Rohan Patel

rsp502

11247205

Assignment 2

Question 1 ():

For each of the following functions, give the tightest upper bound chosen from among the usual simple functions listed in Section 3.5 of the course readings. Answers should be expressed in big-O notation.

(a) $f1(n) = n \log_2 n + n^4 \log 280\, n + 2^n/42$

Fastest executing function: $2^n/42$

Big-Oh Notation: $O(2^n)$

(b) $f3(n) = 0.4n^4 + n^2 \log n^2 + \log_2(2^n)$

Fastest executing function: n^4

Big-Oh Notation: $O(n^4)$

(c) $f2(n) = 4n^{0.7} + 29n \log_2 n + 280$

Fastest executing function: $n \log_2 n$

Big-Oh Notation: O(n log n)

Question 2 ():

Suppose the exact time required for an algorithm A in both the best and worst cases is given by the function.

$TA(n) = ((1/280) * n^2) + 42 \log n + 12n^3 + 280\sqrt{n}$

(a) For each of the following statements, indicate whether the statement is true or false.

1. Algorithm A is O(log n) -> False
2. Algorithm A is $O(n^2)$ -> False
3. Algorithm A is $O(n^3)$ -> True
4. Algorithm A is $O(2^n)$ -> True

(b) What is the time complexity of algorithm A in big-Θ notation.

- As n^3 is the fastest growing term
  TA(n) ∈ Θ(n3).
  Θ (n^3) is the big theta Θ of Algorithm A because big Θ implies that the given function is not only an upper bound.

Question 3 ():

If possible, simplify the following expressions. Hint: See slide 11 of topic 4 of the lecture slides!

(a) $O(n^2) + O(\log n) + O(n \log n)$
$= O(n^2)$

(b) $O(2^n) \cdot O(n^2)$
$= O(2^n \cdot n^2)$

(c) $42\ O(n \log n) + 18\ O(n^3)$
$= O(n^3)$

(d) $O(n^2 \log_2 n^2) + O(m)$
$= O(n^2 \log_2 n^2 + m)$

Question 4 ():

(a) Use the statement counting approach to determine the exact number of statements that are executed when we run this code fragment as a function of n. Show all your calculations.

1) The inner loop executes n times for values of j from 1 to n.
2) Each time the loop executes, 2 statements will be executed. So, a total of 2n statements.
3) We also need to consider if the for-loop condition executes to be false. So now, its 2n+1.
4) The outer loop executes n times for values of i between 1 and n.
5) So, for each value of i, as the inner loop has 2n+1 statements. So, 2n+1 executes for the value of n(the outer loop).
6) So, there is one more statement of the outer loop condition.
7) So now (2n+2) statements per outer loop.
8) Now as there is also a possibility of the outer loop being false So it would be +1.
9) Now there are n iterations as seen in 4^{th} line.
10) So, right now the equation is n0(2n+2)+1
11) The number of statements executed are $2n^2+2n+1$

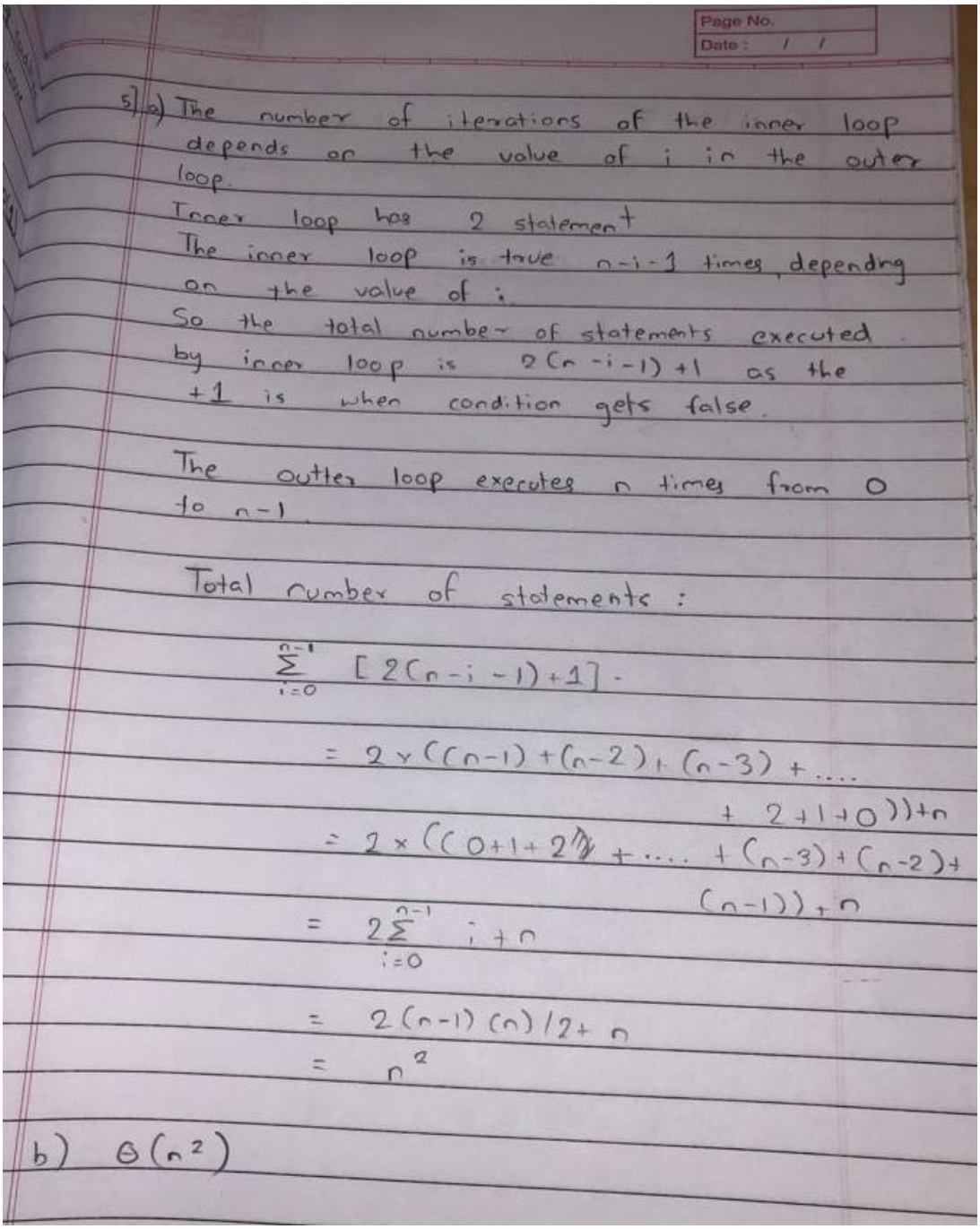
(b) Express the answer you obtained in part (a) in big-Θ notation.

