Asymptotic Analysis

Think Big

UTEC - Competitive Programming

Complexity

- When describing the performance of an algorithm, the word complexity is commonly used.
- Complexity can be considered as the amount of resources my algorithm will consume:
  - **Time Complexity:** How long will my program take to run for a given input.
  - **Space Complexity:** How much memory will my program consume for a given input.
- In many cases there exists a trade-off between space and time complexity. More memory = Less time

Challenge

Lets find a solution for the following problem:

Given an array a with n elements, find the smallest one. $n \leq 10^6$

Challenge

Lets find a solution for the following problem:

Given an array a with n elements, find the smallest one. $n \leq 10^6$

- How long will your solution take if $n = 10^3, 10^5, 10^6$

Challenge

Lets find a solution for the following problem:

Given an array a with n elements, find the smallest one. $n \leq 10^6$

- How long will your solution take if $n = 10^3, 10^5, 10^6$
- How many solutions there are to this problem?

Challenge

Lets find a solution for the following problem:

Given an array a with n elements, find the smallest one. $n \leq 10^6$

- How long will your solution take if $n = 10^3, 10^5, 10^6$
- How many solutions there are to this problem?
- Is this solution the best solution?

Challenge

Lets find a solution for the following problem:

Given an array a with n elements, find the smallest one. $n \leq 10^6$

- How long will your solution take if $n = 10^3, 10^5, 10^6$
- How many solutions there are to this problem?
- Is this solution the best solution?
- How can we compare solutions?

Measuring Complexity

Sampling and Extrapolation

- One first approach to measuring complexity can be to manually measure how long our solution takes to run for different inputs.
- We can then try to extrapolate our measurments and attempt to predict the time our program will take for other inputs.
- Let's try it out!

Sampling and Extrapolation - Problems

- Easy to miss **corner cases**.
- Takes a lot of time.
- Requires a lot of guessing.
- Sometimes it is really hard to generate input.

Changing Perspective

- Lets define the function $T(x)$ as the time our algorithm takes to run for an input x.
- Lets also define $N(x)$, as the number of instructions our algorithm takes for an input x.
- It is simple to see that $T(x) \alpha N(x)$

Changing Perspective

- Overall, we can say that:

$$T(x) = kN(x)$$

- What would k represent in this equation?
- $k_{C++} \approx 10^8$, $k_{py} \approx 10^6$
- Counting the number of steps is esay! ... *usually*

Counting instructions

- The goal is to count how many basic instructions are executed during an algorithm.
- Assume that basic instructions are those that can be executed natively by the computer:
  - addition, subtraction, multiplication, division, remainder
  - modifying and calling variables
  - calling functions
  - etc...
- We assume that all instructions take the same time.
- Find $N(n)$ for the following solutions

Counting instructions - Practice

```
int main() {
    int n;
    cin >> n;

    int a[n];
    for (int i = 0; i < n; i++) {
        cin >> a[i];
    }

    int smallest = INT_MAX;
    for (int i = 0; i < n; i++) {
        smallest = min(smallest, a[i]);
    }

    cout << smallest << endl;
    return 0;
}
```

Counting instructions - Practice

```
int main() {
    int n;
    cin >> n;

    int a[n];
    for (int i = 0; i < n; i++) {
        cin >> a[i];
    }

    sort(a, a + n);
    cout << a[0] << endl;
    return 0;
}
```

Asymptotic Analysis

Asymptotic Analysis - Introduction

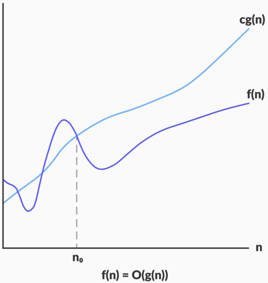
- It is a method for defining the mathemathical boundry of the run-time performance of an algorithm.
- Using asymptotic analysis we look to find:
  - Lower Bound - $\Omega(f(n))$ Big Omega Notation
  - Tight Bound - $\Theta(f(n))$ Big Theta Notation
  - **Upper Bound** - $O(f(n))$ **Big Oh Notation**
- In competitive programming we only care about $O(n)$. **Why?**

Big Oh - A Formal Introduction

- $O(f(n))$ is a set that contains all functions that are asymptotically smaller than $f(n)$.
- Formally:

$$f(n) = O(g(n)) \rightarrow f(n) < kg(n)$$

For $n > n_0$, $n_0 \in \mathbb{N}$ and $k > 0$



Big Oh - In Practice

- In general lets say we can estimate that our program has a time complexity of:

$$T(n) = a_m n^m a_{m-1} n^{m-1} + \ldots + a_1 n + a_0$$

- It is simple to see that as n gets bigger, $T(n)$ will be more heavily dominated by its largest term.
- Therefore,$T(n) \approx n^m$, for sufficiently large n.
- We can also realize that $T(n) = O(n^m)$. **Why?**
- Now that we know we only care about the largest term calculating $T(n)$ should be much faster!

Using Asymptotic Analysis

In general, from the constrains of a problem we can get the expected complexity our solution should have. For example:

Input Size	**Maximum Valid Complexity**
$n \leq 10$	$O(n!)$
$n \leq 20$	$O(2^n)$
$n \leq 500$	$O(n^3)$
$n \leq 5000$	$O(n^2)$
$n \leq 10^6$	$O(nlgn)$
$n \leq 10^7$	$O(n)$
$n > 10^7$	$O(1)$ or $O(lgn)$