Neha Maddali

Problem 1:

a. $f(n) = \Theta(g(n))$
b. $f(n) = \Omega(g(n))$
c. $f(n) = \Omega(g(n))$
d. $f(n) = \Theta(g(n))$
e. $f(n) = O(g(n))$
f. $f(n) = O(g(n))$
g. $f(n) = \Omega(g(n))$
h. $f(n) = \Omega(g(n))$
i. $f(n) = \Theta(g(n))$
j. $f(n) = \Omega(g(n))$
k. $f(n) = \Omega(g(n))$
l. $f(n) = \Omega(g(n))$

Problem 2: (sum of squares citation)

a. $\Sigma^n_{i=1} i^2$ is $O(n^3)$

   $\Sigma^n_{i=1} i^2 = f(n) = \frac{n(n+1)(2n+1)}{6}$

   $\Sigma^n_{i=1} i^2 \leq \Sigma^n_{i=1} n^2$ since $i \leq n$

   $= n^2 \, \Sigma^n_{i=1} 1$ taking n^2 out of the sum

   $= n^3$ since $\Sigma^n_{i=1} 1 = n$

   $\Sigma^n_{i=1} i^2 = f(n) \leq \frac{n(n+1)(2n+1)}{6} \leq n^3$

   So $\Sigma^n_{i=1} i^2$ is $O(n^3)$ for $c = 6$ and $n_0 > 1$

b. $\Sigma^n_{i=1} i^2 = \Omega(n^3)$

   $= \frac{n(n+1)(2n+1)}{6}$

   $= \frac{2n^3+3n^2+n}{6}$

   $\Sigma^n_{i=1} i^2 > 2n^3 / 6$ for all $n \geq 1$ and $c = 2/6$

   $\Omega(g(n))$ - $f(n) \geq c * g(n)$

   $\Sigma^n_{i=1} i^2 \geq 2/6 * n^3$

   $\Sigma^n_{i=1} i^2$ is $\Omega(n^3)$ with $c=2/6$ because $g(n) = 2/6 * n^3$ is the lower bound

c. *Problem i:*

   $(n+a)^b = O(n^b)$

   $(n+a)^b \leq (2n)^b$ for $n \geq |a|$

   $= 2^b n^b$

   $= cn^b$ for $c=2^b$

   So $(n+a)^b$ is $O(n^b)$

   Problem ii:

   $(n+a)^b = \Omega(n^b)$

   $(n+a)^b \geq (n - |a|)^b$ where $n > 0$

   $\geq (c'_2 n)^b$ for $c'_2 = ½$ where $n \geq 2|a|$ as $n/2 \leq n - |a|$, for $n \geq 2|a|$

   So $(n+a)^b$ is $\Omega(n^b)$

Therefore for any real constants, where b > 0, $(n+a)^b = \Theta(n^b)$

Problem 3:

a. $\frac{5n^2logn + 3n^2 + 7nlogn + 9n + 2}{n^2logn} \leq c$ for all $n \geq 1$

5+3+7+9+2 = 26 = c

$26 > \frac{5n^2logn + 3n^2 + 7nlogn + 9n + 2}{n^2logn}$ for all $n \geq 1$

$\frac{5n^2logn + 3n^2 + 7nlogn + 9n + 2}{n^2logn} \leq 26$ for all $n \geq 1$

If $5n^2logn + 3n^2 + 7nlogn + 9n + 2 \leq 26 * n^2logn$ for all $n \geq 1$ then f(n) = O(g(n))

$5n^2logn + 3n^2 + 7nlogn + 9n + 2 \leq 26n^2logn$ for all $n \geq 1$

$3n^2 + 7nlogn + 9n + 2 \leq 21n^2logn$ for all $n \geq 1$

$0 \leq 21n^2logn - 3n^2 + 7nlogn + 9n + 2$ for all $n \geq 1$

$0 \leq 8$ where n = 1 which is a true statement.

Therefore, $5n^2logn + 3n^2 + 7nlogn + 9n + 2$is in $O(n^2logn)$

b. $f(n) = 3n^2\sqrt{n} + 5nlogn + 7$

$g(n) = n^2$

f(n) > c * g(n) for all $n \geq k$

$3n^2\sqrt{n} + 5nlogn + 7 > c*n^2$ for all $n \geq k$

$3\sqrt{n} + 5nlogn + 7 > c$ for all $n \geq k$

Supposed $3\sqrt{n} + 5nlogn + 7 > c$ for all $3\sqrt{n} + 5nlogn + 7 > c + 1$ which is true

Therefore, f(n) is in ω(g(n)) which implies $3n^2\sqrt{n} + 5nlogn + 7$is in $\omega(n^2)$ which implies $3n^2\sqrt{n} + 5nlogn + 7$ is not in $O(n^2)$

Problem 4:

a. The first loop runs from 1 to n so n-1 times which is in linear time. The j loop runs i to n. Lets say n-i+1 = m times which also runs in linear time which is proportional to n. Therefore $m = c_2 * n$ where c_2 is some constant. The k loop runs from i to j for a large value of n where n lands to infinity. |i-j+1| = p. $p = c_3 * n$ where c_3 is some constant greater than zero.

$= n^2 + n*(c_2*n)*(c_3*n)$ where $c_2, c_3 > 0$

$= n^2 + c_4*n^3$ where $c_4 = c_2+c_3 > 0$

Since n^3 dominates, the runtime of SubTotal(A) is $\Theta(n^3)$

b. The definition of O notation on Θ notation assumes that n trends to infinity. So for SubTotal(A) there is no best case scenario since the loops will index every element in A to fill the upper triangle of matrix B. In every case, all values of B will be filled and each loop will run c_r*n times. The total algorithm will run in proportion to n^3 times because there are 3 nested loops.

c. NewSubTotal(A)

```
        for i = 1 to n do
        sum = 0
        for j = 1 to n do
            if i < j
                B[i,j] = 0
            else
                sum = sum + A[j]
                B[i,j] = sum
    Return B
```

Since both i and j go from 1 to n, both of them run n times so the new algorithm has a runtime of $\Theta(n^2) = O(n^2) = \Omega(n^2)$. Moreover, the limit of n to infinity of $\Theta(n^2) / \Theta(n^3) = 0$

Problem 5:

Input: heap H length n, int key k, int index i, solution array S at length n

Output: solution array S with size r that has all numbers greater than or equal to k from heap H

```
HeapAlgo(H, k)
    return HeapAlgoHelper(H, R, O, S)
HeapAlgoHelper(H, k, i, S)
    if(H[i] ≥ k && i < H.length)
        S.append(H[i])
        HeapAlgoHelper(H, k, 2i, S)
        HeapAlgoHelper(H, k, 2i, S)
    return S
```

I used the divide and conquer method to develop an algorithm. HeapAlgoHelper checks if the current node is greater than k and if that is true it adds it to the solution array S. Then it calls itself on the left child and the right child. For the last step, the algorithm returns the solution array S. Each step of the function will run in constant time. In this algorithm the recursion happens S.size = r times because every element will only be visited once and there are r elements in S. With each visit, there is one append statement. The method will call itself r times which is why it runs in O(r) time. However, the subtree where the parent is less than k is not traversed because the method returns just the solution array and skips the recursive step for H[i] < k. So $r \le n$ because not all less than nodes are traversed. The part of the left subtree is 2i and the right is 2i + 1.