runtime complexity and use only constant extra space.<br>**Example 1:**<br>**Input:** nums = [2,2,3,2]<br>**Output:** 3<br><br>**Example 2:** | Medium |

***Little practice is worth more than a ton of theory***

| | | |
|---|---|---|
| | **Input:** nums = [0,1,0,1,0,1,99]<br>**Output:** 99<br><br>**Constraints:**<br>• 1 <= nums.length <= 3 * 10^4<br>• -2^31 <= nums[i] <= 2^31 - 1<br>• Each element in nums appears exactly three times except for one element which appears once. | |
| **17** | **Total Hamming Distance**<br>**Problem Statement:** The Hamming distance between two integers is the number of positions at which the corresponding bits are different. Given an integer array nums, return the sum of Hamming distances between all the pairs of the integers in nums.<br>**Example 1:**<br>**Input:** nums = [4,14,2]<br>**Output:** 6<br>**Explanation:** In binary representation, the 4 is 0100, 14 is 1110, and 2 is 0010 (just<br>showing the four bits relevant in this case).<br>The answer will be:<br>HammingDistance(4, 14) + HammingDistance(4, 2) + HammingDistance(14, 2) = 2 + 2 + 2 = 6.<br><br>**Example 2:**<br>**Input:** nums = [4,14,4]<br>**Output:** 4<br><br>**Constraints:**<br>• 1 <= nums.length <= 10^4<br>• 0 <= nums[i] <= 10^9<br>• The answer for the given input will fit in a 32-bit integer. | Medium |
| **18** | **XOR Queries of a Subarray**<br>**Problem Statement:** You are given an array arr of positive integers. You are also given the array queries where queries[i] = [lefti, righti].For each query i compute the **XOR** of elements from lefti to | Medium |

righti (that is, arr[lefti] XOR arr[lefti + 1] XOR ... XOR arr[righti] ).Return an array answer where answer[i] is the answer to the ith query.

**Example 1:**

**Input:** arr = [1,3,4,8], queries = [[0,1],[1,2],[0,3],[3,3]]

**Output**: [2,7,14,8]

**Explanation:**

The binary representation of the elements in the array are:

1 = 0001

3 = 0011

4 = 0100

8 = 1000

The XOR values for queries are:

[0,1] = 1 xor 3 = 2

[1,2] = 3 xor 4 = 7

[0,3] = 1 xor 3 xor 4 xor 8 = 14

[3,3] = 8

**Example 2:**

**Input:** arr = [4,8,2,10], queries = [[2,3],[1,3],[0,0],[0,3]]

**Output:** [8,0,4,4]

**Constraints:**

- 1 <= arr.length, queries.length <= 3 * 10^4
- 1 <= arr[i] <= 10^9
- queries[i].length == 2
- 0 <= left <= right < arr.length

**19** **XOR DARE** Medium

**Problem Statement:** Ninja and his friends are playing a truth or dare game. Ninja has chosen dare, and his friends have given him a problem to solve. Now ninja is unable to solve the problem, so he asks for your help. In the problem, you are given an integer array ‘ARR’ consisting of ‘N’ elements and an integer number ‘K’.

You need to find the maximum possible strength among all the subsets of array ‘ARR’.

The strength of a set of elements is bitwise XOR of all the elements present in the set and 'K'.

Note: For an empty subset, You can consider the XOR of its elements as 0.

**Sample Input 1:**

2

4 3

5 4 1 2

3 8

1 2 8

**Sample Output 1:**

7

11

**Explanation of sample input 1:**

For the first test case:

The subset [3] will produce maximum strength of (3 XOR 4) = 7

For the second test case:

