HW7 Efficiency and Asymptotic Complexity

Name: Katrina Li

Question 1

a. Snippet 1: The statement is printed n times in every outer loop so is printed $2\mathrm{n} * \mathrm{n} = 2n^2$ times.

$$\mathrm{f(n)} = 2n^2 \in O(n^2)$$

So the number of times the statement is printed grows in n^2.

b. Snippet 2: The statement is printed 5 times in every outer loop so is printed $5 * \mathrm{n} = 5\mathrm{n}$ times in total.

$$\mathrm{f(n)} = 5\mathrm{n} \in \mathrm{O(n)}$$

So the number of times the statement is printed grows in n.

c. Snippet 3: The statement is printed n times in every outer loop so is printed $(\mathrm{n}+3) * \mathrm{n} = n^2 + 3\mathrm{n}$ times in total.

$$\mathrm{f(n)} = n^2 + 3\mathrm{n} \in \mathrm{O}(n^2)$$

So the number of times the statement is printed grows in n^2.

d. Snippet 4: The statement is printed 0 times because j is always less than i in the inner 'for' loop so the 'if' statement will never be true.

$$\mathrm{f(n)} = 0 \in \mathrm{O}(0)$$

So the number of times the statement is print never grows with the size of n.

e. Snippet 5: The statement is printed $7\mathrm{n} + \mathrm{n} = 8\mathrm{n}$ times because the first one prints 7n times and the second one prints n times.

$$\mathrm{f(n)} = 8n \in \mathrm{O(n)}$$

So the number of times the statement is printed grows in n.

Question 2

$f(A) = 80n^2 + 5n \in O(n^2)$

$f(B) = 9n + 50000 \in O(n)$

Let $80n^2 + 5n < 9n + 50000$ so that $n = 25.02 \approx 25$. Since n is a positive integer,

$$1 \leq n \leq 25$$

Question 3

Since for each item in array we need $O(nlogn)$ and we have n items in the array, the asymptotic order is to multiply these two values.

$$n \log n * n = n^2 \log n$$

Question 4

Assumption: When transferring analysis from Big-O to Big-Theta for each program, all the expressions would be monomial. E.g. Program C could be $\Theta(3\mathrm{n}^3)$ but not $\Theta(\mathrm{n}^3 + n)$.

A. Suppose that the time to solve a problem for A could be expressed in $\mathrm{a} * \log(\mathrm{n})$ where a is a constant. Now I know that $\mathrm{a} * \log(300) = 5$ so a = 2.02. Thus $\mathrm{a} * \log(600) = 2.02 * 2.78 = 5.61$ seconds (approximated to 2 digit nums).
B. Similar to (A), if $\mathrm{b} * 300 = 5$ then $\mathrm{b} * 600 = 10$ seconds.
C. Similar to (A), Big-Theta is approximately $\Theta(\mathrm{c} * n^3)$. $\mathrm{c} * 300^3 = 5$ so c = 1.85E-7. $\mathrm{c} * 600^3 = 40$ seconds. The time becomes 2^3 greater when size n is 2 times greater.
D. Similar to (A), Big-Theta is approximately $\Theta(\mathrm{d} * 2^n)$. $\mathrm{d} * 2^{300} = 5$ so d = 2.45E-90. $\mathrm{d} * 2^{600} = 1.02 \times 10^{91}$ seconds (approximated to 2 digit nums).

Question 5

The (alternative) definition of O(n): a function f(n) is in O(g(n)) if $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ exists and is finite.

a. $\lim_{n\to\infty} \frac{\frac{1}{5}n+30}{n} = \frac{1}{5}$. This limit exists and is finite.
b. $\lim_{n\to\infty} \frac{7n^3+2^n}{2^n} = 1$. This limit exists and is finite.
c. $\lim_{n\to\infty} \frac{8n^2+3n}{n^3} = \lim_{n\to\infty} \frac{8+\frac{3}{n}}{n}$ if I divide n^2on both the nominator and denominator. $\lim_{n\to\infty} \frac{8+\frac{3}{n}}{n} = 0$ and is finite. Therefore $8n^2 + 3n \; is \; in \; O(n^3)$.

