C++ Notes

Bhanu Prasanna Koppolu

2023-10-25

Table of contents

Day - 1

Time Complexity

Important Points

- ***Definition*** - Time Complexity is the relationship about how the time will grow as the input grows.
- Time Complexity != Total Time Taken by a Machine to execute.
- There are mainly 4 points for Time Complexity:

    1. Always look for the **Worst Case Time Complexity**.
    2. Always look at the **Complexity for Large Data**.
    3. Always **Ignore Constants**.
    4. Always **Ignore Less Dominant Terms**. *From Point 2*

Akra-Bazzi Time Complexity Method

If the recurrence become more complex then finding the time complexity will become harder. So, Akra-Bazzi method is the easy and simple method to find the Time Complexity.

The method goes as follows:

Let p be the unique real number for which

$$\sum_{i=1}^{k} a_i b_i^p = 1$$

Then,

$$T(x) = \Theta\left(x^p + x^p \int_1^x \frac{g(u)}{u^{p+1}} du\right)$$

Asymptotic Notation

The usage of Asymptotic Notation is for specifying and for the identification of the upper and lower bounds.

$$\lim_{x \to \infty} \frac{f(n)}{g(n)}$$

Below are the Asymptotic Notations:

Big - O Notation ($\leq$ Upper Bound)

This is the Upper Bound and the complexity cannot go beyond this Upper Bound.

$$\lim_{x \to \infty} \frac{f(n)}{g(n)} < \infty$$

Big - Ω Notation ($\geq$ Lower Bound)

This is the opposite of Big - O Notation. This is the Lower Bound and the function complexity will atleast require Lower Bound complexity.

$$\lim_{x \to \infty} \frac{f(n)}{g(n)} > 0$$

Big - Θ Notation (Average)

The Big - Θ Notation was created when a function lies in both Upper Bound (Big - O) and Lower Bound (Big - Ω).

$$0 < \lim_{x \to \infty} \frac{f(n)}{g(n)} < \infty$$

Little - o Notation ($<$ Upper Bound)

The Little - o is same as Big - O but it is slightly loose and strictly lesser than.

$$\lim_{x \to \infty} \frac{f(n)}{g(n)} = 0$$

Little - ω Notation ($>$ Lower Bound)

The Little - ω is same as Big - Ω but it is slightly loose and strictly greater than.

$$\lim_{x \to \infty} \frac{f(n)}{g(n)} = \infty$$

Space Complexity

- Space Complexity is measured by combining the Input Space and the Auxilary Space. The same Asymptotic Notations discussed above are the ones used for Space Complexity.

Linear Recurrence (Homogenous and Non-Homogenous)

- Resources Used:
  - Click Here!

This is the end of ***Day - 1***.

Day - 2

Recursion

Important Points

- Recursion helps in solving bigger/complex problems into smaller problems which can be solved in a simple way.
- You can convert a Recursion solution into Iterative and Vice Versa.
- Space Complexity is not constant due to Recursive Calls.

Types of Recurrence Relations

1. Linear Recurrence Relation
2. Divide & Conquer Recurrence Relation

How to understand & approach a problem?

- Identify if you can break down problem into smaller problems.
- Write the Recurrence Relations if needed.
- Draw the Recursive Tree.
- About the Tree:
  - See the flow of functions, how they are getting into the stack.
  - Identify and Focus on Left Tree Calls and Right Tree Calls.
  - Draw the tree and pointers using Pen & Paper for better understanding. *
  - Use a debugger to see the flow.
- See how the values and what type of values are returned at each step. See where the function will come out. In the end you will come out of the main function.
- Three Variables to concentrate on:
  - Arguments
  - Return Type
  - Body of Function
- Resources used:

  - Click Here!

Math for DSA

- Resources used:
  - Math for DSA 1 - Click Here!
  - Math for DSA 2 - Click Here!

This is the end of ***Day - 2***.

Day - 3

C++ Start

- I have solved 18 Hackerrank Problems in the C++ section. - Click Here!.

Day - 4

- I have solved total of 28 Hackerrank Problems in the C++ section. ***I feel confident in C++.*** - Click Here!.

C++ from Book

- C++ developed by Bjarne Stroustroup at Bell Labs (1979).

Learning Language Tradition - Hello, World! Program

```
// Preprocessor Directive
#include <iostream>

// Start of the Program
int main() {
  // Tell the compiler what namespace to search in
  using namespace std;

  // Write to the screen using std::cout
  cout << "Hello, World!" << endl;

  // Return a value to the OS
  return 0;
}
```

Day - 5

- I have solved 5 LeetCode Questions using C++ in the Arrays Section. - Click Here!.

LeetCode Questions & Answers

Two Sum

```cpp
//  Two_Sum.cpp
//  LeetCode - C++
//  Created by Bhanu Prasanna

class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        int l = nums.size();

        for (int i = 0; i < l; i++) {
            for (int j = 0; j < l && i != j; j ++) {
                if (nums[i] + nums[j] == target) {
                    return {i, j};
                }
            }
        }

        return {};
    }
};
```

Remove Duplicates from Sorted Array

```cpp
//  Remove_Duplicates_from_Sorted_Array.cpp
//  LeetCode - C++
//  Created by Bhanu Prasanna

class Solution {
public:
    int removeDuplicates(vector<int>& v) {
        int j = 1;

        for (auto i = v.begin() + 1; i != v.end(); i++) {
            if (*i != *(i - 1)) {
                v[j] = *i;
                j++;
            }
        }

        return j;
    }
};
```

Remove Element

```cpp
//  Remove_Element.cpp
//  LeetCode - C++
//  Created by Bhanu Prasanna

class Solution {
public:
    int removeElement(vector<int>& v, int val) {
        int j = 0;

        for (auto i = v.begin(); i != v.end(); i++) {
            if (*i != val) {
                v[j] = *i;
                j++;
            }
        }
```

```
        return j;
    }
};
```

Search Insert Position

```
//  Search_Insert_Position.cpp
//  LeetCode - C++
//  Created by Bhanu Prasanna

class Solution {
public:
    int searchInsert(vector<int>& nums, int target) {
        auto x = std::lower_bound(nums.begin(), nums.end(), target);

        cout << (x - nums.begin()) << endl;

        return x - nums.begin();
    }
};
```

Plus One

```
//  Plus_One.cpp
//  LeetCode - C++
//  Created by Bhanu Prasanna
//  Not the Best Solution - Must Learn How to Optimize a Solution.


class Solution {
public:
    vector<int> plusOne(vector<int>& v) {
        int c = 0, l = v.size();

        for (int i = l-1; i >= 0; i--) {
            if (i == l-1) {
                if (v[i] == 9) {
                    v[i] = 0;
                    c = 1;
```

```
                } else {
                    v[i] += 1;
                    return v;
                }
            } else {
                if (v[i] == 9) {
                    v[i] = 0;
                    c = 1;
                } else {
                    v[i] += c;
                    return v;
                }
            }
        }

        if (*v.begin() == 0) {
            v.insert(v.begin(), 1);
        }

        return v;
    }
};
```