# Searching – 2

Given an array with repeated numbers in sorted form. find the **starting** and **ending** index of a target.

A = [1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4]; target = 2. This is a Linear Search Code.

```
A = [1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4]
target = 2
start = -1
end = -1

for i in range(len(A)):
    if A[i] == target:
        if start == -1:
            start = i
        elif A[i+1] != target:
            end = i

print (start, end)
```

arr = [1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3], this is a sorted array.

0 1 2 3 4 5 6 7 8 9 10

For a Binary Search, left = 0 and right = 10 and mid = (0+10)>>1 = 5. We have to find the left most value of 2 which would lie to the left side of the array from the mid.

Now, the mid value is 2. It is also the target value so we can assign it to “ans”. As we are trying to find the lower bound, the right value will come to mid – 1 = 4.

So, left = 0, right = mid – 1 = 4.

Now, mid = (0 + 4) >> 1 = 2. So, the mid value now is at 2nd index and its value is 1. Our target would lie towards right. So the solution space will move to mid + 1 = left. The mid value at this point is 1, so it cannot be the “ans” or target we are looking for.

In the next step, with left = 3, right = 4 the mid value would be 3. This could be our “ans” so let us save it to “ans” and update it to 3. Our solution space, in which left = 3 and right = 4 since the answer might lie towards the left side, our left = 3 and the right = 2.

Left > Right == break

**CODE:**

```
def lowerBound(A, target):
    n = len(A)

    left = 0
    right = n - 1

    ans = -1

    while left <= right:
        mid = (left + right) >> 1 # or // 2

        if A[mid] == target:
            ans = mid
            right = mid - 1
        elif A[mid] > target:
            right = mid - 1
        elif A[mid] < target:
            left = mid + 1
    return ans

def upperBound(A, target):
    n = len(A)

    left = 0
    right = n - 1

    ans = -1

    while left <= right:
        mid = (left + right)>>1

        if A[mid] == target:
            ans = mid
            left = mid + 1
        elif A[mid] > target:
            left = mid + 1
        elif A[mid] < target:
            right = mid - 1
    return ans

if __name__ == "__main__":
    A = [1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3]
    print(lowerBound(A,2), upperBound(A,2))
```

**Ex: 02**

```python
def lowerBound(A, target):
    n = len(A)

    left = 0
    right = n - 1

    ans = -1

    while left <= right:
        mid = (left + right) >> 1 # or // 2

        if A[mid] == target:
            ans = mid
            right = mid - 1
        elif A[mid] > target:
            right = mid - 1
        elif A[mid] < target:
            left = mid + 1
    return ans

def upperBound(A, target):
    n = len(A)

    left = 0
    right = n - 1

    ans = -1

    while left <= right:
        mid = (left + right)>>1

        if A[mid] == target:
            ans = mid
            left = mid + 1
        elif A[mid] > target:
            right = mid - 1
        elif A[mid] < target:
            left = mid + 1
    return ans

if __name__ == "__main__":
    A = [1, 1, 1, 1, 2, 3, 3, 4, 5, 5]
    print(lowerBound(A,3), upperBound(A,3))
```

Q) Given in an array, what is the frequency of a number.

Frequency of a number using lower & upper bound method would be,

Upper_Bound - Lower_Bound + 1

```
def lowerBound(A, target):
    n = len(A)
    left = 0
    right = n - 1
    ans = -1

    while left <= right:
        mid = (left + right) >> 1 # or // 2

        if A[mid] == target:
            ans = mid
            right = mid - 1
        elif A[mid] > target:
            right = mid - 1
        elif A[mid] < target:
            left = mid + 1
    return ans

def upperBound(A, target):
    n = len(A)
    left = 0
    right = n - 1
    ans = -1

    while left <= right:
        mid = (left + right)>>1

        if A[mid] == target:
            ans = mid
            left = mid + 1
        elif A[mid] > target:
            right = mid - 1
        elif A[mid] < target:
            left = mid + 1
    return ans

if __name__ == "__main__":
    A = [1, 1, 1, 1, 2, 3, 3, 4, 5, 5]
    x = 1
    print (lowerBound (A,3), upperBound (A,3))
    print (upperBound (A, x) - lowerBound (A, x) + 1)
```

Q) Given, peaks and troughs in a graph, represented as array. Find any one of the peak elements in it. Ex: A = [2, 5, 3, 7, 9, 13, 8]

## **Peak:**

In an array element is a peak if it is NOT smaller than its neighbours. For corner elements, we need to consider only one neighbour.

In the give case, 13 is a peak element, because 9 & 8 are less than 13.

In [1, 3, 5, 8, 10]; 10 is the peak because its neighbor 8 is less than 10.

In [5, 2, 3, 1, 7, 6, 15] peak values are 5, 3, 7 & 15

A = [5, 2, 3, 4, 5]

At i = 0, val = 5, i+1 val = 2, then you peak element is 5.

If i = n-1, val = 5, n-1-i = 4, then your peak element is 5.

If we have A = [5, 2, 13, 7, 6]

At i = 3, val = 13 so, if i+1 and i-1 < i then peak element is 13.

**162. Find Peak Element**

```
class Solution:
    def findPeakElement(self, nums):
        n = len(nums)
        for i in range(n):
            if i == 0:
                if nums[i] > nums[i+1]:
                    return i
            elif i == n - 1:
                if nums[i] > nums[i-1]:
                    return i
            else:
                if nums[i] > nums [i-1] and nums[i] > nums[i+1]:
                    return i

sol = Solution ()
print(sol.findPeakElement([1,1,1,12,3]))
```

**OUTPUT**: **3**

For edge case, A = [1], then we need to add

if len(nums) == 1:
    return 0

```
class Solution:
    def findPeakElement(self, nums):
        n = len(nums)
        if len(nums) == 1:
            return 0

        for i in range(n):
            if i == 0:
                if nums[i] > nums[i+1]:
                    return i
            elif i == n - 1:
                if nums[i] > nums[i-1]:
                    return i
            else:
                if nums[i] > nums [i-1] and nums[i] > nums[i+1]:
                    return i

sol = Solution()
print(sol.findPeakElement([1]))
```

OUTPUT: **0**

To solve this using Binary Search, let's see a graph.

In [2,4,3,5,3,7,4]. We can say the peak element is 5.

In [2,3,5,8,1], left = 0, right = 4, mid = 2. Now to the right we have 8, so the left = 3, right = 4 and mid = 3. Now, at 3rd index the value is 8 which is greater than i+1 and i-1. So, 8 is the peak element.

```
def Solve(A):
    left = 0
    right = len(A) - 1
    while left <= right:
        mid = (left + right) >> 1
        if mid == 0 and A[mid + 1] < A[mid]:
            return mid
        if mid == len(A) - 1 and A[mid] > A[mid - 1]:
            return mid
```

```
        if A[mid] >= A[mid - 1] and A[mid] > A[mid + 1]:
            return mid
        elif A[mid + 1] > A[mid]:
            left = mid + 1
        elif A[mid - 1] > A[mid]:
            right = mid - 1
    return -1
```