Zachary Neeley
2/12/2020

0. Read Chapter 0 from the textbook.

1. Compute running time of the following piece of code. For convenience the lines are numbered. Use cost and times columns to compute the running time. (Cost of a line is the amount of time that is required to execute the line, e.g. constant c_1. Times is the number of times the line is executed.) After you get the exact expression for the running time, represent it in big-Oh notation (O-notation).

```
1.  int maxSum = 0;
2.  for( int i = 0; i < n; i++ )
    {
3.      int thisSum = 0;
4.      for( int j = i; j<n; j++ )
        {
5.          thisSum += a[j];
6.          if( thisSum > maxSum )
7.              maxSum = thisSum;
        }
    }
```

The running time of this algorithm is O(n^2).

2. Explain clearly why the statement "The running time of algorithm A is at least $O(n^2)$" does not make sense.

The reason why the statement of "The running time of algorithm A is at least O(n^2) does not make sense is because the big-Oh notation is used to calculate the worst-case running time. To get the running at the best case you will need to calculate Ω notation.

3.

a) Let $f(n) = 6n^2 - 100n + 44$ and $g(n) = 0.5n^3$. Prove that $f(n) = O(g(n))$ using the definition of Big-O notation. (You need to find constants c and n_0).

6n^2-100n+44 =< 6n^2+44 =< 6n^3 <= C * 0.5n^3

c = 15, n0 = 1 so f(n) = O(g(n))

b) Let $f(n) = 3n^2 + n$ and $g(n) = 2n^2$. Use the definition of big-O notation to prove that

i. $f(n) = O(g(n))$ (you need to find constants c and n_0) and
ii. $g(n) = O(f(n))$ (you need to find constants c and n_0).

Conclude that $f(n) = \Theta(g(n))$.

i) 3n^2 + n <= 3n^2 + n^2 <= 4n^2 C = 2, N0 = 1 so 4(1)^2 <= 2 * 2(1)^2 proving f(n) = O(g(n)).

ii) 2n^2 <= 3n^2 + n if C1 = 1 and N0 = 1 so 1 * 2(1)^2 <= 3(1)^2 + 1 showing C1 * g(n) <= f(n) while C2= 20 showing that 10 + 2(1)^2 >= 3(1) ^2 + 1 showing that C2 * g(n) >= f(n). Knowing this information you can conclude that f(n) = Θ(g(n)).

4. Do Exercise 0.1 (a), (b), (c), (e), (f), (h), (k), (l), (n), (o) from p.8 of the textbook. (You may use $\lim_{n\to\infty} (f(n) / g(n))$ to help you find the answers).

a) f = Θ(g)	b) f = O(g)
c) f = Θ(g)	e) f = O(g)
f) f = Θ(g)	h) f = Ω(g)
k) f = Ω(g)	l) f = O(g)
n) f = Θ(g)	o) f = O(g)

5. Order the following 16 functions by asymptotic growth rate from lowest to highest. If any are of the same order then circle them on your list.

$5n-6$, 3, $5n+n^3$, $\sqrt{n}$, $n^{2.01}$, $3^{\log_2 n}$, $4\log n$, $n!$, $\log n^3$, $n^{1/3}$, $3n^3-5n+n^4$, $n \log n +4n$, $10n^4$, 4^n, 2^n, 4^{n+1}.

Note: When comparing two functions f(n) and g(n) you may use $\lim_{n\to\infty} (f(n) / g(n))$ to compare their asymptotic growth rates.

lowest: 3
n^⅓
log n^3
4 log n
sqrt n
5n-6
n log n + 4n
3^log2 n
n^2.01
5n+n^3
2^n
n!

3n^3-5n+n^4
4^n
4^n+1
highest: 10n^4

6. The purpose of this exercise is for you to experience a difference between exponential and polynomial algorithms.
a) Implement the exponential algorithm to determine n^{th} Fibonacci number from page 3 of the textbook. Implement the polynomial algorithm to determine n^{th} Fibonacci number from page 4 of the textbook. Write a program that would compute execution time (as measured by the system clock) of each of the two algorithms for various values of n.

b) Report obtained execution times of the two algorithms for different values of n (at least 10 different values). Your goal is to demonstrate the difference in execution times of the algorithms when n grows. Write obtained execution times in a table like the following:

n	Exponential Algorithm	Polynomial Algorithm

c) Experiment with your program or write another program to answer the following questions. What is the largest value of n for which n^{th} Fibonacci number can be computed in less than 10 milliseconds by the exponential algorithm? What is the largest value of n for which n^{th} Fibonacci number can be computed in less than 10 milliseconds by the polynomial algorithm?

n	Exponential Algorithm	Polynomial Algorithm
20	995600	7900
21	107500	1400
22	112900	5800
23	173400	1700
24	320000	1500
25	419500	1400
26	712600	1500
27	1189000	1500
28	1781700	1500
29	2968400	1500

Part C) The largest number of Fibonacci I would run within 10 milliseconds is 31.