Problem Set 1

Jose Chavez

September 1, 2019

1 Ordering functions

Exercise 1.1. Arrange the following functions in increasing order of growth rate and justify your answer through a proof.

a) 2^n

b) $\log_2 n$

c) $n^{\log_2 n}$

d) 2^{n^2}

e) 2^{2^n}

The order is as follow:

$$\log_2 n < n^{\log_2 n} < 2^n < 2^{n^2} < 2^{2^n}$$

1. *Proof* of $\log_2 n < n^{\log_2 n}$
   We say $a = \log_2 n$, then

   $$an < n^a \text{ ; for every } a \geqslant 0$$

2. *Proof* of $2^n < 2^{n^2}$
   As the bases are the same we just compare the exponents

   $$n < n^2$$

3. *Proof* of $2^{n^2} < 2^{2^n}$
   As the bases are the same we just compare the exponents

   $$n^2 < 2^n$$

   We know that it holds when $n \geqslant 5$
   So we will Assume is true an we will try to proof it with $n + 1$

   $$\begin{aligned} n^2 &< 2^{n+1} \\ &< 2^n * 2 \end{aligned}$$

   We can multiply by 2 our first term, so it will because

$$(n+1)^2 < 2*n^2 < 2*2^n$$

we know that $(n-1)^2 \geqslant 4^2 > 2$ since $n \geqslant 5$ so we will expand that inequality

$$\begin{aligned}(n-1)^2 &> 2\\ n^2-2n-1 &> 0\\ 2n^2-2n-1 &> n^2\\ 2n^2 &> n^2+2n+1\\ 2n^2 &> (n+1)^2\end{aligned}$$

2 O, Θ and Ω exercises that were definitely not taken from CLRS

Exercise 2.1. Prove or disprove: $f(n) = \Theta(g(n))$ if and only if $g(n) = \Theta(f(n))$.

Assuming the first statement is true. And by Θ definition we have

$$\begin{aligned}c_1 g(n) \leqslant &f(n) \leqslant c_2 g(n)\\ g(n) &\leqslant 1/c_1 f(n)\\ 1/c_2 f(n) &\leqslant g(n)\end{aligned}$$

Taking a $z_1 = 1/c_1$ y $z_2 = 1/c_2$ we have

$$z_2 f(n) \leqslant g(n) \leqslant z_1 g(n)$$

we proof that $g(n) \in \Theta(f(n))$

Exercise 2.2. Prove or disprove: $f(n) = \Theta(f(n))$.

We start by assuming that $f(n) = \Theta(f(n))$ is true. So $c_1 f(n) \leqslant f(n) \leqslant c_2 f(n)$. Then we must proof that $f(n) = \Omega(n)$ and $f(n) = O(n)$. We use such definitions

$$0 \leqslant c_1 f(n) \leqslant f(n) \text{ and } f(n) \leqslant c_2 f(n)$$

Thus,

$$\begin{aligned}c_1 f(n) &\leqslant c_2 f(n)\\ c_1 &\leqslant c_2\end{aligned}$$

From our definiton $f(n) = \Omega(n)$ and $f(n) = O(n)$ holds when $c_1 \leqslant c_2$. Thus , $f(n) = \Theta(f(n))$.

Exercise 2.3. Using the basic definition of Θ-notation, prove that $\max(f(n), g(n)) = \Theta(f(n) + g(n))$.

We will see two ways of proving it.

First using the definition we can say

$$c_1(f(n)+g(n)) \leqslant max(f(n), g(n)) \leqslant c_2(f(n)+g(n))$$

$$max(f(n), g(n)) \leqslant 1(f(n) + g(n))$$
$$max(f(n), g(n)) \geqslant 1/2(f(n) + g(n))$$

So it holds true for $c_1 \leqslant 1/2$ and $c_2 \geqslant 1$.