The subset [ 1, 2] will produce maximum strength of (1 XOR 2 XOR 8)= 11

**Sample Input 2:**

2

2

4 3

3 3 3 3

3 7

1 2 8

**Sample Output 2:**

3

15

**Constraints:**

1 <= N <= 1000

0 <= ARR[i] <=1000 for 'i' in range 0 to N-1

0 <= K <= 1000

| | | |
|---|---|---|
| **20** | **Count Total Setbits** | Medium |

| | | |
|---|---|---|
| | **Problem Statement:** You are given a number N. Find the total number of setbits in the numbers from 1 to N.<br>**Example 1:**<br>**Input:** N = 3<br>**Output:** 4<br>**Explanation:**<br>1 -> 01, 2 -> 10 and 3 -> 11.<br>So total 4 setbits.<br><br>**Example 2:**<br>**Input:** N = 4<br>**Output:** 5<br><br>**Constraints:**<br>1 ≤ N ≤ 10^6 | |
| **21** | **Reverse Bits**<br>**Problem Statement:** Reverse the bits of an 32 bit unsigned integer A.<br>**Example Input**<br>**Input 1:**<br>0<br>**Input 2:**<br>3<br>**Example Output**<br>**Output 1:**<br>0<br>**Output 2:**<br>3221225472 | Medium |
| **22** | **Majority Element**<br><br>**Problem Statement:** Given an array A of N elements. Find the majority element in the array. A majority element in an array A of size N is an element that appears strictly more than N/2 times in the array.<br>**Example 1:**<br>**Input:** | Medium |

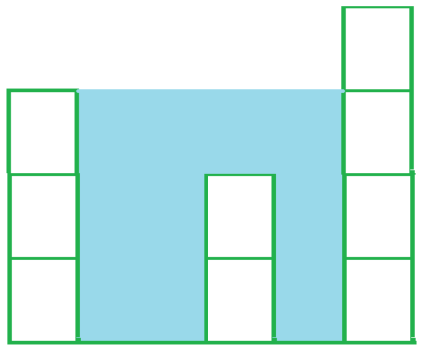
| | | |
|---|---|---|
| | N = 3 | |
| | A[] = {1,2,3} | |
| | **Output:** | |
| | -1 | |
| | **Explanation:** | |
| | Since, each element in {1,2,3} appears only once so there is no majority element. | |
| | | |
| | **Example 2:** | |
| | **Input**: | |
| | N = 5 | |
| | A[] = {3,1,3,3,2} | |
| | **Output:** | |
| | 3 | |
| | **Explanation:** | |
| | Since, 3 is present more than N/2 times, so it is the majority element. | |
| | **Constraints:** | |
| | 1 ≤ N ≤ 10^7 | |
| | 0 ≤ Ai ≤ 10^6 | |
| **23** | **Find Missing and Repeating** | Medium |
| | **Problem Statement:** Given an unsorted array Arr of size N of positive integers. One number 'A' from set {1, 2,....,N} is missing and one number 'B' occurs twice in array. Find these two numbers. | |
| | **Example 1:** | |
| | **Input:** | |
| | N = 2 | |
| | Arr[] = {2, 2} | |
| | **Output:** 2 1 | |
| | **Explanation:** Repeating number is 2 and smallest positive missing number is 1. | |
| | | |
| | **Example 2:** | |
| | **Input:** | |
| | N = 3 | |
| | Arr[] = {1, 3, 3} | |
| | **Output**: 3 2 | |

| | | |
|---|---|---|
| | **Explanation**: Repeating number is 3 and smallest positive missing number is 2.<br><br>**Constraints:**<br>2 ≤ N ≤ 10^5<br>1 ≤ Arr[i] ≤ N | |
| **24** | **Kadane's Algorithm**<br>**Problem Statement:** Given an array Arr[] of N integers. Find the contiguous sub-array (containing at least one number) which has the maximum sum and return its sum.<br><br>**Example 1:**<br>**Input:**<br>N = 5<br>Arr[] = {1,2,3,-2,5}<br>**Output:**<br>9<br>**Explanation:**<br>Max subarray sum is 9 of elements (1, 2, 3, -2, 5) which is a contiguous subarray.<br><br>**Example 2:**<br>**Input:**<br>N = 4<br>Arr[] = {-1,-2,-3,-4}<br>**Output:**<br>-1<br>**Explanation:**<br>Max subarray sum is -1 of element (-1)<br><br>**Constraints:**<br>1 ≤ N ≤ 10^6<br>-107 ≤ A[i] ≤ 10^7 | Medium |
| **25** | **Sort an array of 0s, 1s and 2s** | Medium |

