Written Assignment

Q1. What is the big-Oh (O) time complexity for the following algorithm (shown in pseudocode) in terms of input size n? Show all necessary steps:

(a) For the time complexity, (note that C_i where i is positive integer is constant and n is input size)

1. count = 0 → C_1

2. for i = 0 to n-1 → C_2n (For loop take n times, constant time will be omitted on C_1)

3. sum = 0 → C_3n (because it is under for loop)

4. for j = 0 to n-1 → C_4n^2 (Under the for loop, there is another for loop so it will run n^2 times. n times by constant will be omitted)

5. sum = sum + A[0] → C_5n^2 (Calling A[0] and assigning new sum value, it is constant time * n^2 times)

6. for k = 1 to j → C_6 n^3 (This for loop is increasing k(1+2+3+…+n-1) = k*(n-1)*n/2 and the first for loop also multiplied, then the result will be c_6*n^3)

7. sum = sum + A[k] → c_7*n^3 (Since it is under 3rd for loop)

8. if B[i] == sum then count = count + 1 → c_8*n (it is under the first loop)

9. return count → c_9 (Return only once (constant time))

The sum of all the steps is $C'_1 + C'2*n + C'_3*n^2 + C'_4*n^3$ and it satisfies $C'_1 + C'2*n + C'_3*n^2 + C'_4*n^3 < c*n^3$ where any $c > C'_1 + C'2 + C'_3 + C'_4$ or $c = C'_1 + C'2 + C'_3 + C'_4 + 1$ and $n_0 = 1$ ($n \geq 1$)

Therefore, the answer is time complexity is **$O(n^3)$**

(b) Document a hand-run on MyAlgorithm for input arrays A = [1 2 5 9] and B = [2 29 40 57] and show the final output.

1. count = 0

2. input size is the same as size of Array which is 4 (A and B are the same input size given pseudo code)

3. Note that when ever for loop is executed, sum will be reset as i value increments. Therefore, we only can consider j for loop case. When j = 0, k loop does not execute because k = 1 and j =0 so sum is the same as A[0] which is 1.

4. Next, j = 1, then k loop executes 1 time so sum is 1 + A[0] + A[1] = 4.

5. Next, j = 2, then k loop executes 2 times so sum is 4 + A[0] + A[1] + A[2] = 12

6. Lastly, j = 3, then k loop executes 3 times so sum is 12 + A[0] + A[1] + A[2] = 29

7. j loop is done, and if B[i] == sum will be executed. However when i = 0, sum = 29 and B[0] = 2 so count will not increment.

8. when i = 1, do the same procedures 3 to 6 and sum is exactly the same value as 29 (since sume value becomes zero when i is incremented.). Therefore, count will be incremented.

9. when i = 2 and 3, do the same procedures 3 to 6 and since B[2] and B[3] are not the same as 29, count will not be incremented.

10. return count which is 1 and **output will be 1**.

Q2. Consider the following code fragments (a), (b) and (c) where n is the variable specifying data size and C is a constant. What is the big-Oh time complexity in terms of n in each case? Show all necessary steps.

Note that k_i is arbitrary constant for question 2.

(a) for (int i = 0; i < n; i = i + C)

for (int j = 0; j < 10; j++)

Sum[i] += j * Sum[i];

Since C is constant, for the first for loop, i = 0, one operation, i = i + C is running n/C times because of i < n. i < n is n/C + 1 operation. Total operation is $k_1n + k_2$ operations.

For the second for loop, it is constant time of loop, so we can simply conclude that it will be multiplication of k_3 times on first loop. Therefore, it is $k_4n + k_5$ operations (note that $k_1*k_3=k_4$ and $k_2*k_3 = k_5$).

Sum[i] += j * Sum[i] is actually Sum[i] = Sum[i] + j*Sum[i]. Therefore, sum[i] is calling constant times. For the simplicity, we denote another constant time k_6 times will be multiplied. Therefore, it is $k_7*n + k_8$ operations.

In total, we can simply denote $k'_1 + k'_2n$ and it satisfies $k'_1 + k'_2n < c*n$ where any $c > k'_1 + k'_2 +1$ and $n_0=1$ ($n\geq1$).

The big-Oh is thus **O(n)**.

(b) for (int i = 1; i < n; i = i * C)

for (int j = 0; j < i; j++)

Sum[i] += j * Sum[i];

i = 1 is 1 operation. For i = i * C, it will accumulate until the value of 1*C*C*…*C is reached to n because of i < n, so it means that when $\log_c n$ is reached, the loop will be stopped (Note that $C^x = n$, then $x = \log_c n$). Therefore, we can let it $k_1* \log_C n$ operations.

For the second loop, j = 0 is 1 operation and since j<i and j increment by 1, we can say that it is 1 + C + C*C (which is C^2)+ C*C*C (which is C^3)+ … + C^($\log_C n$) times. Note that it is combining from previous loop. The sum of this calculation is (1-C^($\log_c n$+1))/1-C = 1-Cn/1-C so we can denote k_2n operations.

In total, it is k_1*$\log_c n$ + $k_2 n$ and k_1*$\log_c n$ + $k_2 n$ < c*n where c > k_1 + k_2 and n_0 = 1 (n≥1)

Therefore, the big-Oh is **O(n).**

(c) for (int i = 1 ; i<n; i = i*2)

```
for (int j =0; j<n; j = j +2)

        Sum[i] += j*sum[i];
```

Similar to question (b), first for loop operates k_1*log(n) times because of similar reason to question (b) (just it is log(n) instead of $\log_c n$).

