Searching and Sorting

Srivaths P

Goal

- Understand searching
- Learn prefix sums and frequency arrays
- Understand sorting
- Understand binary search

What is Searching

Searching is the process of finding some information in a container depending on some conditions.

Searching is predominant in CP problems.

Important searching algorithms:

- Prefix sums
- Frequency arrays
- Binary Search

Prefix Sums

A prefix sum stores the sum of the prefix of an array at each index. Takes O(N) time complexity to compute.

```
prefix[k] = sum of array from 0 to k
```

Prefix sums can be used to answer queries such as “Sum of elements of array from [L, R]” in O(1) time complexity

Implementation

- $O(N^2)$:

```
for (int i = 0; i < n; i++) {
    prefix_sum[i] = 0;
    for (int j = 0; j <= i; j++)
        prefix_sum[i] += a[j];
}
```

- $O(N)$:

```
prefix_sum[0] = a[0];
for (int i = 1; i < n; i++)
    prefix_sum[i] = prefix_sum[i-1] + a[i];
```

Sum of range in O(1)

We can write sum from [L, R] as
sum from [0, R] – sum from [0, L-1]

Which can be written as

```
prefix_sum[r] - prefix_sum[l-1]
```

Note: Pre-computation takes O(N)

Frequency array/map

Map that stores the count for each value is known as frequency map.

Useful when the count of each character is required multiple times

Implemented as:

```
for (auto i: a)
    freq[i]++;
```

Quiz 1:

1. Print all the values which are greater than given integer

2. For each element, print it if it has the maximum frequency in the array

3. Queries on number of primes in the range [L, R] in O(1)

What is Sorting

Sorting is used to arrange the elements of a container in a specific way.

The comparator of a sort function takes two values, and decides which one should come first.

Binary Search

Binary search is a searching algorithm for a sorted collection of data.

It divides the range to search by half every iteration.

Time complexity: O(logn)

Takes ~20 iterations to search 10^6 elements

Implementation 1

Checks if target is present in the array

```
bool search(vector<int> a, int target) {
    int left = 0, right = a.size() - 1;

    while (left <= right) {
        int mid = (left + right) / 2;
        if (a[mid] == target)
            return true;

        if (a[mid] < target) left = mid + 1;
        if (a[mid] > target) right = mid - 1;
    }

    return false;
}
```

Implementation 2

Finds the last index of target

```
int search(vector<int> a, int target) {
    int left = 0, right = a.size() - 1;

    while (left < right) {
        int mid = (left + right + 1) / 2;

        if (a[mid] <= target) left = mid;
        if (a[mid] > target) right = mid - 1;
    }

    return (a[left] == target) ? left : -1;
}
```

Points to Note

- If you are stopping when L = R, check if (L+R)/2 should be floored or ceiled. It might be an infinite loop otherwise.
- Make sure your boundaries are correct.
- You can use L + ((R-L)/2) to avoid errors in some cases where L+R can overflow
- If you ever need to run binary search on an infinite list, you can use LLONG_MAX or some other appropriate value as the upper-bound

Quiz 3

1. Check if either A or –A exists in array sorted by the abs value.
2. How would you find the first index an element appears in a sorted array.
3. Find the integer square root of a number using binary search.
4. Find the largest N such that N*(N+1) / 2 is less than given target.

Quiz 3 – P2 solution

Finds the first index of target

```cpp
int search(vector<int> a, int target) {
    int left = 0, right = a.size() - 1;

    while (left < right) {
        int mid = (left + right) / 2;

        if (a[mid] < target) left = mid + 1;
        if (a[mid] >= target) right = mid;
    }

    return (a[left] == target) ? left : -1;
}
```

Thanks for watching!

Feedback form: https://forms.gle/LtCUeh5JNbpZHxVu9