C S 313E Assignment-4

Thomas Kuo, Mark Panjaitan

TOTAL POINTS

90 / 100

QUESTION 1

1 Task-1 **15 / 15**

✓ **- 0 pts** Correct

QUESTION 2

2 Task-2 **15 / 15**

✓ **+ 4 pts** (a) Correct

✓ **+ 3 pts** (a) Explained clearly

✓ **+ 4 pts** (b) Correct

✓ **+ 4 pts** (b) Explained clearly

+ 0 pts (b) Wrong

+ 0 pts (b) Not explained correctly

+ 0 pts (a) Wrong

+ 0 pts (a) Not explained correctly

QUESTION 3

3 Task-3 **30 / 30**

✓ **- 0 pts** Correct

QUESTION 4

4 Task-4 **10 / 20**

✓ **+ 5 pts** Identified first loop as $$O(n)$$

✓ **+ 5 pts** Identified second (outer) loop as $$O(n)$$

+ 5 pts Identified inner loop as $$O(log(n))$$

+ 5 pts Clearly states final answer as $$O(n\times log(n))$$

- 2 pts Need more structured explanation

+ 0 pts Incomplete

Nested loop has a running time of O(nlog(n)) as the inner loop starts from j till n, hence when j increases, the number of iterations of the inner loop will be reduced - giving it a O(log(n)) time complexity

QUESTION 5

5 Task-5 **20 / 20**

✓ **+ 8 pts** Identifies outer loop as $$O(n)$$

✓ **+ 8 pts** Identifies inner loop as $$O(n^2)$$

✓ **+ 4 pts** Clear explanation

+ 0 pts Incomplete

+ 0 pts Incorrect

CS 313E HW 4

Mark Panjaitan and Thomas Kuo

September 24 2021

1 Question 1

Below are the three graphs for question 1. You can find the descriptions for each graph further down. We stored the code for these graphs at the end of this PDF. The code is as follows:

- $f_1(n) = 2^{10}(n) + 2^{10}$ is red
- $f_2(n) = n^{3.5} - 1000$ is blue
- $f_3(n) = 100n^{(2.1)} + 50$ is green