| | | |
|---|---|---|
| | **Problem Statement:** Given an array of size N containing only 0s, 1s, and 2s; sort the array in ascending order.<br><br>**Example 1:**<br>**Input:**<br>N = 5<br>arr[]= {0 2 1 2 0}<br>**Output:**<br>0 0 1 2 2<br>**Explanation:**<br>0s 1s and 2s are segregated into ascending order.<br><br>**Example 2:**<br>**Input:**<br>N = 3<br>arr[] = {0 1 0}<br>**Output:**<br>0 0 1<br>**Explanation:**<br>0s 1s and 2s are segregated into ascending order.<br><br>**Constraints:**<br>1 <= N <= 10^6<br>0 <= A[i] <= 2 | |
| **26** | **Count Primes**<br>**Problem Statement:** Given an integer n, return the number of prime numbers that are strictly less than n.<br><br>**Example 1:**<br>**Input:** n = 10<br>**Output**: 4<br>**Explanation:** There are 4 prime numbers less than 10, they are 2, 3, 5, 7.<br><br>**Example 2:**<br>**Input:** n = 0 | Medium |

| | | |
|---|---|---|
| | **Output:** 0<br><br>**Constraints:**<br>• 0 <= n <= 5 * 10^6 | |
| 27 | **Watering Plants**<br>**Problem Statement:** You want to water n plants in your garden with a watering can. The plants are arranged in a row and are labeled from 0 to n - 1 from left to right where the ith plant is located at x = i. There is a river at x = -1 that you can refill your watering can at.<br>Each plant needs a specific amount of water. You will water the plants in the following way:<br>• Water the plants in order from left to right.<br>• After watering the current plant, if you do not have enough water to completely water the next plant, return to the river to fully refill the watering can.<br>• You cannot refill the watering can early.<br>You are initially at the river (i.e., x = -1). It takes one step to move one unit on the x-axis.<br>Given a 0-indexed integer array plants of n integers, where plants[i] is the amount of water the ith plant needs, and an integer capacity representing the watering can capacity, return the number of steps needed to water all the plants.<br>**Example 1:**<br>**Input:** plants = [2,2,3,3], capacity = 5<br>**Output:** 14<br>**Explanation:**<br>Start at the river with a full watering can:<br>- Walk to plant 0 (1 step) and water it. Watering can has 3 units of water.<br>- Walk to plant 1 (1 step) and water it. Watering can has 1 unit of water.<br>- Since you cannot completely water plant 2, walk back to the river to refill (2 steps).<br>- Walk to plant 2 (3 steps) and water it. Watering can has 2 units of water. | Medium |

| | | |
|---|---|---|
| | - Since you cannot completely water plant 3, walk back to the river to refill (3 steps).<br>- Walk to plant 3 (4 steps) and water it.<br>Steps needed = 1 + 1 + 2 + 3 + 3 + 4 = 14.<br><br>**Exam**ple 2:<br>**Input:** plants = [1,1,1,4,2,3], capacity = 4<br>**Output:** 30<br>**Explanation:**<br>Start at the river with a full watering can:<br>- Water plants 0, 1, and 2 (3 steps). Return to river (3 steps).<br>- Water plant 3 (4 steps). Return to river (4 steps).<br>- Water plant 4 (5 steps). Return to river (5 steps).<br>- Water plant 5 (6 steps).<br>Steps needed = 3 + 3 + 4 + 4 + 5 + 5 + 6 = 30.<br><br>**Constraints:**<br>● n == plants.length<br>● 1 <= n <= 1000<br>● 1 <= plants[i] <= 10^6<br>● max(plants[i]) <= capacity <= 10^9 | |
| **28** | **Trapping Rain Water**<br>**Problem Statement:** Given an array arr[] of N non-negative integers representing the height of blocks. If width of each block is 1, compute how much water can be trapped between the blocks during the rainy season.<br>**Example 1:**<br>**Input:**<br>N = 6<br>arr[] = {3,0,0,2,0,4}<br>**Output:**<br>10<br>**Explanation:** | Medium |

