Binary Search
+ Problem Solving

Srivaths P

Goal:

- To learn about prefix sums
- To learn the concept of binary search
- To learn when binary search can be applied
- Problem Solving

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[i];
}
```

- 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 [0, R] - sum from [0, L-1]

Which can be written as:

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

We need to take [0, L-1] as L is included in [L, R].

Note: Pre-computation takes O(N)

Binary Search

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

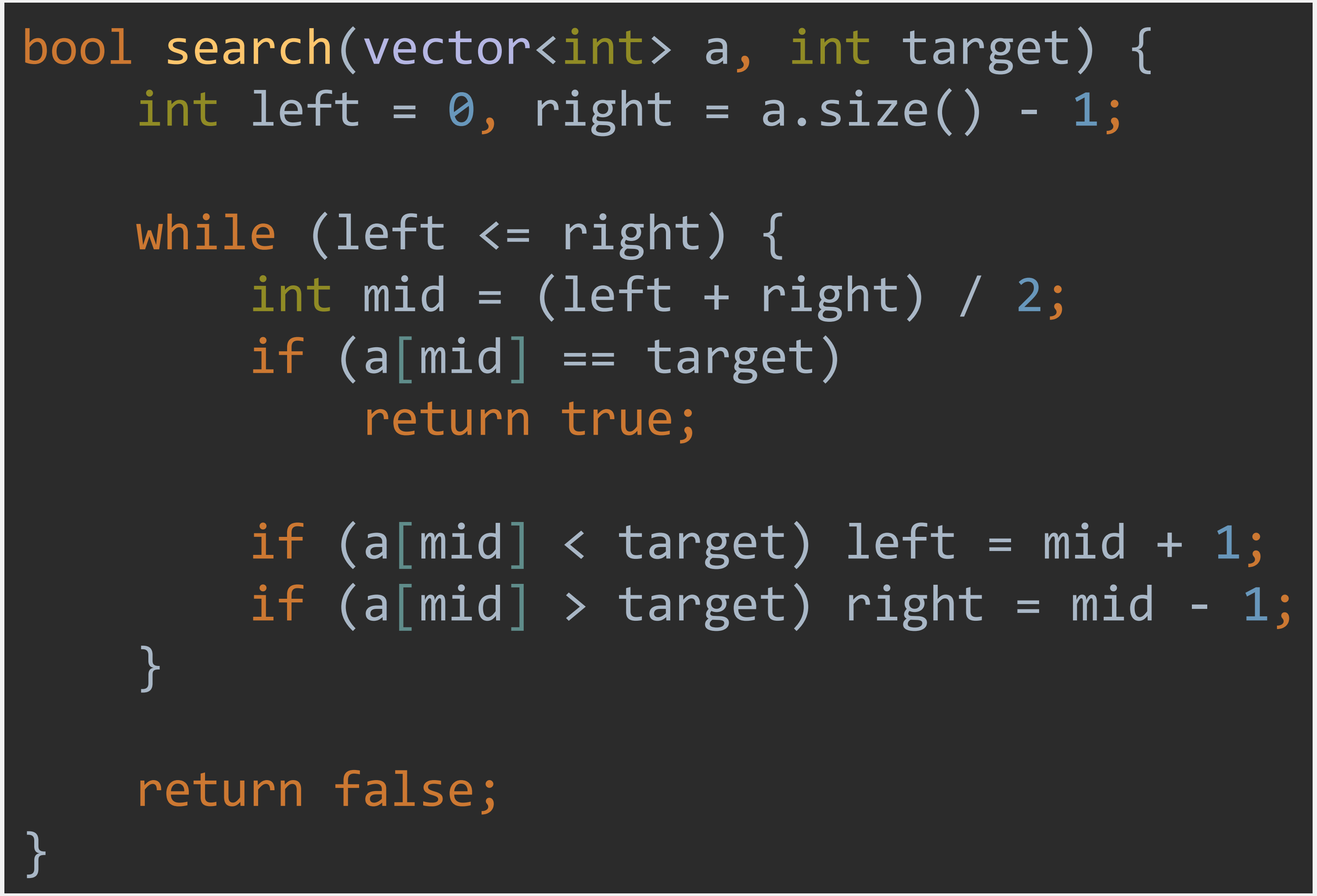
It divides the range to search by half every iteration.

Time complexity: O(logn)

Takes ~20 iterations to search 10^6 elements

Implementation 1

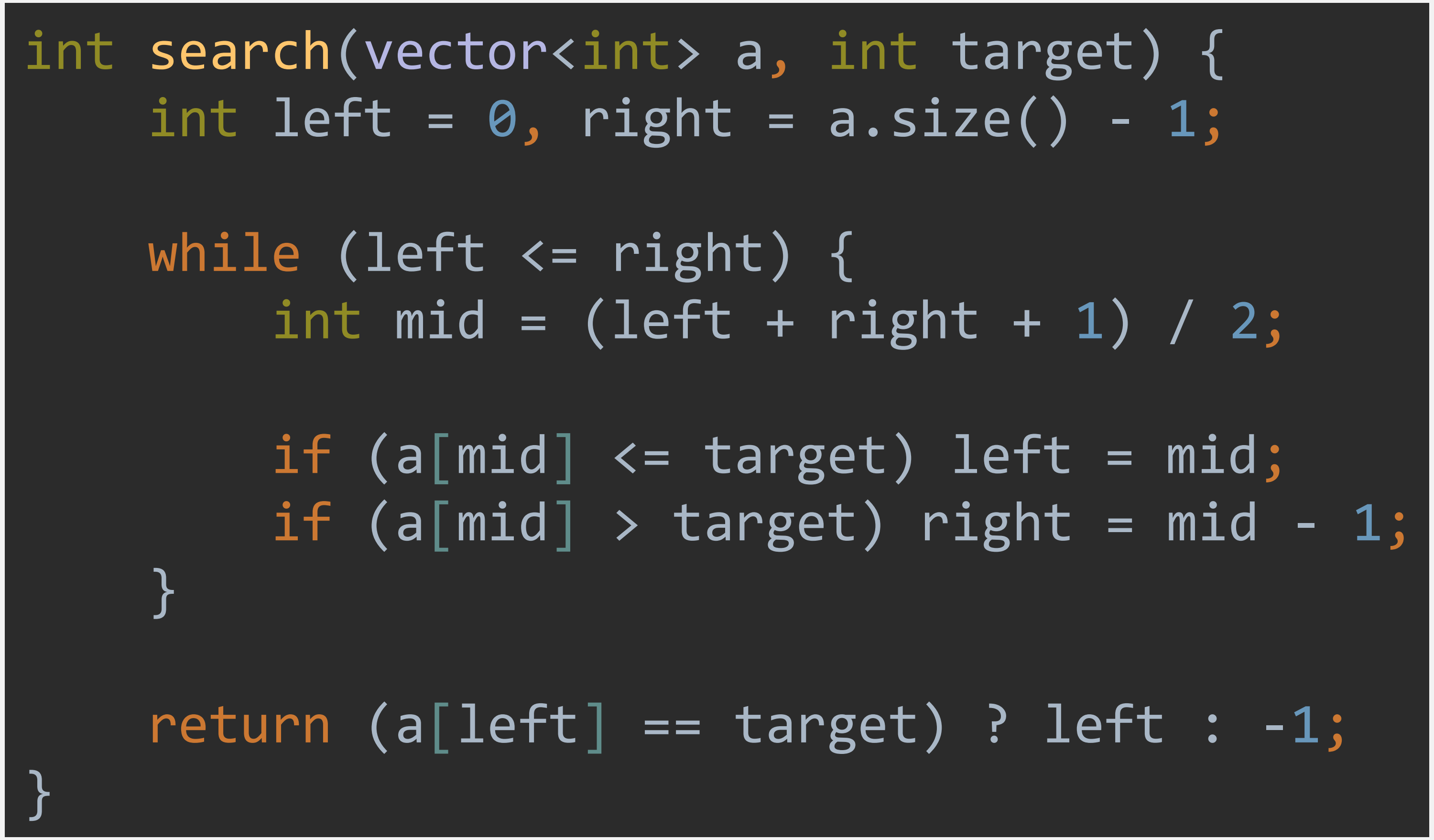
Checks if target is present in the array



```
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;
}
```

Binary Search Conditions

Binary search works on a set of elements where the “predicate” function applied on it is as follows:

$$T\ T\ T\ \ \ldots\ T\ T\ F\ F\ \ \ldots\ F\ F\ F$$

Binary search will move:

- L to mid when predicate is true.
- R to mid when predicate is false.

Alternative Binary Search

```
int l = min-1, r = max+1;
while (r-l > 1) {
    int m = (l + r) / 2;
    if (predicate(m))
        l = m;
    else
        r = m;
}

// l is the last true
// r is the first false
```

Points to Note

- When L = R-1, 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/overflows in some cases where L+R exceeds the integer limit.

- 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.

Problem Solving

- https://cses.fi/problemset/task/1068
- https://cses.fi/problemset/task/1083
- https://cses.fi/problemset/task/1069
- https://cses.fi/problemset/task/1094
- https://cses.fi/problemset/task/1070
- https://leetcode.com/problems/find-peak-element/

Thanks for Watching!

Feedback form:
https://docs.google.com/forms/d/e/1FAIpQLScQEC2PyA4QmBdVFhbZ7_2h5x-vjeC9Lg-FXxMRXSyJMKsM2g/viewform