Jiayu Li HW 1

Prof. Shinko

Due. 10/3

Problem 1. Show that that $n \log n$ is big-O of $n^{1.1}$, but not the other way.

Proof: Let $f(n) = n \log n$, $g(n) = n^{1.1}$

WTS $f(n) = O(g(n))$.

$$\limsup_n \frac{f(n)}{g(n)} = \limsup_n \frac{n \log n}{n^{1.1}}$$

$$= \limsup_n \frac{\log n}{n^{0.1}} \to 0$$

$$\limsup_n \frac{g(n)}{f(n)} = \limsup_n \frac{n^{1.1}}{n \log n}$$

$$= \limsup_n \frac{n^{0.1}}{\log n} \to \infty$$

Therefore $f(n) = n \log n$ grows no faster than $g(n) = n^{1.1}$

So $n \log n = O(n^{1.1})$ but not the other way.

$\blacksquare$

Problem 2. Show that n^{50} is big-O of $n^{\log n}$, but not the other way.

~~proof~~: Let $f(n) = n^{50}$, $g(n) = n^{\log n}$

WTS $f(n) = O(g(n))$,

$$\lim_n \sup \frac{f(n)}{g(n)} = \lim_n \sup \frac{n^{50}}{n^{\log n}}$$

$$= \lim_n \sup n^{50-\log n} \longrightarrow 0$$

and

$$\lim_n \sup \frac{g(n)}{f(n)} = \lim_n \sup \frac{n^{\log n}}{n^{50}}$$

$$= \lim_n \sup n^{\log n - 50} \longrightarrow \infty$$

Therefore, $f(n) = n^{50}$ grows no faster than $g(n) = 1.1$ when $n \to \infty$.

So $n^{50} = O(n^{\log n})$, but not the other way.

$\boxtimes$

Problem 3. Show that $\log(n!)$ is $\Theta(n \log n)$, i.e. that $\log(n!)$ and $n \log n$ are big-O of each other.

proof: Let $f(n) = \log(n!)$, $g(n) = n\log n$.

WTS $f(n) = \Theta(g(n))$,

First we have to prove $\log(n!) = O(n\log n)$

we have $\log(n!) = \log(1 \cdot 2 \cdot 3 \cdot \ldots \cdot (n-1) \cdot n)$

$$= \log(1) + \log(2) + \log(3) + \cdots + \log(n-1) + \log(n)$$

$$\leq \log(n) + \log(n) + \log(n) + \cdots + \log(n) + \log(n)$$

$$= n \cdot \log(n)$$

$$= O(n\log(n))$$

Second we have to prove $\log(n!) = \Omega(n\log n)$

we also have $\log(n!) = \log(1) + \log(2) + \log(3) + \cdots + \log(n-1) + \log(n)$

$$= \log(1) + \log(2) + \cdots + \log\left(\frac{n}{2}-1\right) + \underbrace{\log\left(\frac{n}{2}\right) + \cdots + \log(n-1) + \log(n)}_{n/2 \text{ terms}}$$

$$\geq \log\left(\frac{n}{2}\right) + \log\left(\frac{n}{2}\right) + \cdots \log\left(\frac{n}{2}\right)$$

$$= \log\left(\frac{n}{2}^{\frac{n}{2}}\right)$$

$$= \frac{n}{2}\log\left(\frac{n}{2}\right)$$

$$= \frac{n}{2}\left(\log(n) - \log(2)\right)$$

$$= \Omega(n\log(n))$$

Therefore $\log(n!) = \Theta(n\log n)$.

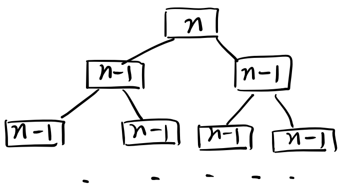
$\boxtimes$

Problem 4. What is the time complexity of the following algorithm in terms of n? Your answer should be of the form $\Theta(f(n))$ for some $f(n)$.

```
# n is a natural number
def foo(n):
    if n == 0:
        print("I am zero.")
    else:
        foo(n - 1)
        foo(n - 1)
```

Let $f(n)$ be the time complexity.

if we run the above for a couple of cases we will get:



for 1^{st}-step $n=0$: $2^0 = 1$

for 2^{nd}-step $n=1$: $2^1 = 2$

for 3^{rd}-step $n=2$: $2^2 = 4$

. . . .

for n^{th}-step $n = n-1$: 2^{n-1}

$$\Rightarrow \quad f(n) \le f(2^{n-1}) + O(n)$$

total number of steps $= n + 2(n-1) + 4(n-1) + 8(n-1) + \cdots 2^i(n-1)$

$$= \sum_{i=1}^{\infty} 2^{i-1} = \sum_{i=0}^{\infty} 2^i$$

$$= 2^n$$

$$= \theta(2^n)$$

Therefore, we will notice that it will be exactly like the function 2^n.

This algorithm has a running time of $\theta(2^n)$.

$\boxtimes$

Problem 5. What is the time complexity of the following algorithm in terms of n? Your answer should be of the form $\Theta(f(n))$ for some $f(n)$.

```
# n is a natural number
def bar(n):
    for i in range(n):
        for j in range(i, n):
            for k in range(i, j):
                print(k)
```

Let $f(n)$ is the time complexity,

total number of steps $= \sum_{i=1}^{n} \sum_{j=1}^{n-i} \sum_{k=1}^{j-i} 1$

length of span $= n$

since start of span : $n - i \leq n$

partition of span : $j - i \leq n$

$$\Rightarrow \quad = \sum_{i=1}^{n} \sum_{j=1}^{n-i} j - i \leq \sum_{i=1}^{n} \sum_{j=1}^{n-i} n$$

$$= \sum_{i=1}^{n} (n-i) \cdot n \leq \sum_{i=1}^{n} n^2$$

$$= n \cdot n^2$$

$$\leq \Theta(n^3)$$

An algorithm is said to run in cubic time if the running time of the three loops is proportional to the cube of n.

When n triples, the running time increases by $n \cdot n \cdot n$.

Therefore, the time complexity of the above algorithm is $\Theta(n^3)$.

$\boxtimes$

Problem 6. Let f be the function which takes as input an array A of natural numbers, and returns the number of indices i such that $A[i+1]$ is the square of $A[i]$.

Write an algorithm to compute f.

The file `data.txt` consists of rows, where each row has some integers separated by spaces.

The submission file `submission.txt` should consist of rows, where the i-th row is the value of f applied to the i-th row of `data.txt`.

code:

```python
# Homework 1 problem 6
# Written by Jiayu Li

def f(A):
    cnt = 0
    # 0 ~ n-2
    for i in range(0, len(A) - 1):
        a = A[i]
        b = A[i + 1]
        if a > b:
            continue
        if a * a == b:
            cnt += 1
    return cnt


# open file in read mode
file = open('data.txt', 'r')
# open output file in write mode
output_file = open('submission.txt', 'w')

# Get all lines of a file
lines = file.readlines()
# iterate
# line refers to each line of the file
for line in lines:
    # Convert segmented text to integer
    # then turn into a list
    A = list(map(int, line.split()))
    # call function f and get the result
    cnt = f(A)
    # print the result for each line
    output_file.write(str(cnt) + '\n')
    print(cnt)

# Close file
file.close()
output_file.close()
```