Complexity and Basic C++

Neeyanth Kopparapu

October 2019

1 Complexity Introduction

Complexity, which is often expressed in Big-O notation, is useful for estimating how long a solution for take for large test cases. You should calculate the complexity of your solution before implementing to ensure that it will fit within the time limit. Also, knowing what complexity is required to solve the problem based on the sizes of the test inputs can give you a massive hint on how to solve the problem.

2 Calculating Complexity

Calculating the complexity of a solution is very simple. Consider all loops that vary based on a variable given from the test input, such as:
for(int $k = 0$; $k \leq N$; $k++$), where $1 \leq N \leq 100000$.
Those loops are of size O(N). You may have a loop, such as a binary search, that you know will only loop $logN$ times, so those loops are of size O($logN$).
Nested loops multiply. A O($logN$) loop nested inside of a O(N) loop will result in O($NlogN$).
Different sets of loops add. Usually, only the largest term is represented, but you can write every term for clarity. For example, if you have an O(N^2) loop followed by O(N) loop, the Big-O will be O(N^2), although O($N^2 + N$) would be more precise.

3 Predicting Complexity Required based on the Problem

The following table details the complexity required based on N.

N	Complexity
10^9	O(1)
10^6	O(N)
10^5	O(N) or O($N \log N$)
10^4	O(N^2)
10^3	O(N^2)
10^2	O(N^3)

You can intuitively determine if a complexity is fast enough without memorizing that chart by substituting the values of N into the Big-O expression and seeing if the result is $<< 10^9$.
Before implementing a solution, you should first calculate the complexity of your solution and see if it will fit within the time limit. If it doesn't, don't waste time implementing a solution that will time out (unless you are desperate).

4 Introduction to C++

C++ is a general purpose programming language that is very quick because it is very low-level compared to other scripting languages like Python and general purposes languages like Java. C++ is similar in syntax to Java in which it is **statically typed** and **compiled**. Additionally, C++ is an **Object Oriented** programming language, so you can write classes, structs, and many more.

5 General Format of C++ Program

C++ functions need a main() method that is run from the start. The program can have helper methods, and global variables. Typically using global variables is advised compared to local variables (especially for arrays!) because local variables are created on the stack, whereas global variables are stored in the same place as constants, (neither the stack nor the heap), which allows for quicker and dynamic memory allocation. Thus, when you are making arrays that are global variables (recommended), you need to specify the size of the array right at the beginning, which might not be the most memory efficient. In general, global variables are the easiest to pass variables and edit them between functions, instead of passing variables (unless you are using recursion).

6 Object Oriented Programming in C++

Like Java, C++ is object oriented. But it is special in that there isn't one type of "Class". There are Classes, Structs, and Namespaces to group data together. Some common classes include vectors (like ArrayLists in Java), string, and queue. Class names are lower cased in C++. Namespaces and Structs are just ways to structure data in an easily accessible manner. Namespaces and Classes allow for custom methods as well.
For competitive programming in specific, you can find many namespaces that contain methods and algorithms related to a specific topic (Segment tree querying and construction, etc.)

6.1 Example Struct Definition

```
// Point.cpp
struct Point {
    int x, y;
};
```

Some pointers:

1. You can give variables default values, but it is generally unrecommended due to further inconsistencies with initialization.

2. You can initialize a Point in many ways. By saying Point p; (0,0) or Point p = 4,2; (4,2) both initialize an object. Then the member variables can be accessed just like in Java.

7 References and Pointers

One of the most complicated topics in C++ is the use of references and pointers. Understanding all of the nuance with references and pointers allows you to make complicated structures that can be used for creating data structures like linked lists, etc.

7.1 Example Definition of Linked List

```
// LinkedList.cpp
class Node
{
public:
    Node* next;
    int data;
};
class LinkedList
{
public:
    int length;
    Node* head;
    LinkedList();
    ~LinkedList();
    void add(int data);
    void print();
};
LinkedList::LinkedList(){
    this->length = 0;
    this->head = NULL;
}
LinkedList::~LinkedList(){
    std::cout << "LIST DELETED";
}
void LinkedList::add(int data){
    Node* node = new Node();
    node->data = data;
    node->next = this->head;
    this->head = node;
    this->length++;
}
void LinkedList::print(){
    Node* head = this->head;
    int i = 1;
    while(head){
        std::cout << i << ": " << head->data << std::endl;
        head = head->next;
        i++;
    }
}
```

8 Exercises and Further Reading

To learn C++, you can go to https://www.learncpp.com/ for further teaching and resources.
We will try to put a few interactive questions for you to delve into the concept further than thought/written questions. **We cannot stress how important it is to not look up the answer.** Questions like these and USACO questions are meant to be challenging and hard. Nothing good comes easy. USACO (as mentioned above) gives you 4-5 hours to solve these problems! And most, if not everyone, needs the 4-5 hours to solve maybe 1-2 questions! Looking up answers for questions like these puts you in the mentality that you can ask for help for everything in life. **That is not always the case.** If you need a hint, ask the officers. Keep in mind, though, that you can't do that in competitions that matter. Try your hardest before asking for a hint!

Problem 1: One of the famous coding and algorithmic programming questions that often appears in Coding Interviews like Facebook's is the **longest increasing subsequence** problem. The Longest Increasing Subsequence (LIS) problem is to find the length of the longest subsequence of a given sequence such that all elements of the subsequence are sorted in increasing order. For example, the length of the LIS for $\{10, 22, 9, 33, 21, 50, 41, 60, 80\}$ is 6 and the LIS is $\{10, 22, 33, 50, 60, 80\}$.

There are 2 solutions to this question. One of the first that comes to the minds of many is the $O(N^2)$ solution. Now this isn't completely straightforward. Your first task is to find this $O(N^2)$ solution and code it. Then, empirically prove that your algorithm runs in $O(N^2)$. Write a way to generate a sequence and solve the problem given an N. Then, time different runs of N and see that it runs in $O(N^2)$.

Challenge: Find a way to solve this problem in $O(N \log N)$. Use the same empirical method as described above to prove complexity.