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 occurent 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 infite 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 occurence of an element by finding the occurence of an element and then proceeding to go left. Similarly we find the last occurence 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 occurence of an element, then we find the last occurence of an element. Formula for count = Last - FirstOccurence + 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(left + (right-left)/2)
            if arr[mid] == target:
                return mid
            elif arr[mid] < target:
                left = mid + 1
            elif arr[mid] > target:
                right = 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 occurence 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(left + (right-left)/2)
            if arr[mid] == target:
                res = mid
                right = mid - 1
            elif arr[mid] < target:
                left = mid + 1
            elif arr[mid] > target:
                right = 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 = 1
        while start <= end:
            mid = int(left + (right-left)/2)
            if arr[mid] == target:
                return mid
            elif arr[mid] < target:
                left = mid + 1
            elif arr[mid] > target:
                right = mid -1
        return min(abs(target - arr[left]), abs(target-arr[right]))
```

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"

16 Peak Element

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

Output: 2

Explanation: 3 is a peak element and your function should return the index number 2.

We use binary search. We find the peak with the condition if nums[mid] ¿ nums[mid+1] and nums[mid] ¿ nums[mid-1]

We move in the direction of the larger value as there is a higher chance of encountering our result in that area.

```
class Solution:
    def findPeakElement(self, nums: List[int]) -> int:
        left = 0
        right = len(nums)-1
        if len(nums) == 1: return 0
        while left <= right:
            mid = int(left + (right-left)/2)
            if mid != 0 and mid != len(nums)-1:
                if nums[mid] > nums[mid-1] and nums[mid] > nums[mid+1]:
                    return mid
                elif nums[mid] < nums[mid+1]:
                    left = mid + 1
                elif nums[mid] < nums[mid-1]:
                    right = mid - 1
            if mid == 0:
                if nums[mid]>nums[1]:
                    return mid
                else:
                    return 1
            if mid == len(nums)-1:
                if nums[mid] > nums[mid-1]:
                    return mid
                else:
                    return mid-1
        return -1
```

17 Find maximum in a bitonic array

Given a bitonic array find the maximum value of the array. An array is said to be bitonic if it has an increasing sequence of integers followed immediately by a decreasing sequence of integers.

Hence this problem is the same as finding the peak element.

Input: 1 4 8 3 2

Output: 8

18 Search an element in a bitonic array

This problem is a mix of both peak element and binary search. All the elements before the peak are sorted in ascending order, while the one's after the peak are in descending order. Hence we just need to find the peak index and after excluding it, we run binary search on the two arrays, if the key is not the peak.

```
class Solution:
    def findPeakElement(self, nums: List[int]) -> int:
        left = 0
        right = len(nums)-1
        if len(nums) == 1: return 0
        while left <= right:
            mid = int(left + (right-left)/2)
            if mid != 0 and mid != len(nums)-1:
                if nums[mid] > nums[mid-1] and nums[mid] > nums[mid+1]:
                    return mid
                elif nums[mid] < nums[mid+1]:
                    left = mid + 1
                elif nums[mid] < nums[mid-1]:
                    right = mid - 1
            if mid == 0:
                if nums[mid]>nums[1]:
                    return mid
                else:
                    return 1
            if mid == len(nums)-1:
                if nums[mid] > nums[mid-1]:
                    return mid
                else:
                    return mid-1
        return -1

    def binary_search_asc(self, arr, target):
        start = 0
        end = len(arr)-1
        while start <= end:
            mid = int(left + (right-left)/2)
            if arr[mid] == target:
                return mid
            elif arr[mid] < target:
                left = mid+1
            elif arr[mid] > target:
                right = mid-1
        return -1

    def binary_search_desc(self, arr, target):
        start = 0
        end = len(arr)-1
        while start <= end:
            mid = int(left + (right-left)/2)
            if arr[mid] == target:
                return mid
            elif arr[mid] < target:
                right = mid -1

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

19 Search in row wise and column wise sorted array

Write an efficient algorithm that searches for a target value in an m x n integer matrix. The matrix has the following properties:

Integers in each row are sorted in ascending from left to right. Integers in each column are sorted in ascending from top to bottom.

Input: matrix = [[1,4,7,11,15],[2,5,8,12,19],[3,6,9,16,22],[10,13,14,17,24] ,[18,21,23,26,30]], target = 5

Output: true

```
class Solution:
    def searchMatrix(self, matrix: List[List[int]], target: int) -> bool:
        i = 0
        j = len(matrix[0])-1
        while i>=0 and j>=0 and i<= len(matrix)-1 and j<= len(matrix[0])-1:
            if matrix[i][j] == target:
                return True
            elif matrix[i][j] > target:
                j-=1
            elif matrix[i][j] < target:
                i+=1
        return False
```

20 Allocate minimum number of pages

You are given N number of books. Every ith book has Ai number of pages. You have to allocate contagious books to M number of students. There can be many ways or permutations to do so. In each permutation, one of the M students will be allocated the maximum number of pages. Out of all these permutations, the task is to find that particular permutation in which the maximum number of pages allocated to a student is minimum of those in all the other permutations and print this minimum value.

Each book will be allocated to exactly one student. Each student has to be allocated at least one book.

Note: Return -1 if a valid assignment is not possible, and allotment should be in contiguous order (see the explanation for better understanding).

Input:

N = 4

A[] = 12,34,67,90

M = 2

Output:

113 Explanation:

Allocation can be done in following ways: 12 and 34, 67, 90 Maximum Pages = 191

12, 34 and 67, 90 Maximum Pages = 157

12, 34, 67 and 90 Maximum Pages =113

Therefore, the minimum of these cases is

113, which is selected as the output.

Main challenge is to create the is_valid function, rest of the problem is trivial.

```
class Solution:
    def findPages(self,A, N, M):
        start = max(A)
        end = sum(A)
        res = -1

        def is_valid(arr, k, mid):
            student = 1
            curr_sum = 0
            for i in range(0, len(arr)):
                curr_sum += arr[i]
                if curr_sum > mid:
                    student+= 1
                    curr_sum = arr[i]

            if student > k:
                return False
            return True

        while start <= end:
            mid = int(start + (end-start)/2)
            if is_valid(A, M, mid):
                res = mid
                end = mid - 1
            else:
                start = mid + 1
        return res
```