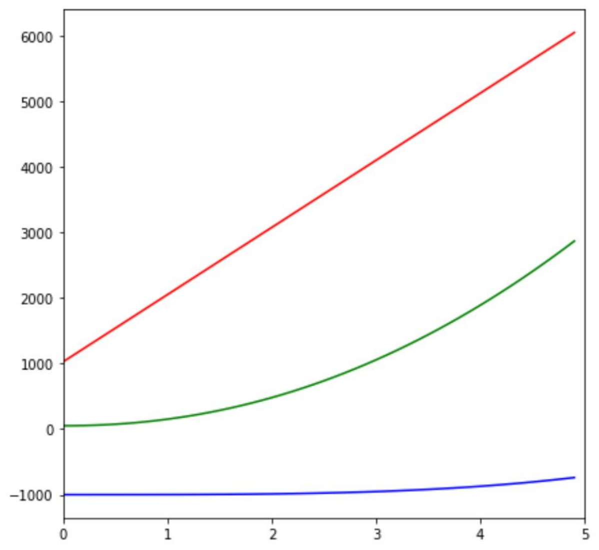

Figure 1: Graph with range $1 - 5$ for n

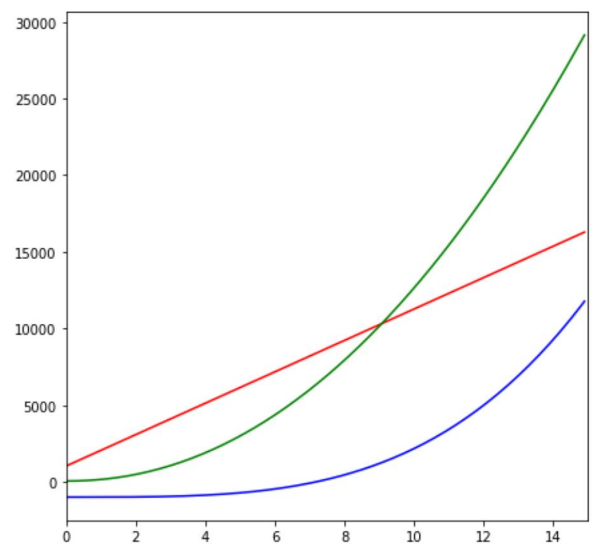

Figure 2: Graph with range $1 - 15$ for n

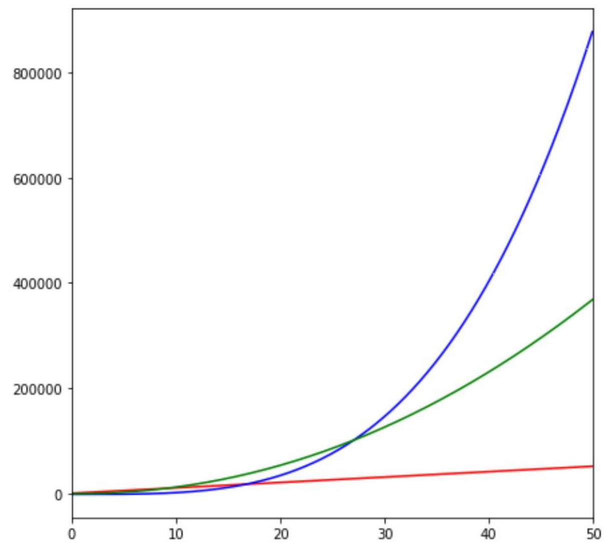

Figure 3: Graph with range $1 - 50$ for n

In the plot where the n value is 5, you can see that the linear function $f_1(n)$ has the highest value. However, when the plot goes until n = 10, we see $f_3(n)$ start to dominate. For larger n values, $f_2(n)$ has the largest value. So, we can say that asymptotically $f_2(n)$ is the largest, followed by $f_3(n)$ and then $f_1(n)$.

2 Question 2

2.1

$$\text{Is } 2^{(n+1.3)} = O(2^n)$$

Yes $2^{(n+1.3)}$ is equal to $O(2^n)$. The first rule of converting a $f(n)$ function into big O notation is "If f(x) is a sum of several terms, the highest order term is kept and others are discarded." If we apply this rule to our problem we must get rid of the 1.3 in the exponent. This gives us a big notation of $O(2^n)$.

2.2

$$\text{Is } 3^{(2n)} = O(3^n)$$

No $3^{(2n)}$ is not equal to $O(3^n)$. This is because if we simplify $3^{(2n)}$, we actually get 9^n. However, this means that the growth rate of $3^{(2n)}$ will always be higher than that of 3^n even if multiplied by a constant. Because of this, $O(3^n)$ is not the big O of $3^{(2n)}$ because it will not always be larger asymptotically.

3 Question 3

For each pair of functions $f(n)$ and $g(n)$, check if $f(n) = O(g(n))$

3.1

$$f(n) = (4(n))^{150} + (2(n) + 1024)^{400} \text{ vs. } g(n) = 20(n^{400}) + (n + 1024)^{200}$$

Yes in this case $f(n) = O((g(n))$ is true. This is because it fits the proof that $|f(n)| \leq c|g(n)|$ for a big enough n and some constant C. As $f(n)$ and $g(n)$ approach infinity:

- $f(n)$ becomes n^{400}
- $g(n)$ also becomes n^{400}

These two growth rates are the same so it fits the rule that $|f(n)| \leq c|g(n)|$. These equations fulfill $f(n) = O((g(n))$.

1 Task-1 **15 / 15**

✓ **- 0 pts** Correct

In the plot where the n value is 5, you can see that the linear function $f_1(n)$ has the highest value. However, when the plot goes until n = 10, we see $f_3(n)$ start to dominate. For larger n values, $f_2(n)$ has the largest value. So, we can say that asymptotically $f_2(n)$ is the largest, followed by $f_3(n)$ and then $f_1(n)$.

