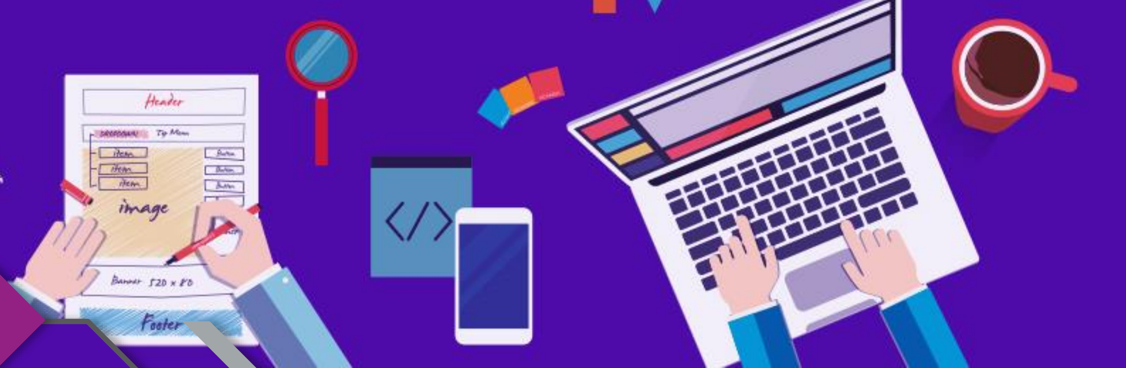

Data Structures & Algorithms

Binary Search

B.Bhuvaneswaran, AP (SG) / CSE

9791519152

bhuvaneswaran@rajalakshmi.edu.in

RAJALAKSHMI ENGINEERING COLLEGE
An AUTONOMOUS Institution
Affiliated to ANNA UNIVERSITY, Chennai

Binary Search

- Binary search is a search algorithm that runs in O(logn) in the worst case, where n is the size of the search space.
- For binary search to work, your search space usually needs to be sorted.
- Binary search trees, which we looked at in the trees and graphs chapter, are based on binary search.

Binary Search

- If you have a sorted array arr and an element x, then in O(logn) time and O(1) space, binary search can:
  - Find the index of x if it is in arr
  - Find the first or the last index in which x can be inserted to maintain being sorted otherwise

Binary Search

- Let's say that there is a sorted integer array arr, and you know that the number x is in it, but you don't know at what index.
- You want to find the position of x. Start by checking the element in the middle of arr. If this element is too small, then we know every element in the left half will also be too small, since the array is sorted.
- Similarly, if the element is too large, then every element in the right half will also be too large.
- We can discard the half that can't contain x, and then repeat the process on the other half.
- We continue this process of cutting the array in half until we find x.

Implementation

1. Declare left = 0 and right = arr.length - 1. These variables represent the inclusive bounds of the current search space at any given time. Initially, we consider the entire array.
2. While left <= right:
    - Calculate the middle of the current search space, mid = (left + right) // 2 (floor division)
    - Check arr[mid]. There are 3 possibilities:
        - If arr[mid] = x, then the element has been found, return.
        - If arr[mid] > x, then halve the search space by doing right = mid - 1.
        - If arr[mid] < x, then halve the search space by doing left = mid + 1.
3. If you get to this point without arr[mid] = x, then the search was unsuccessful. The left pointer will be at the index where x would need to be inserted to maintain arr being sorted.

Implementation

- Because the search space is halved at every iteration, binary search's worst-case time complexity is O(logn).
- This makes it an extremely powerful algorithm as logarithmic time is very fast compared to linear time.

Note

- You have probably used binary search in real life without even realizing it.
- For example, if you have ever looked up a word in a dictionary, then you probably flipped to about the middle, looked at the first letter of the words on the page you flipped to, and then either checked the left or right half depending on the first letter of the word you were looking for.

Implementation Template

```
public int binarySearch(int[] arr, int target) {
    int left = 0;
    int right = arr.length - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == target) {
            // do something
            return mid;
        }
        if (arr[mid] > target) {
            right = mid - 1;
        } else {
            left = mid + 1;
        }
    }

    // target is not in arr, but left is at the insertion point
    return left;
}
```

Note

- In the Java and C++ implementations, instead of doing (left + right) / 2, we do left + (right - left) / 2 to avoid overflow.
- The equations are equivalent, but the second one makes sure that no value greater than right is ever stored.
- In Python and JavaScript, numbers don't overflow (or at least, the limit is ridiculously huge), so we are fine with having left + right potentially being large.

On arrays

- Binary search is a common optimization to a linear scan when searching for an element's index or insertion point if it doesn't exist.
- In these problems, left and right represent the bounds of the subarray we are currently considering. mid represents the index of the middle of the current search space.
- Sometimes, you will directly be binary searching for the answer.
- Other times, binary search will just be a tool that speeds up your algorithm.

Binary Search

- You are given an array of integers nums which is sorted in ascending order, and an integer target.
- If target exists in nums, return its index. Otherwise, return -1.

Search a 2D Matrix

- Write an efficient algorithm that searches for a value target in an m x n integer matrix matrix.
- Integers in each row are sorted from left to right.
- The first integer of each row is greater than the last integer of the previous row.

Successful Pairs of Spells and Potions

- You are given two positive integer arrays spells and potions, where spells[i] represents the strength of the i^{th} spell and potions[j] represents the strength of the j^{th} potion.
- You are also given an integer success.
- A spell and potion pair is considered successful if the product of their strengths is at least success.
- For each spell, find how many potions it can pair with to be successful.
- Return an integer array where the i^{th} element is the answer for the i^{th} spell.

Search Insert Position

- Given a sorted array of distinct integers and a target value, return the index if the target is found.
- If not, return the index where it would be if it were inserted in order.
- You must write an algorithm with O(log n) runtime complexity.

Examples

- Input:
  - nums = [1,3,5,6], target = 5
- Output:
  - 2
- Input:
  - nums = [1,3,5,6], target = 2
- Output:
  - 1
- Input:
  - nums = [1,3,5,6], target = 7
- Output:
  - 4

Constraints

- $1 <= nums.length <= 10^4$
- $-10^4 <= nums[i] <= 10^4$
- nums contains distinct values sorted in ascending order.
- $-10^4 <= target <= 10^4$

Longest Subsequence With Limited Sum

- You are given an integer array nums of length n, and an integer array queries of length m.
- Return an array answer of length m where answer[i] is the maximum size of a subsequence that you can take from nums such that the sum of its elements is less than or equal to queries[i].
- A subsequence is an array that can be derived from another array by deleting some or no elements without changing the order of the remaining elements.

Example

- Input:
  - nums = [4,5,2,1], queries = [3,10,21]
- Output:
  - [2,3,4]

Explanation

- The subsequence [2,1] has a sum less than or equal to 3. It can be proven that 2 is the maximum size of such a subsequence, so answer[0] = 2.
- The subsequence [4,5,1] has a sum less than or equal to 10. It can be proven that 3 is the maximum size of such a subsequence, so answer[1] = 3.
- The subsequence [4,5,2,1] has a sum less than or equal to 21. It can be proven that 4 is the maximum size of such a subsequence, so answer[2] = 4.

Example

- Input:
  - nums = [2,3,4,5], queries = [1]
- Output:
  - [0]

Explanation

- The empty subsequence is the only subsequence that has a sum less than or equal to 1, so answer[0] = 0.

Constraints

- n == nums.length
- m == queries.length
- 1 <= n, m <= 1000
- 1 <= nums[i], queries[i] <= 10^6

Queries?

Thank You…!