- Θ(n^2) because the given function is not only an upper bound, but also a lower bound

Question 5 ():

(a) Use the statement counting approach to determine the exact number of statements that are executed by this pseudocode as a function of n. Show all your calculations.

(b) Express the answer you obtained in part a) in big-Θ notation



Page No.
Date : / /

5] a) The number of iterations of the inner loop depends on the value of i in the outer loop.
Inner loop has 2 statement
The inner loop is true n-i-1 times, depending on the value of i.
So the total number of statements executed by inner loop is 2(n-i-1)+1 as the +1 is when condition gets false.

The outter loop executes n times from 0 to n-1.

Total number of statements :

$$\sum_{i=0}^{n-1} [2(n-i-1)+1]$$

$$= 2 \times ((n-1)+(n-2)+(n-3)+\dots + 2+1+0))+n$$

$$= 2 \times ((0+1+2+\dots+(n-3)+(n-2)+(n-1))+n$$

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

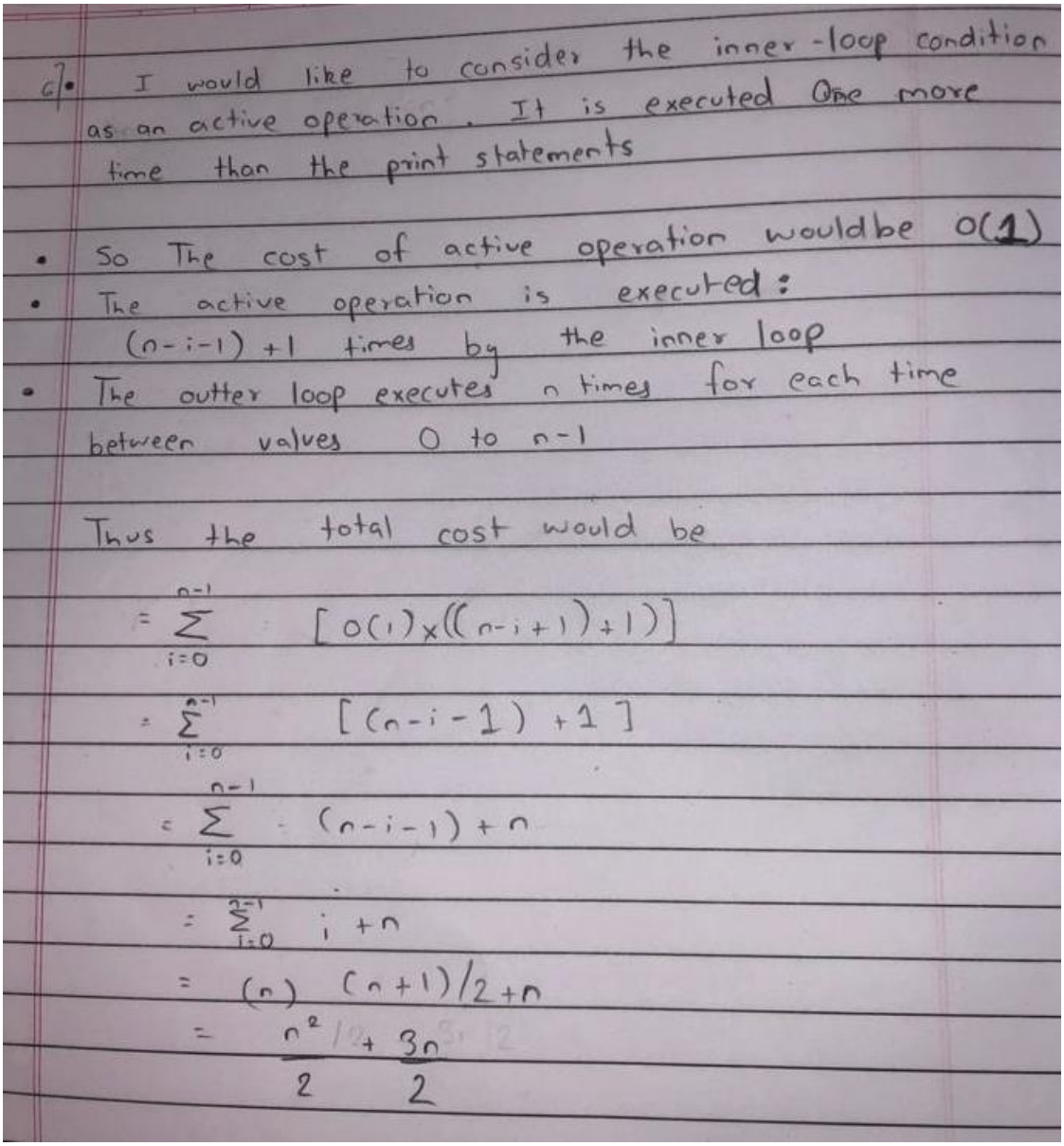
$$= 2(n-1)(n)/2 + n$$

$$= n^2$$

b) $\Theta(n^2)$

Question 6():

Using the active operation approach, determine the time complexity of the pseudocode in question 5. Show all your work and express your final answer in Big-Θ notation.



6) I would like to consider the inner-loop condition as an active operation. It is executed One more time than the print statements

- So The cost of active operation wouldbe O(1)
- The active operation is executed:
  $(n-i-1)+1$ times by the inner loop
- The outter loop executes n times for each time between valves 0 to n-1

Thus the total cost would be

$$= \sum_{i=0}^{n-1} [O(1) \times ((n-i+1)+1)]$$

$$= \sum_{i=0}^{n-1} [(n-i-1)+1]$$

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

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

$$= (n)(n+1)/2 + n$$

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

Question 7():

Using the active operation approach to timing analysis determine the time complexity of this pseudocode in the worst case. Assume that the list of arrays contains n arrays and that each array has exactly m items in it. Show all your work and express your final answer in Big-O notation.

- The loop body would be the first item even though it executes one less time than the loop condition.
  The cost of the active operation is O(log m).