Question 6

$\lim_{n\to\infty} \frac{7n^2+3n+21}{n^2} = 7$ and is finite.

There exists c = 1 and $N_0 = 0$ such that for all $n \geq N_0$, $\mathrm{c} \cdot \mathrm{n} \leq 4n^2 + 21n + 8$.

Therefore, $\mathrm{f(n)} = 7n^2 + 3n + 21 \in O(n^2) \; and \; g(n) = 4n^2 + 21n + 8 \in \Omega(n)$.

$$\text{there exists } \mathrm{h(n)} = n^2 \; such \; that \; f(n) \in \Theta(\mathrm{h(n)})$$

Proof: $\mathrm{f(n)} \in O\big(h(n)\big) = O(n^2)$ because there exists $\mathrm{c1} = 8$ and $N_{01} = 7$ such that for all $\mathrm{n} \geq N_0$, $f(n) \leq c \cdot h(n)$.

$\mathrm{f(n)} \in \Omega\big(h(n)\big) = O(n^2)$ because there exists c2 = 7 and $N_{02} = 0$ such that for all $\mathrm{n} \geq N_0,\ f(n) \geq c \cdot h(n)$.
Thus there exists $\mathrm{h(n)} = n^2$ *such that* $f(n) \in \Theta(\mathrm{h(n)})$.

Question 7

$\mathrm{n}^2 > n$ *for all* $n \in Z^+$*(positive integers)*. So $\min(\mathrm{n}, \mathrm{n}^2) = n$ *and* $\max(\mathrm{n}, \mathrm{n}^2) = n^2$.

a. $\mathrm{f(n)} = n$ and $\mathrm{g(n)} = n^2$. $\mathrm{h(n)} = \mathrm{O}(\mathrm{n}^2 + \mathrm{n}) \in \mathrm{O}(\mathrm{n}^2) \notin O(\mathrm{n}) = \mathrm{O}(\min(\mathrm{f(n)}, \mathrm{g(n)}))$
b. $\mathrm{f(n)} = \mathrm{n}$ and $\mathrm{g(n)} = n^2$. $\mathrm{h(n)} = \mathrm{O}(\mathrm{n}^2 \cdot \mathrm{n}) \in \mathrm{O}(\mathrm{n}^3) \notin \mathrm{O}(\mathrm{n}^2) = \mathrm{O}\big(\max(\mathrm{f(n)}, \mathrm{g(n)})\big)$

Question 8

Suppose that I create an array of size 1 at the beginning and I want to add c objects in total. For the first method – double the size of the array every time I need to add additional capacity, I would need to create a new array in a total of $\log_2 n$ times ($\log n$ times) but if I use the second method, even if the "constant amount" is really big (I say 1000) I would need to create a new array in a total of n/1000 times. When n becomes greater, $\mathrm{O}(\log n) < O\left(\frac{n}{1000}\right) = O(n)$ therefore doubling the size of the array every time I needed to add additional capacity makes more sense as it saves time and operations.

Question 9

Linear search:

The graph for linear search looks like a straight line with a slope of $\frac{160000}{250000} = 0.64$. The big-O cost of this search is $\mathrm{O}(0.64\mathrm{n}) \in \mathrm{O}(\mathrm{n})$.

Hash Map Search:

The graph for hash map search is amazing because the slope is almost 0. As I recall from intro class, hash maps are made so that searching for each key takes constant time, I think the big-O cost of hash map search is $\mathrm{O}(30) \in \mathrm{O}(1)$.

Binary Search:

The average running time for binary search doesn't increase as much as linear search does. By recalling from Friday's class, the algorithm always divides the ArrayList and compares the middle number with the aimed number. The big-O cost for binary search is thus $\mathrm{O}(\log_2 n) \in O(logn)$.

Tree Map Search:

The graph for tree map search is really similar to binary search, so I think the big-O cost for tree map search is $\mathrm{O}(\log_2 n) \in O(logn)$.