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) $T(n) = 2T(\lfloor n/2 \rfloor + 16) + n$
Guess: $O(nlogn)$
I.H: $T(n) \leqslant cnlogn$
I.S:

$$\begin{aligned} T(n/2+16) &\leqslant 2(c(n/2+16)log(n/2+16)) + n \\ T(n/2+16) + cnlogn &= cnlogn + c(n+32)log(n/2+16)) + n \\ &= cnlogn - cnlogn + c(n+32)log(n/2+16)) + n \\ &= cnlogn + cnlog(\frac{(n/2)+16}{n}) + 32clog((n/2)+16) + n \\ &\leqslant cnlogn \end{aligned}$$

We have to proof that $[cnlog(\frac{(n/2)+16}{n}) + 32clog((n/2)+16) + n] \leqslant 0$

$$\begin{aligned} &[cnlog(\frac{(n/2)+16}{n}) + 32clog((n/2)+16) + n] \\ &= cnlog(1/2 + 16/n) + 32clog(n(1/2+16/n)) \\ &= cnlog(1/2(1+(32/n))) + 32clogn + 32clog(1/2(1+(32/n))) + n \\ &= cnlog(1/2) + cnlog(1+32/n) + 32clogn + 32clog(1/2) + 32clog(1+(32/n)) + n \end{aligned}$$

Now choosing any $0 < \epsilon \leqslant 1$

$$\begin{aligned} &cn\log(1/2) + cn\epsilon + 32c\log n + 32c\log(1/2) + 32c\epsilon + n \\ &= c(n\log(1/2) + n\epsilon + 32\log n + 32\epsilon) + 32clog(1/2) + n \\ &< c(nlog(1/2) + n\epsilon + 32logn + 34\epsilon) + n \end{aligned}$$

Now we note that $nlog(1/2) + n\epsilon + 32logn + 34\epsilon = n(log(1/2)+\epsilon) + 34logn + 34\epsilon$ this is equivalent to $g(n)$, that is negative for large enough n since we have a negative linear function competing with a positive log function.
Thus , since $|g(n)| \in O(n)$, a c constant exists.

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: $s_1, s_2, \dots, s_i, \dots, s_n$. 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.
A solution will be if we take our sample set $s_1, s_2, \dots, s_i, \dots, s_n$ as points in a 2d-plane. So we can apply the concept of the **geometric median** . The geometric median is a point minimizing the sum of distances to the sample points. This generalizes the median, which has the property of minimizing the sum of distances for one-dimensional data, and provides a central tendency in higher dimensions. It is also known as the 1-median,spatial median, Euclidean minisum point,or Torricelli point.

   1) First we have to sort our list of points or street numbers. We can use heap sort or merge sort to do it (cost of $O(n \log n)$).
   2) Then we will just need to apply the formula to find the geometric median, which is $median = \sqrt[n]{s_1 * s_2 * \dots * s_{n-1} * s_n}$
   3) Thus , it will cost $O(n \log n)$ because of the sorting.

b) *Bonus*: Can you do better? Sketch a faster algorithm. A better approach will be one know as the median of medians algorithm. It can achieve a linear cost $O(n)$.
The algorithm takes in a list (A) and an index i : "$median - of - median(A, i)$". Assume that all elements of A are distinct (though the algorithm can be further generalized to allow for duplicate elements).

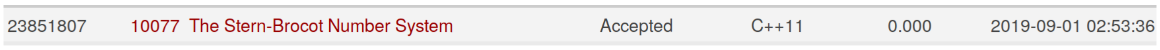
   1) Divide the list into sublists each of length five (if there are fewer than five elements available for the last list, that is fine)
   2) Sort each sublist and determine the median. Sorting very small lists takes linear time since these sublists have five elements, and this takes $O(n)$ time. In the algorithm described on this page, if the list has an even number of elements, take the floor of the length of the list divided by 2 to find the index of the median.
   3) Use the median-of-median algorithm to recursively determine the median of the set of all the medians.
   4) Use this median as the pivot element, x. The pivot is an approximate median of the whole list and then each recursive step hones in on the true median.

5) Reorder A such that all elements less than x are to the left of x, and all elements of A that are greater than x are to the right. This is called partitioning. The elements are in no particular order once they are placed on either side of x. For example, if x is 5, the list to the right of x maybe look like [8,7,12,6] i.e. not in sorted order. This takes linear time since $O(n)$ comparisons occur—each element in A is compared against x only.

6) Let k be the "rank" of x, meaning, for a set of numbers S, x is the k^{th} smallest number in S.

7)
   - If $i = k$,then return x.
   - f $i < k$, then recurse using median-of-medians on $(A[1, \ldots, k1], i)$.
   - If $i > k$, recurse using the median-of-medians algorithm on $(A[k+1, \ldots, i], ik)$.

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>

class Node
{
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()

{

    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;
}
```