The other approach will be:
Note that $f(n) \leqslant f(n) + g(n)$ and $g(n) \leqslant f(n) + g(n)$. Hence

$$max(f(n), g(n)) = O(f(n) + g(n))$$

Now note that if $f(n) \geqslant g(n)$, we can say $f(n) + f(n) \geqslant f(n) + g(n)$, this equals $2(f(n)) \geqslant f(n) + g(n)$. We will use the same strategie if $g(n) \geqslant f(n)$. Hence we have

$$f(n) + g(n) \leqslant 2max(f(n), g(n))$$

Therefor

$$max(f(n), g(n)) = \Omega(f(n) + g(n))$$

Thus

$$max(f(n), g(n)) = \Theta(f(n) + g(n))$$

Exercise 2.4. Prove or disprove: $O(f(n) + g(n)) = f(n) + O(g(n))$, if $f(n)$ and $g(n)$.

Exercise 2.5. Prove or disprove: If $f(n) = O(g(n))$ and $g(n) = O(h(n))$, then $f(n) = O(h(n))$.
Based on the definitions we have

$$f(n) \leqslant c_1 g(n) \text{ and } g(n) \leqslant c_2 h(n)$$
$$f(n) \leqslant c_x h(n)$$

We arrange the inequalities then we multiply by c_1 the following expressions without interfering with the inequality

$$f(n) \leqslant g(n) \leqslant c_1 g(n) \leqslant c_2 h(n)$$
$$c_1 g(n) \leqslant c_1 c_2 h(n)$$
$$f(n) \leqslant c_1 c_2 h(n)$$

Then by reverse-defintion we have

$$f(n) = O(h(n))$$
$$f(n) \leqslant c_x h(n)$$

that holds true when $c_x \geqslant c_1 c_2$

Exercise 2.6. Prove or disprove: Suppose that $f(n)$ and $g(n)$ are two functions such that for some other function $h(n)$, we have $f(n) = O(h(n))$ and $g(n) = O(h(n))$. Then $f(n) + g(n) = O(h(n))$.
Based on the definitions we have

$$f(n) \leqslant c_1 h(n) \text{ and } g(n) \leqslant c_2 h(n)$$
$$f(n) + g(n) \leqslant (c_1 + c_2) h(n)$$

And by using O defintion, we have

$$
\begin{aligned}
f(n) + g(n) &= O(h(n)) \\
f(n) + g(n) &\leqslant c_3 h(n)
\end{aligned}
$$

It holds as long as $c_3 = c_1 + c_2$

Exercise 2.7. Prove or disprove: Let k be a fixed constant, and let $f_1(n), f_2(n), \ldots, f_k(n)$ and $h(n)$ be functions such that $f_i(n) = O(h(n))$ for all i. Then $f_1(n) + f_2(n) + \cdots + f_k(n) = O(h(n))$.

3 Solving Recurrences

Exercise 3.1. Use the Master Theorem to give an asymptotic upper bound for $T(n)$ in each of the following recurrences. Assume that $T(n)$ is constant for $n \leq 2$. Make your bounds as tight as possible.

a) $T(n) = 4T(n/4) + 5n \Longrightarrow T(n) = O(n \log n)$

b) $T(n) = 4T(n/5) + 5n \Longrightarrow T(n) = O(n)$

c) $T(n) = 5T(n/4) + 4n \Longrightarrow T(n) = O(n^{\log_4 5})$

d) $T(n) = 25T(n/5) + n^2 \Longrightarrow T(n) = O(n^2 \log n)$

e) $T(n) = 4T(\sqrt{n}) + \log^5 n \Longrightarrow T(n) = O(\log^5 n)$

f) $T(n) = T(\sqrt{n}) + 5 \Longrightarrow T(n) = O(\log \log n)$

Exercise 3.2. Use the Recurrence Tree (optional, but really helpful sometimes) to guess an upper bound and prove it using the Substitution Method.