2 Question 2

2.1

$$\text{Is } 2^{(n+1.3)} = O(2^n)$$

Yes $2^{(n+1.3)}$ is equal to $O(2^n)$. The first rule of converting a $f(n)$ function into big O notation is "If f(x) is a sum of several terms, the highest order term is kept and others are discarded." If we apply this rule to our problem we must get rid of the 1.3 in the exponent. This gives us a big notation of $O(2^n)$.

2.2

$$\text{Is } 3^{(2n)} = O(3^n)$$

No $3^{(2n)}$ is not equal to $O(3^n)$. This is because if we simplify $3^{(2n)}$, we actually get 9^n. However, this means that the growth rate of $3^{(2n)}$ will always be higher than that of 3^n even if multiplied by a constant. Because of this, $O(3^n)$ is not the big O of $3^{(2n)}$ because it will not always be larger asymptotically.

3 Question 3

For each pair of functions $f(n)$ and $g(n)$, check if $f(n) = O(g(n))$

3.1

$$f(n) = (4(n))^{150} + (2(n) + 1024)^{400} \text{ vs. } g(n) = 20(n^{400}) + (n + 1024)^{200}$$

Yes in this case $f(n) = O((g(n))$ is true. This is because it fits the proof that $|f(n)| \leq c|g(n)|$ for a big enough n and some constant C. As $f(n)$ and $g(n)$ approach infinity:

- $f(n)$ becomes n^{400}
- $g(n)$ also becomes n^{400}

These two growth rates are the same so it fits the rule that $|f(n)| \leq c|g(n)|$. These equations fulfill $f(n) = O((g(n))$.

2 Task-2 **15 / 15**

- ✓ **+ 4 pts** **(a) Correct**
- ✓ **+ 3 pts** **(a) Explained clearly**
- ✓ **+ 4 pts** **(b) Correct**
- ✓ **+ 4 pts** **(b) Explained clearly**
- **+ 0 pts** (b) Wrong
- **+ 0 pts** (b) Not explained correctly
- **+ 0 pts** (a) Wrong
- **+ 0 pts** (a) Not explained correctly

In the plot where the n value is 5, you can see that the linear function $f_1(n)$ has the highest value. However, when the plot goes until n = 10, we see $f_3(n)$ start to dominate. For larger n values, $f_2(n)$ has the largest value. So, we can say that asymptotically $f_2(n)$ is the largest, followed by $f_3(n)$ and then $f_1(n)$.

2 Question 2

2.1

$$\text{Is } 2^{(n+1.3)} = O(2^n)$$

Yes $2^{(n+1.3)}$ is equal to $O(2^n)$. The first rule of converting a $f(n)$ function into big O notation is "If f(x) is a sum of several terms, the highest order term is kept and others are discarded." If we apply this rule to our problem we must get rid of the 1.3 in the exponent. This gives us a big notation of $O(2^n)$.

2.2

$$\text{Is } 3^{(2n)} = O(3^n)$$

No $3^{(2n)}$ is not equal to $O(3^n)$. This is because if we simplify $3^{(2n)}$, we actually get 9^n. However, this means that the growth rate of $3^{(2n)}$ will always be higher than that of 3^n even if multiplied by a constant. Because of this, $O(3^n)$ is not the big O of $3^{(2n)}$ because it will not always be larger asymptotically.

3 Question 3

For each pair of functions $f(n)$ and $g(n)$, check if $f(n) = O(g(n))$

3.1

$$f(n) = (4(n))^{150} + (2(n) + 1024)^{400} \text{ vs. } g(n) = 20(n^{400}) + (n + 1024)^{200}$$

Yes in this case $f(n) = O((g(n))$ is true. This is because it fits the proof that $|f(n)| \leq c|g(n)|$ for a big enough n and some constant C. As $f(n)$ and $g(n)$ approach infinity:

- $f(n)$ becomes n^{400}
- $g(n)$ also becomes n^{400}

These two growth rates are the same so it fits the rule that $|f(n)| \leq c|g(n)|$. These equations fulfill $f(n) = O((g(n))$.

3.2

$$f(n) = n^{1.4} + 4^n \text{ vs. } g(n) = n^{200} + (3.99^n)$$

In this case $f(n) = O((g(n))$ is not true. The rule for $f(n) = O((g(n))$ to be true is $|f(n)| \leq c|g(n)|$ for a big enough n and some constant C. The above pair of equations does not fit this rule. Specifically, $f(n)$ has a larger exponent base than $g(n)$.