However, for the second loop, the loop will be done n/2 times. The second for loop will run n/2 times because j is incremented by 2. Therefore, n/2 times can be written k_2*n times. Note that regardless of previous loop it will run the same amount of time. Therefore, combining with previous loop, it will be k_3*nlog(n) times.

The last operation is Same as (a), (b), simple constant time, so combining with loop, it will be k_4*nlog(n).

In total, total operations can be denoted as k'_1*log(n) + k'_2*nlog(n) and it satisfies k'_1*log(n) + k'_2*nlog(n) < c*nlog(n) where c > k'_1 + k'_2 or c = k'_1 + k'_2 + 1 and n_0 = 1 (n≥1).

Therefore, the big-Oh is **O(n*log(n)).**

Q3. The number of operations executed by algorithms A and B are $12n^3 + 40n\log n$ and $5n^4 - 100n^2$ respectively. Determine an n 0 such that B is greater than A for n ≥ n_0.

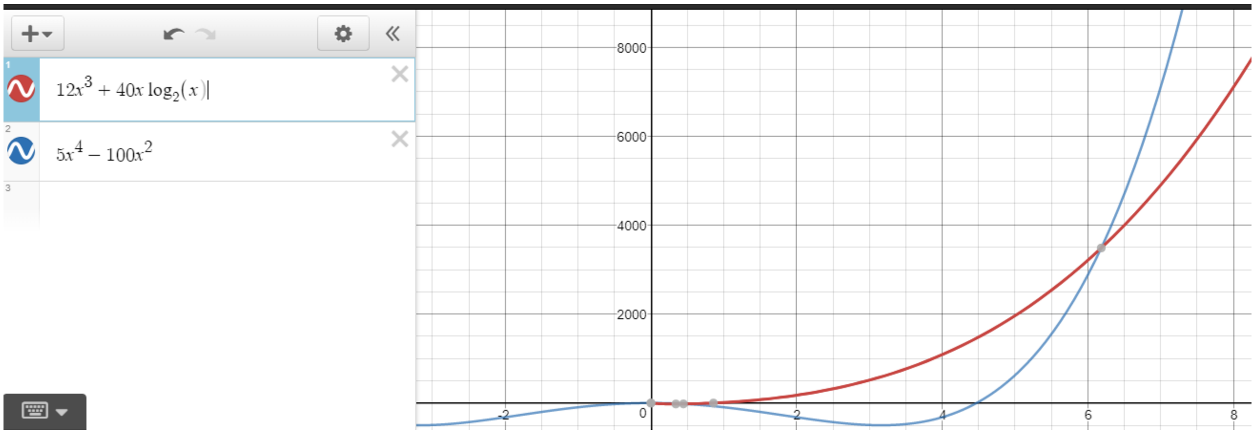
B-A should be greater than 0. $5n^4 - 100n^2 - 12n^3 - 40n\log n > 0$. We know that nlog(n) >= n when n >= 2. Please note that in algorithm log(n) = $\log_2 n$. Assume $5n^4 - 100n^2 - 12n^3 - 40n > 0$ then n >5.97, base on this, we test 6 or 7 (or more if it's applicable)

When n = 6, B-A = -332.39 and when n = 7, B-A = 2202.94

Therefore, B is greater than A for n ≥ n_0 where **n_0 = 7**

Also, we can simply prove by the graph. Check the below image. In this case, it is also n_0 = 7

You can also verify with below image.



Please note that if by mistake, if we denote log(n) as $\log_{10}(n)$ (normal mathematical intuition), Then the answer will be different or wrong, However, in algorithm calculation, log(n) should be $\log_2(n)$

Q4. Answer the following questions:

---

a) Show that if d(n) is O(f(n)) and e(n) is O(g(n)), then d(n) + e(n) is O(f(n) + g(n)).

By definition, $d(n) \le C_1*f(n)$, $n \ge n_1$ and $e(n) \le C_2*g(n)$, $n \ge n_2$. Adding them, then,

$d(n) + e(n) \le C_1*f(n) + C_2*g(n)$. Actually, O(f(n) + g(n)) is $C_3(f(n)+(g(n))$.

If we denote $C_3 = C_1 + C_2$. We get

$d(n) + e(n) \le C_1*f(n) + C_2*g(n) \le (C_1 + C_2)* (f(n)+(g(n))$ where $n \ge \max(n_1,n_2)$ (Note that if max is not allowed, we can simply determine $n \ge n_1 + n_2$)

b) Show that if d(n) is O(f(n)) and e(n) is O(g(n)), then d(n) – e(n) is not necessarily O(f(n) - g(n)).

We can prove by counter example.

Assume that d(n) = 3x, then f(n) = n. e(n) = 2x, then g(n) = n as well. d(n) – e(n) = x; however, f(n)-g(n) = 0 which we denote in big-Oh as O(1). It shows that d(n) – e(n) = x so its big-Oh should be O(x), but f(n)-g(n) can be 0 in some cases so given statement is true.

c) Show that $2^{n+1} + n^2$ is $O(2^n)$

$2^{n+1} + n^2 \le g(n)$ where $g(n) = c*2^n$ where c = 5 and $n \ge 1$.

Therefore $g(n) = 2^n$ so its big-Oh is $O(2^n)$.

d) Show that f(n) = $\sum_{i=1}^{n} i^2$ is $O(n^3)$

$\sum_{i=1}^{n} i^2 = 1 + 4 + 9 + \ldots + n^2$. It can be shown as $n(n+1)(2n+1)/6 = (2n^3+3n^2+n)/6$

$(2n^3+3n^2+n)/6 \leq C*n^3$ where $C = 1$ and $n \geq 1$ so big-Oh is $O(n^3)$