Agenda:

- Contiguous memory allocation in Arrays
- Bucketing Technique:
  - Smallest Positive missing number
  - Boolean Matrix Question
- Kadane's Algorithm
- Max Circular Subarray Sum **(Just discuss the approach)**
- Binary Search:
  - Find the first and last positions of an element in a sorted array
  - Floor square root

Smallest Positive missing number

You are given an array **arr**[] of **N** integers including 0. The task is to find the smallest positive number missing from the array.

Example 1:

```
Input:
N = 5
arr[] = {1,2,3,4,5}
Output: 6
Explanation: Smallest positive missing
number is 6.
```

Example 2:

```
Input:
N = 5
arr[] = {0,-10,1,3,-20}
Output: 2
Explanation: Smallest positive missing
number is 2.
```

A Boolean Matrix Question

Given a matrix, A of size M x N of 0s and 1s. If an element is 0, set its entire row and column to 0.

```
Input 1:
    [   [1, 0, 1],
        [1, 1, 1],
        [1, 1, 1]   ]


Output 1:
    [   [0, 0, 0],
        [1, 0, 1],
        [1, 0, 1]   ]


Input 2:
    [   [1, 0, 1],
        [1, 1, 1],
        [1, 0, 1]   ]


Output 2:
    [   [0, 0, 0],
        [1, 0, 1],
        [0, 0, 0]   ]
```

Kadane's Algorithm

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.

```
Input:
N = 5
Arr[] = {1,2,3,-2,5}
Output:
9
Explanation:
Max subarray sum is 9
of elements (1, 2, 3, -2, 5) which
is a contiguous subarray.
```

Example 2:

```
Input:
N = 4
Arr[] = {-1,-2,-3,-4}
Output:
-1
Explanation:
Max subarray sum is -1
of element (-1)
```

Find first and last positions of an element in a sorted array

Given a sorted array with possibly duplicate elements, the task is to find indexes of first and last occurrences of an element x in the given array.

```
Input : arr[] = {1, 3, 5, 5, 5, 5, 67, 123, 125}
        x = 5
Output : First Occurrence = 2
         Last Occurrence = 5

Input : arr[] = {1, 3, 5, 5, 5, 5, 7, 123, 125 }
        x = 7
Output : First Occurrence = 6
         Last Occurrence = 6
```

Floor square root without using sqrt() function

Given a number **N**, the task is to find the floor square root of the number N without using the built-in square root function. **Floor square root** of a number is the greatest whole number which is less than or equal to its square root.

Input: *N = 25*
Output: *5*
Explanation:
Square root of 25 = 5. Therefore 5 is the greatest whole number less than equal to Squar e root of 25.

Input: *N = 30*
Output: *5*
Explanation:
Square root of 25 = 5.47
Therefore 5 is the greatest whole number less than equal to Square root of 25 (5.47)