- $f(n)$ has a growth rate of 4^n
- $g(n)$ has a growth rate of 3.99^n

As a result, for this pair of equations $|f(n)| > c|g(n)|$. This breaks the rule that $|f(n)| \leq c|g(n)|$. In conclusion, this pair of equations does not fulfill $f(n) = O((g(n))$.

3.3

$$f(n) = 2^{log(n)} \text{ vs. } g(n) = n^{1024}$$

In this case $f(n) = O((g(n))$ is true. The rule for $f(n) = O((g(n))$ to be true is $|f(n)| \leq c|g(n)|$ for a big enough n and some constant C. The above pair of equations fits this rule. Specifically, $g(n)$ grows larger than $f(n)$.

- $f(n)$ grows at a linear rate of 2n. This occurs because $2^{\log_2 n}$ is simplified to $2n$.
- $g(n)$ grows at an exponential rate of n^{1024}

As a result, for this pair of equations $|f(n)| < c|g(n)|$. This follows the rule that $|f(n)| \leq c|g(n)|$. In conclusion, this pair of equations fulfills $f(n) = O((g(n))$.

4 Question 4

The big O notation for this code is $O(N^2)$. We found this by first finding the function that represents the number of operations in the worst-case scenario. The algorithm has one operation at the top, a for loop that has 3 operations for n, and a nested for loop. This function is:

$$f(n) = 1 + 3n + (2n)(3n)$$

We then simplify this equation into:

$$f(n) = 1 + 3n + 6n^2$$

From there we apply the rules that finds the Big O notation of a function. so first we keep the highest order term and get rid of the other terms. So we now have:

3 Task-3 **30 / 30**

✓ **- 0 pts** Correct

3.2

$$f(n) = n^{1.4} + 4^n \text{ vs. } g(n) = n^{200} + (3.99^n)$$

In this case $f(n) = O((g(n))$ is not true. The rule for $f(n) = O((g(n))$ to be true is $|f(n)| \leq c|g(n)|$ for a big enough n and some constant C. The above pair of equations does not fit this rule. Specifically, $f(n)$ has a larger exponent base than $g(n)$.

- $f(n)$ has a growth rate of 4^n
- $g(n)$ has a growth rate of 3.99^n

As a result, for this pair of equations $|f(n)| > c|g(n)|$. This breaks the rule that $|f(n)| \leq c|g(n)|$. In conclusion, this pair of equations does not fulfill $f(n) = O((g(n))$.

3.3

$$f(n) = 2^{log(n)} \text{ vs. } g(n) = n^{1024}$$

In this case $f(n) = O((g(n))$ is true. The rule for $f(n) = O((g(n))$ to be true is $|f(n)| \leq c|g(n)|$ for a big enough n and some constant C. The above pair of equations fits this rule. Specifically, $g(n)$ grows larger than $f(n)$.

- $f(n)$ grows at a linear rate of 2n. This occurs because $2^{\log_2 n}$ is simplified to $2n$.
- $g(n)$ grows at an exponential rate of n^{1024}

As a result, for this pair of equations $|f(n)| < c|g(n)|$. This follows the rule that $|f(n)| \leq c|g(n)|$. In conclusion, this pair of equations fulfills $f(n) = O((g(n))$.

4 Question 4

The big O notation for this code is $O(N^2)$. We found this by first finding the function that represents the number of operations in the worst-case scenario. The algorithm has one operation at the top, a for loop that has 3 operations for n, and a nested for loop. This function is:

$$f(n) = 1 + 3n + (2n)(3n)$$

We then simplify this equation into:

$$f(n) = 1 + 3n + 6n^2$$

From there we apply the rules that finds the Big O notation of a function. so first we keep the highest order term and get rid of the other terms. So we now have:

$$O(6n^2)$$

Then we will follow the rule to get rid of the constants, which gives us our final answer of:

