

Binary Search

Mustafa Muhammad

10th October 2021

List of Questions:

1. Binary search
2. Binary search on reverse sorted array
3. Order not known search
4. First and last occurrence of an element
5. Count of an element in a sorted array
6. Find floor of an element in a sorted array
7. Ceil of an element in a sorted array
8. Next alphabetical element
9. Find position of an element in an infinite sorted array
10. Index of first in sorted infinite array (binary)
11. Minimum difference element in a sorted array
12. Peak element
13. Find maximum element in a bitonic array
14. Search an element in a bitonic array
15. Search in row and column wise sorted array
16. Allocate minimum number of pages

1 Identification

If you see that the given array is sorted, think binary search.

```
def searchInSorted(self, arr, N, K):
    left = 0
    right = len(arr)-1

    while(left <= right):
        mid = int((left + (right-left))/2)

        if arr[mid] == K:
            return 1

        elif arr[mid] < K:
            left = mid+1

        else:
            right = mid -1

    return -1
```

2 Binary search - Reverse Sorted Array

```
def binary_reverse_search(arr, target):
    left = 0
    right = len(arr)-1

    while(left <= right):
        mid = int((left+(right-left))/2)

        if arr[mid] < target:
            right = mid - 1

        elif arr[mid] > target:
            left = mid + 1

        else:
            return mid

    return -1
```

3 Order Not Known

We can check the difference between the first and last element. If the first element is smaller than the last, we do normal binary search, else we simply reverse the order.

4 First and Last Occurence of An element

We check the first occurrence of an element by finding the occurrence of an element and then proceeding to go left. Similarly we find the last occurrence of an element by going right once found.

```
def find(arr,n,x):
    # code here
    def first_occurence(arr, x):
        left = 0
        right = len(arr)-1
        res = -1
        while(left <= right):
            mid = int(left+ (right-left)/2)
            if arr[mid] == x:
                res = mid
                right = mid - 1
            elif arr[mid] < x:
                left = mid+1
            elif arr[mid] > x:
                right = mid - 1
        return res
    def last_occurence(arr, x):
        left = 0
        right = len(arr)-1
        res = -1
        while(left <= right):
            mid = int(left+ (right-left)/2)
            if arr[mid] == x:
                res = mid
                left = mid + 1
            elif arr[mid] < x:
                left = mid+1
            elif arr[mid] > x:
                right = mid - 1
        return res
    results = []
    results.append(first_occurence(arr, x))
    results.append(last_occurence(arr, x))
    return results
```

5 Count of an element in a sorted array

We find the first occurrence of an element, then we find the last occurrence of an element. Formula for count = Last - FirstOccurrence + 1 if neither element is -1, else we return 0.

6 Number of times a sorted array is rotated

Input: nums = [3,4,5,1,2]

Output: 1

Explanation: The original array was [1,2,3,4,5] rotated 3 times. We know this because the minimum elements index is 3. We perform binary search to find the smallest element. Its index in the number of rotations.

```
class Solution:
    def findMin(self, nums: List[int]) -> int:
        # minimum in rotated sorted array
        left = 0
        right = len(nums)-1
        n = len(nums)

        while left <= right:
            mid = int((left + (right-left)/2)

            # To avoid index out of bounds
            next_ = (mid+1)%n
            prev = (mid-1+n)%n

            # Minimum element will be smaller than both its neighbors.
            if nums[mid]<= nums[next_] and nums[mid] <= nums[prev]:
                return nums[mid]

            #We go towards the unsorted array (location of min element)

            elif nums[0] <= nums[mid]:
                left = mid+1

            elif nums[mid] <= nums[-1]:
                right = mid -1

        return nums[0]
```

7 Find an element in a rotated sorted array

Building upon the previous question on rotation. We find the minimum index and divide the array in half and apply binary search on both sides.

```
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        # find minimum index , then binary search on both sides
        def find_index(nums):
            left = 0
            right = len(nums)-1
            n = len(nums)
            while left <= right:
                mid = int(left + (right-left)/2)
                next_ = (mid+1)%n
                prev = (mid-1+n)%n

                if nums[mid]<=nums[next_] and nums[mid]<=nums[prev]:
                    return mid

                elif nums[0] <= nums[mid]:
                    left = mid + 1

                elif nums[mid] <= nums[-1]:
                    right = mid -1

            return -1

        def binary_search(nums, target):
            left = 0
            right = len(nums)-1
            while(left <= right):
                mid = int(left + (right-left)/2)
                if nums[mid] == target:
                    return mid
                elif nums[mid] < target:
                    left = mid + 1
                elif nums[mid] > target:
                    right = mid -1
            return -1

        min_index = find_index(nums)
        left_arr = binary_search(nums[0:min_index], target)
        right_arr = binary_search(nums[min_index:], target)

        return right_arr+len(nums[0:min_index]) if right_arr != -1 else left_arr
```