| | | |
|---|---|---|
| | <br>Bars for input {3, 0, 0, 2, 0, 4}<br>Total trapped water = 3 + 3 + 1 + 3 = 10<br><br>**Example 2:**<br>**Input**:<br>N = 4<br>arr[] = {7,4,0,9}<br>**Output:**<br>10<br>**Explanation:**<br>Water trapped by above block of height 4 is 3 units and above block of height 0 is 7 units. So, the total unit of water trapped is 10 units.<br><br>**Example 3:**<br>**Input:**<br>N = 3<br>arr[] = {6,9,9}<br>**Output:**<br>0<br>**Explanation:**<br>No water will be trapped.<br><br>**Constraints:**<br>3 < N < 10^6<br>0 < Ai < 10^8 | |
| **29** | **Minimize Maximum Pair Sum in Array**<br>**Problem Statement :** The pair sum of a pair (a,b) is equal to a + b. The maximum pair sum is the largest pair sum in a list of pairs. | Medium |

| | | |
|---|---|---|
| | • For example, if we have pairs (1,5), (2,3), and (4,4), the maximum pair sum would be max(1+5, 2+3, 4+4) = max(6, 5, 8) = 8.<br>Given an array nums of even length n, pair up the elements of nums into n / 2 pairs such that:<br>• Each element of nums is in exactly one pair, and<br>• The maximum pair sum is minimized.<br>Return the minimized maximum pair sum after optimally pairing up the elements.<br><br>**Example 1:**<br>**Input:** nums = [3,5,2,3]<br>**Output:** 7<br>**Explanation:** The elements can be paired up into pairs (3,3) and (5,2).<br>The maximum pair sum is max(3+3, 5+2) = max(6, 7) = 7.<br><br>**Example 2:**<br>**Input:** nums = [3,5,4,2,4,6]<br>**Output:** 8<br>**Explanation:** The elements can be paired up into pairs (3,5), (4,4), and (6,2).<br>The maximum pair sum is max(3+5, 4+4, 6+2) = max(8, 8, 8) = 8.<br><br>**Constraints:**<br>• n == nums.length<br>• 2 <= n <= 10^5<br>• n is even.<br>• 1 <= nums[i] <= 10^5 | |
| **30** | **Minimum Absolute Sum Difference**<br>**Problem Statement:** You are given two positive integer arrays nums1 and nums2, both of length n.The absolute sum difference of arrays nums1 and nums2 is defined as the sum of \|nums1[i] - nums2[i]\| for each 0 <= i < n (0-indexed).You can replace at most one element of nums1 with any other element in nums1 to minimize the absolute sum difference.Return the minimum absolute sum | |

difference after replacing at most one element in the array nums1.

Since the answer may be large, return it modulo 109 + 7.

|x| is defined as:

- x if x >= 0, or
- -x if x < 0.

**Example 1:**

**Input:** nums1 = [1,7,5], nums2 = [2,3,5]

**Output:** 3

**Explanation:** There are two possible optimal solutions:

- Replace the second element with the first: [1,7,5] => [1,1,5], or

- Replace the second element with the third: [1,7,5] => [1,5,5].

Both will yield an absolute sum difference of |1-2| + (|1-3| or |5-3|) + |5-5| = 3.

**Example 2:**

**Input:** nums1 = [2,4,6,8,10], nums2 = [2,4,6,8,10]

**Output:** 0

**Explanation:** nums1 is equal to nums2 so no replacement is needed. This will result in an absolute sum difference of 0.

**Constraints:**

- n == nums1.length
- n == nums2.length
- 1<= n <= 10^5
- 1 <= nums1[i], nums2[i] <= 10^5