a) $T(n) = 2T(n-1) + 1$
Guess: $O(2^n)$
I.H: $T(n) \leqslant c2^n - d$
I.S:

$$
\begin{aligned}
T(n-1) &\leqslant 2(c2^{n-1} - d) + 1 \\
&= c2^n - 2d + 1 \\
&\leqslant c2^n - d \quad \text{; when } d > 0
\end{aligned}
$$

b) $T(n) = 2T(n-1) - 1$
Guess: $O(2^n)$
I.H: $T(n) \leqslant c2^n$
I.S:

$$
\begin{aligned}
T(n-1) &\leqslant 2(c2^{n-1}) - 1 \\
&= c2^n - 1 \\
&\leqslant c2^n
\end{aligned}
$$

c) $T(n) = T(n/2) + T(n/4) + n^2$
Guess: $O(n^2)$
As we are trying to prove an upper bound we can stablish that our second term is $T(n/2)$ because $T(n/2) > T(n/4)$ so we can fix our recurrence

$$T(n) = 2T(n/2) + n^2$$

I.H: $T(n) \leqslant cn^2$
I.S:

$$\begin{aligned} T(n/2) &\leqslant 2(c(n/2)^2) + n^2 \\ &= cn^2/2 + n^2 \\ &= (2n^2 + cn^2)/2 \\ &= ((2+c)/2)n^2 \\ &\leqslant c2^n \quad ; \text{when } c \geqslant 2 \end{aligned}$$

d) $T(n) = 2T(\lfloor n/2 \rfloor + 16) + n$

4 *Spooky Problems*

Problem 4.1. Our friend Martin is moving to Barranco. He is a big fan of beer and pretty much anything that contains alcohol. Luckily, he has found a street with n bars. Since he will definitely visit all of them frequently, he wants to find an apartment close to them.

Martin wants to minimize the total distance to all of the bars and has offered you a *ronnie* to come up with an algorithm to solve his problem.

Let n be the number of bars in the street and the let the following sequence represent the street numbers where they are: $s_1, s_2, \ldots, s_i, \ldots, s_n$. Note that several bars might be in the same location/street number.

Both n and the sequence of s_i's are integers. The distance between two street numbers s_i and s_j is $d_{ij} = |s_i - s_j|$.

a) Find an $O(n \log n)$ solution.

b) *Bonus*: Can you do better? Sketch a faster algorithm.

Problem 4.2. Solve `UVa 10077: The Stern-Brocot Number System`.

23851807	10077 The Stern-Brocot Number System	Accepted	C++11	0.000	2019-09-01 02:53:36

```
#include <iostream>
#include <vector>

class Node
{
```

```cpp
public:
    double up, down;

    Node() = default;
    bool operator==(Node const &node)
    {
        if (node.up == up && node.down == down)
            return true;
        return false;
    }
    bool operator<(Node const &node)
    {
        if (node.up / node.down > up / down)
            return true;
        return false;
    }
    bool operator>(Node const &node)
    {
        if (node.up / node.down < up / down)
            return true;
        return false;
    }
    Node operator+(Node const &node)
    {
        Node newNode{node.up + up, node.down + down};
        return newNode;
    }
};

std::string find(Node node)
{
    Node left{0, 1};
    Node middle{1, 1};
    Node right{1, 0};

    std::string path;

    while (true)
    {
        if (node == middle)
            return path;
        if (node < middle)
        {
            Node temp = middle;
            right = middle;
            middle = temp + left;
            path += 'L';
        }
        if (node > middle)
        {
            Node temp = middle;
            left = middle;
            middle = temp + right;
            path += 'R';
        }
    }
}

int main()
```

```cpp
{

    double div1, div2;
    std::cin >> div1 >> div2;
    while (true)
    {
        if (div1 == 1 and div2 == 1)
            break;
        Node a1{div1, div2};
        std::cout << find(a1) << '\n';
        std::cin >> div1 >> div2;
    }
    return 0;
}
```