Sunny Ka Patel
SDK438
11267665
CMPT 280 – Assignment 2

Question 1():

(a) $O(n^4 \log_{280} n)$

(b) $O(n^4)$

(c) $O(n \log_2 n)$

Question 2():

(a) **1.** False
2. False
3. True
4. True

(b) $\Theta(n^3)$

Question 3():

(a) $O(n^2)$

(b) $O(n^2 2^n)$

(c) $O(n^3)$

(d) $O(\mathit{max}(n^2 \log_2 n^2, m))$
because we don't know the relation between *n* and *m*.

Question 4():

(a) **Inner loop:**
- Number of statements: 2
- Number of executions: n
- Total cost: 2n + 1 (the +1 when loop condition gets false)

Outer loop:
- Number of statements: 1 + (inner loop) = 1 + (2n + 1) = 2n + 2
- Number of executions: n
- Total cost: = n(2n + 2) + 1 (the +1 when the loop condition gets false)
  $= 2n^2 + 2n + 1$, which is $O(n^2)$

(b) The above equation is true for both the best case and worst case, thus $\Theta(n^2)$.

Question 5():

(a) Inner loop:

- Number of statements: 2
- Number of executions: n - (i + 1) = n - i - 1
- Total cost: = 2(n - i - 1) + 1 (the +1 when the loop condition gets false)
  = 2n - 2i - 1
- Here, the total cost for the inner loop depends on the value of i.

Outer loop:

- Number of statements: 1 + (inner loop) = 1 + (2n - 2i - 1) = 2n - 2i
- Total cost: = 1 + $\sum_{i=0}^{n-1} 2n - 2i$ (the +1 when the loop condition gets false)

$= 1 + \sum_{i=0}^{n-1} 2n - \sum_{i=0}^{n-1} 2i$

$= 1 + 2n \sum_{i=0}^{n-1} 1 - 2\sum_{i=0}^{n-1} i$

$= 1 + 2n \cdot (n) - 2\left[\frac{(n-1)n}{2}\right]$

$= 1 + 2n^2 - (n^2 - n)$

$= 1 + 2n^2 - n^2 + n$

$= n^2 + n + 1$

which is $O(n^2)$.

(b) The above equation is true for both the best-case and the worst-case, thus $\Theta(n^2)$.

Question 6():

(a) The active operation for the pseudocode is the condition for the inner while loop. Each time the inner loop is executed by the outer loop, the active operation is executed (n - i) times. I did not select the outer loop condition as the active operation, even though it is executed *n* times, because the inner loop condition is depended on the value of *i* due to which sometimes the inner loop condition executes *n* times. Thus, it is:

$= \sum_{i=0}^{n-1} n - i$

$= \sum_{i=0}^{n-1} n - \sum_{i=0}^{n-1} i$

$= n \cdot (n) - \left[\frac{(n-1)n}{2}\right]$

$= n^2 - \frac{1}{2}n^2 + \frac{1}{2}n$

$= \frac{1}{2}n^2 + \frac{1}{2}n$ which is $O(n^2)$.

(b) The above equation is same for the best-case and the worst-case, thus $\Theta(n^2)$.

Question 7():

- Even though the loop condition executes *n+1* times, the active operation for the given pseudocode should be the body of the while loop.
- The time-complexity of the binary-search function is *O(log(m))*, where *m* is the number of items in each array.
- Moreover, in the worst-case for binary search, the given *target* integer is at either ends of the array or the *target* integer does not exist in the array.
- Thus, the active operation executes *n* times.
- The time complexity for the given pseudocode must be: $O(n \cdot \log(m))$

Question 8():

Name: PriorityQueue<G>

Sets:

Q: set of all priority queues containing elements from G

G: set of items that can be in the priority queue

B: {true, false}

N_0: set of non-negative integers

Signatures:

newPriorityQueue<G>(n): N_0 —/> Q

Q.insert(g): G —/> Q

Q.isEmpty: —> B

Q.isFull: —> B

Q.maxItem: —/> G

Q.minItem: —/> G

Q.deleteMax: —/> Q

Q.deleteAllMax: —/> Q

Q.deleteMin: —/> Q

Q.frequnecy(g): G —> N_0

Preconditions: $\forall q \in Q, g \in G, n \in N_0$

newPriorityQueue<G>(n): $n > 0$

q.insert(g): q is not full

q.isEmpty: none

q.isFull: none

q.maxItem: q is not empty

q.minItem: q is not empty

q.deleteMax: q is not empty

q.deleteAllMax: q is not empty

q.deleteMin: q is not empty
q.frequency(g): none

Semantic: $\forall q \in Q, g \in G, n \in N_0$

newPriorityQueue<G>(n): create a priority queue of items from G with capacity n
q.insert(g): inserts item g according to priority in q
q.isEmpty: returns *true* if q is empty, *false* otherwise
q.isFull: returns *true* if q is full, *false* otherwise
q.maxItem: returns the item g with the highest priority from q
q.minItem: returns the item g with lowest priority from q
q.deleteMax: deletes the item g with highest priority from q
q.deleteAllMax: deletes all items g with the same highest priority from q
q.deleteMin: deletes the item g with the lowest priority from q
q.frequency(g): returns the number of times the item g occurs in q, regardless of its priority