CS-2383 A1

1) C: n_log-n and log functions are smaller than quadratic functions.

D: is a linear function and linear functions are smaller than quadratic functions

E: If the upper limit is n^2, the running time function can be the lower limit $\Omega_n(n)$

2) $\log n, n, n \log n, n^2, n^2 + \log n, n - n^3 + 7n^5, 3^n, n!, n^n$

side note: $n - n^3 + 7n^5$ is $O(n^5)$

3) A) $10n^3 + 8n^2 + 6n + 2$ Highest order term: $10n^3$

No negative terms to drop

$10n^3 + 8n^2 + 6n + 2 \leq 10n^3 + 8n^3 + 6n^3 + 2n^3$ replace lower order terms with higher ones

$\leq 26n^3$

Let $c = 26$, $n_0 = 1$

We have $10n^3 + 8n^2 + 6n + 2 \leq c \cdot n^3$, for all $n \geq n_0 = 1$

$\therefore 10n^3 + 8n^2 + 6n + 2$ is $O(n^3)$

~~B) $3(3n+2)^7 + 4$~~

B) $3(3n+2)^7 + 4(2n+3)^5 + n \log n$ is $O(n^7)$

$\leq 3(3n+2n)^7 + 4(2n+3n)^5 + n \log n$

$\leq 3(5n)^7 + 4(5n)^5 + n \log n$

$\leq 3 \cdot 5^7 \cdot n^7 + 4 \cdot 5^5 \cdot n^5 + n \log n$

$\leq 3 \cdot 5^7 \cdot n^7 + 4 \cdot 5^5 \cdot n^7 + n^7$

$\leq (3 \cdot 5^7 + 4 \cdot 5^5 + 1) n^7$

Let $c = (3 \cdot 5^7 + 4 \cdot 5^5 +)$, $n_0 = 1$

We have $3(3n+2)^7 + 4(2n+3)^5 + n \log n \leq cn^7$,

For all $n \geq n_0 = 1$

$\therefore 3(3n+2)^7 + 4(2n+3)^5 + n \log n$ is $O(n^7)$

C) $3n^5 - 9n^4 \log_2 n + 7n^3 - 15n^2 \leq c \cdot n^5$

$3n^5 - 9n^4 \log_2 n + 7n^3 - 15n^2 \leq 3n^5 + 7n^3$ drop negative terms

$3n^5 - 9n^4 \log_2 n + 7n^3 - 15n^2 \leq 3n^5 + 7n^5$ replace lower order terms

$\leq 10n^5$

Let $c = 10$, $n_0 = 1$

We have $3n^5 - 9n^4 \log n + 7n^3 - 15n^2 \leq c \cdot n^5$, for all $n \geq n_0 = 1$

$\therefore 3n^5 - 9n^4 \log n + 7n^3 - 15n^2$ is $O(n^5)$

D) n^4 is $O(10^6 n^3 \log_2 n)$

$n^4 \leq c \cdot 10^6 n^3 \log_2 n$ $(\div n^3)$

$n \leq c \cdot 10^6 \log_2 n$

There are no such n_0 and c (constant) that can make a log function greater than or equal to (the upper limit of) a linear function.

$\therefore n^4$ is not $O(10^6 n^3 \log_2 n)$

E) $2^{\log_{10} n}$ is $O(n^{1/3})$

$2^{\log_{10} n} \leq c \cdot n^{1/3}$

$2^{\frac{\log_2 n}{\log_2 10}} \leq c \cdot n^{1/3}$

$2^{\frac{1}{\log_2 10}} \leq c \cdot n^{1/3}$

$\log_2 10 = 3.32 > 3$, so, $\frac{1}{\log_2 10} < \frac{1}{3}$

Let $c = 1$ and $n_0 = 1$

We have $2^{\log_{10} n} \leq c \cdot n^{1/3}$ for all $n \geq n_0 = 1$

$\therefore 2^{\log_{10} n}$ is $O(n^{1/3})$

4) The program assigns 0 to k and 1 to b. The program, then, goes through a loop in which it increments k by 1 and multiplies b by a and stores the result in b. This loop keeps iterating as long as $k < n$. Once this condition is not met anymore, the program will exit the loop and execute the instruction after. In this case, it is to return b.
The worst case running time; also only running time:

k ← 0	1
b ← 1	1
while k < n do	n
k ← k+1	2n
b ← b * a	2n
return b	1

Total: $5n + 3$ ← running time function

$5n + 3$ is $O(n)$

5) The program assigns (the value of) n to k, 1 to b, and (the value of) a to c. The program, then, enters a loop in which it checks if k is even by dividing it by 2 and checking if there is a remainder. IF there is not a remainder, then k is even. In this case, it divides (the value of) k by 2 and stores the result in k and squares (the value of) c by multiplying it by itself and stores the result in c. If k is odd, (the value of) k is deducted by 1 and stored in k, then b is multiplied by c and the result is stored in b. This loop keeps going as long as $k>0$. Once this condition is broken, the loop will be skipped and (the value of) b will be returned.

```
k ← n                                  1
b ← 1                                  1
c ← a                                  1
while k > 0 do                         log n
    if k mod 2 = 0 then                2 log n
        k ← k/2                        2 log n
        c ← c * c                      2 log n
    else
        k ← k - 1                      2 log n
        b ← b * c                      2 log n
return b                               1
                                       11 log n + 4
```

Worst case: $O(\log n)$

6) This program assigns 1 to F and 1 to j. Then, the program goes in a loop that keeps iterating as long as $f = 1$ and $j \leq (n-1)$. If any of those conditions is broken the loop will be skipped. Inside the first loop, 0 is assigned to f. Then the program goes into another loop insid the first in which o will be assigned to i which will be incremented by 1 after every iteration. Every iteration i will be evaluated to $n-(j+1)$. When $i = n-(j+1)$ the loop will be skipped. Inside the 2nd loop we will check of the value present at A[i] is bigger than that present at A[i+1]. If that is the case those values will swap positions, because this program is sorting arrays from the smallest to the largest, and assign 1 to f. After that j will be incremented by 1 and the result will be stored in j, regardless of the result of the previous check. After the program exits both loops it will return the sorted array. The best

case scenario is the array is to be already sorted. The worst case scenario is for the array to be arranged from the largest to the smallest

2 integer array

```
Program:
f ← 1                                   BCS     WCS
j ← 1                                    1       1
while f = 1 and j ≤ (n-1) do             1       1
                                        4(1)    4n
    f ← 0                               4(1)    n-1
    for i ← 0 to n-(j+1) do            16(1)   2(n-1)+1
        if A[i] > A[i+1]                4(1)    4(n-1)
            swap A[i] and A[i+1]        2(1)    2(n-1)
            f ← 1                      16(1)   16(n-1)
    j ← (j+1)                           1      2(n-1)

    {increment i}                       1      n-1
return A                                1       1
                                       O(1)    O(n)
                                    constant
                                    function
```

7) A) Correct, because addition won't change the highest order (power of n)

B) Correct, since $f(n)$ is $O(h(n))$ and $g(n)$ is $O(h(n))$, the result of multiplying $f(n)$ and $g(n)$ will always be less than or equal to the upper limit of the result of multiplying $h(n)$ by $h(n)$