$$O(n^2)$$

5 Question 5

The big O notation for this code is $O(N^3)$. We found this by first finding the function that represent the worst case run time for the code we were given. The algorithm had one operation at the top and then a nested for loop that had a worst case scenario of running $n(n)$ times. This function is:

$$f(n) = 1 + n(n + 2n + 3n + 4n \ ... \ n(n))$$

From there we want to simplify the equation so now we have:

$$f(n) = 1 + (n^2 + 2n^2 + 3n^2 + 4n^2 \ ... \ n^3))$$

From there we apply the rules that finds the Big O notation of a function. so first we keep the highest order term and get rid of the other terms. So we now have:

$$f(n) = O(n^3)$$

Seeing as $f(n)$ has no terms that are products of several factors we have no more rules to apply to our function. So our final answer is that our function in big O notation is:

$$f(n) = O(n^3)$$

6 Code for Question 1

```
import math
import numpy as np
import matplotlib.pyplot as plt

#Graph size 1-5
msize = 5

t = np.arange(0, msize, 0.1)
```

4 Task-4 **10 / 20**

- ✓ **+ 5 pts** **Identified first loop as $$O(n)$$**
- ✓ **+ 5 pts** **Identified second (outer) loop as $$O(n)$$**
- **+ 5 pts** Identified inner loop as $$O(log(n))$$
- **+ 5 pts** Clearly states final answer as $$O(n\times log(n))$$
- **- 2 pts** Need more structured explanation
- **+ 0 pts** Incomplete

Nested loop has a running time of O(nlog(n)) as the inner loop starts from j till n, hence when j increases, the number of iterations of the inner loop will be reduced - giving it a O(log(n)) time complexity

$$O(6n^2)$$

Then we will follow the rule to get rid of the constants, which gives us our final answer of:

$$O(n^2)$$

5 Question 5

The big O notation for this code is $O(N^3)$. We found this by first finding the function that represent the worst case run time for the code we were given. The algorithm had one operation at the top and then a nested for loop that had a worst case scenario of running $n(n)$ times. This function is:

$$f(n) = 1 + n(n + 2n + 3n + 4n \text{ ... } n(n))$$

From there we want to simplify the equation so now we have:

$$f(n) = 1 + (n^2 + 2n^2 + 3n^2 + 4n^2 \text{ ... } n^3))$$

From there we apply the rules that finds the Big O notation of a function. so first we keep the highest order term and get rid of the other terms. So we now have:

$$f(n) = O(n^3)$$

Seeing as $f(n)$ has no terms that are products of several factors we have no more rules to apply to our function. So our final answer is that our function in big O notation is:

$$f(n) = O(n^3)$$

6 Code for Question 1

```
import math
import numpy as np
import matplotlib.pyplot as plt

#Graph size 1-5
msize = 5

t = np.arange(0, msize, 0.1)
```

```
plt.plot(t, t*(2**10) + 2**10 , ’red’, t, t**3.5 - 1000, ’blue’, t, 100*t**2.1 + 50,
’green’)

plt.xlim(0, msize)
plt.rcParams[”figure.figsize”] = (7,7)
plt.show()

#Graph size 1-15
msize = 15

t = np.arange(0, msize, 0.1)

plt.plot(t, t*(2**10) + 2**10 , ’red’, t, t**3.5 - 1000, ’blue’, t, 100*t**2.1 + 50,
’green’)
plt.xlim(0, msize)
plt.rcParams[”figure.figsize”] = (7,7)
plt.show()

#Graph size 1-50

msize = 50

t = np.arange(0, msize, 0.1)

plt.plot(t, t*(2**10) + 2**10 , ’red’, t, t**3.5 - 1000, ’blue’, t, 100*t**2.1 + 50,
’green’)

plt.xlim(0, msize)
plt.rcParams[”figure.figsize”] = (7,7)
plt.show()
```

5 Task-5 20 / 20

- ✓ **+ 8 pts** **Identifies outer loop as $$O(n)$$**
- ✓ **+ 8 pts** **Identifies inner loop as $$O(n^2)$$**
- ✓ **+ 4 pts** **Clear explanation**
- **+ 0 pts** Incomplete
- **+ 0 pts** Incorrect