Assignment Lecture

07. September 2022

Content - (short lecture today)

- Assignment 1
  - Theory
    - Converting to O-notation
    - Sorting Functions
    - Simplifying Expressions
    - Runtime Analysis
  - Programming
- Assignment 2 (will be posted on friday)
  - Brief Introduction

- Assignment 1
  - Theory
    - Converting to O-notation
    - Sorting Functions
    - Simplifying Expressions
    - Runtime Analysis
  - Programming
    - Named Tuples
- Assignment 2 (will be posted on friday)
  - Brief Introduction

Converting to O-notation

(2n(3n) + 4n)***5log(n)** O((2n(3n) + 4n)*5log(n))

Simplify the expression: O((6n^2 + 4n)*5log(n))

Only keep highest order components: O(6n^2 5log(n))

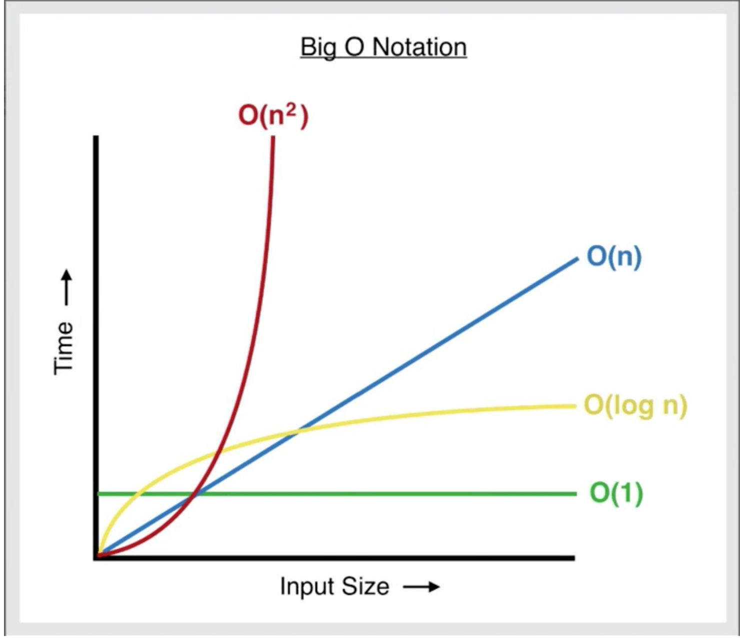
Remove constants: **O(n^2 log(n))**

- Assignment 1
  - Theory
    - Converting to O-notation
    - Sorting Functions
    - Simplifying Expressions
    - Runtime Analysis
  - Programming
- Assignment 2 (will be posted on friday)
  - Brief Introduction

Sorting Functions

1. Convert to O-notation
2. Order by how good/fast they are.
   - Flatter Function = Faster for large input = Better
   - Most common:

     https://alginf.idi.ntnu.no/curriculum/#asymptotic-notation



Example:

$$f = 2n+5n,\ g = (n^2)/5$$

gives **f < g** because:

$$O(f)=O(n) < O(g)=O(n^2)$$

- Assignment 1
  - Theory
    - Converting to O-notation
    - Sorting Functions
    - Simplifying Expressions
    - Runtime Analysis
  - Programming
- Assignment 2 (will be posted on friday)
  - Brief Introduction

Simplifying Expressions

- Same idea as converting to O-notation

$\Theta(n^2) + \Omega(n) + \Theta(\log n) = \Theta(n^2)$

- To avoid losing precision, might need to specify both best and worst case

$O(n^2) + \Omega(n) + \Theta(\log n) = O(n^2)$ and $\Omega(n)$

- Assignment 1
  - Theory
    - Converting to O-notation
    - Sorting Functions
    - Simplifying Expressions
    - Runtime Analysis
  - Programming
- Assignment 2 (will be posted on friday)
  - Brief Introduction

Runtime Analysis

```
int count = 0;
for (int i = 0; i < N; i++)
    for (int j = 0; j < i; j++)
        count++;
```

Lets see how many times **count++** will run.

When $i = 0$, it will run 0 times.
When $i = 1$, it will run 1 times.
When $i = 2$, it will run 2 times and so on.

Total number of times **count++** will run is $0 + 1 + 2 + \ldots + (N-1) = \frac{N*(N-1)}{2}$. So the time complexity will be $O(N^2)$.

Runtime Analysis

```
int count = 0;
for (int i = N; i > 0; i /= 2)
    for (int j = 0; j < i; j++)
        count++;
```

This is a tricky case. In the first look, it seems like the complexity is $O(N * logN)$. N for the $j's$ loop and $logN$ for $i's$ loop. But its wrong. Lets see why.

Think about how many times **count++** will run.

When $i = N$, it will run N times.
When $i = N/2$, it will run $N/2$ times.
When $i = N/4$, it will run $N/4$ times and so on.

Total number of times **count++** will run is $N + N/2 + N/4 + \ldots + 1 = 2 * N$. So the time complexity will be $O(N)$.

- Assignment 1
  - Theory
    - Converting to O-notation
    - Sorting Functions
    - Simplifying Expressions
    - Runtime Analysis
  - Programming
- Assignment 2 (will be posted on friday)
  - Brief Introduction

Programming

- Find optimal match using a set of preferences.

- Helpful Links:
  - Named Tuples

- Assignment 1
  - Theory
    - Converting to O-notation
    - Sorting Functions
    - Simplifying Expressions
    - Runtime Analysis
  - Programming
- Assignment 2 (will be posted on friday)
  - Brief Introduction

Assignment 2 - Graphs

Will contain:

- Graph and Tree theory
- Topological Sorting of a graph
- BFS (Breadth First Search) and DFS (Depth First Search)