  In searching algorithms, the worst case is always that if the element is not found in the arrays.
  This means that the active operation executes n times.
  The total cost of the active operation is : n * log m which is O(n . log m)

Question 8():

A priority queue is a queue where a numeric priority is associated with each element. Access to elements that have been inserted into the queue is limited to inspection and removal of the elements with smallest and largest priority only. A priority queue may have multiple items that are of equal priority.

Give the ADT specification for a bounded priority queue using the specification method described in Topic 7 of the lecture notes. By "bounded", it is meant that the priority queue has a maximum capacity specified when it is created, and it can never contain more than that number of items.

Solution:

Name: PriorityQueue<G>
Sets: Q : set of priority queues containing elements from G.
G : set of items that can be in a priority queue.
B : {true, false} N : set of positive integers.
N0 : set of non-negative integers.

Signatures:

newPriorityQueue<G>: N → Q
Q.insert(g): G 6→ Q
Q.isFull: → B
Q.isEmpty: → B
Q.count: → N0
Q.maxItem: 6→ G
Q.minItem: 6→ G
Q.deleteMax: 6→ Q
Q.deleteMin: 6→ Q
Q.frequency(g): G → N0
Q.deleteAllMax: 6→ Q

Preconditions:

For all q∈ Q, g∈ G,

q.insert(g): queue is not full
q.maxItem: queue is not empty
q.minItem: queue is not empty
q.deleteMax: queue is not empty
q.deleteMin: queue is not empty
q.deleteAllMax: q must not be empty.
(Operations without preconditions are omitted)

Semantics:

For all n∈ N, g∈ G, n∈ N

newPriorityQueue<G>(n): create a new queue with capacity n.
q.insert(g): insert item g into t in priority order with the highest number being the highest priority.
q.isFull: return true if t is full, false otherwise
q.isEmpty: return true if t is empty, false otherwise
q.maxItem: return the largest (highest priority) item in q.
q.minItem: return the smallest (lowest priority) item in q.
q.deleteMax: remove the largest (highest priority) item in q from q.
q.deleteMin: remove the smallest (lowest priority) item in q from q.
q.frequency(g): return number of times element g appears in the queue regardless of priority.
q.deleteAllMax: all occurrences of the highest priority item are deleted from q.