8 Searching in a nearly sorted array

```
class Solution:
    def binary_search_on_nearly_sorted_array(arr, target):
        left = 0
        right = len(arr)-1

        while left <= right:
            mid = int(left + (right-left)/2)
            if arr[mid] == target:
                return mid
            elif mid-1 >= left and arr[mid-1] == target:
                return mid-1
            elif mid+1 <= right and arr[mid+1] == target:
                return mid+1
            elif arr[mid] < target:
                left = mid+2
            elif arr[mid] > target:
                right = mid -2
        return -1
```

9 Find floor of an element in a sorted array

Input:

N = 7, x = 5

arr[] = 1,2,8,10,11,12,19

Output: 1

Explanation: Largest Number less than 5 is 2 (i.e K = 2), whose index is 1(0-based indexing).

```
class Solution:
    def findFloor(self, arr, N, X):
        #Your code here
        left = 0
        right = len(arr)-1
        res = -1
        while left <= right:
            mid = int((left + (right-left)/2))
            if arr[mid] == X:
                return mid
            if arr[mid] < X:
                res = mid
                left = mid + 1
            elif arr[mid] > X:
                right = mid - 1
        return res
```


10 Find ceil of an element in a sorted array

For example, let the input array be 1, 2, 8, 10, 10, 12, 19

For x = 0: floor doesn't exist in array, ceil = 1

For x = 1: floor = 1, ceil = 1

For x = 5: floor = 2, ceil = 8

For x = 20: floor = 19, ceil doesn't exist in array

```
class Solution:
    def findFloor(self, arr, N, X):
        #Your code here
        left = 0
        right = len(arr)-1
        res = -1
        while left <= right:
            mid = int(left + (right-left)/2)
            if arr[mid] == X:
                return mid
            if arr[mid] < X:
                left = mid + 1
            elif arr[mid] > X:
                res = mid
                right = mid - 1
        return res
```

11 Next alphabetical element

Given an array of letters sorted in ascending order, find the smallest letter in the the array which is greater than a given key letter. Problem is very similar to ceil, as in this particular problem we just find the next possible alphabet.

Changes: 1. Alphabets instead of numbers 2. Even if we find the target alphabet, we return the next alphabet.

```
if arr[mid] == key:
    left = mid+1
```

12 Find position of an element in an infinite sorted array

```
class Solution:
    def binary_search_on_inf(arr, target):
        start = 0
        end = 1

        # Moving up the end index and making start the previous end
        while arr[end] < target:
            start = end
            end = end*2

        while start <= end:
            mid = int((start + (end-start))/2)
            if arr[mid] == target:
                return mid
            elif arr[mid] < target:
                start = mid + 1
            elif arr[mid] > target:
                end = mid - 1
        return -1
```

13 Index of First "1" in an Infinite 0/1 Sorted Arr

Combination of binary search on an infinite array and first occurrence of an element.

```
class Solution:
    def binary_search_on_inf(arr):
        start = 0
        end = 1

        # Moving up the end index and making start the previous end
        while arr[end] != 0:
            start = end
            end = end*2

        res = -1
        while start <= end:
            mid = int((start + (end-start)/2))
            if arr[mid] == 1:
                res = mid
                end = mid - 1
            elif arr[mid] < 1:
                start = mid + 1
            elif arr[mid] > 1:
                end = mid - 1
        return res
```

14 Minimum Difference Element in a Sorted Array

We run binary search normally as we do, in the end, left and right will be in between the key value, all we have to do is return the value with the min. difference between the two.

Another way to think about this is to find the ceil and floor of the array using binary search.

```
class Solution:
    def binary_search(arr, target):
        start = 0
        end = len(arr) - 1
        while start <= end:
            mid = int((start + end) / 2)
            if arr[mid] == target:
                return mid
            elif arr[mid] < target:
                start = mid + 1
            elif arr[mid] > target:
                end = mid - 1
        return min(abs(target - arr[start]), abs(target - arr[end]))
```

15 Binary search on answer

There are some cases where the array is not sorted but we can still apply binary search.

We need to find a criteria inorder to split the array and find our answer.

We also need to decide how to move between the array.

Best example of this Concept is "Peak Element"