Assignment 02: Battling Big-Oh

Due: Wed Jan 31, 2024 11:59pmDue: Wed Jan 31, 2024 11:59pm

Kaylee Lewis

1. (15 points) You are given the following algorithm:

Algorithm 1: FunkyIndexes

Input: Some value $n > 0$
Output: A summation based on the value of n
r = 0;
for $i = 1$ *to* n **do**
  for $j = 1$ *to* i **do**
    for $k = j$ *to* $i+j$ **do**
      r = r + 1;
return r

What value is returned by this algorithm, expressed as function of n?

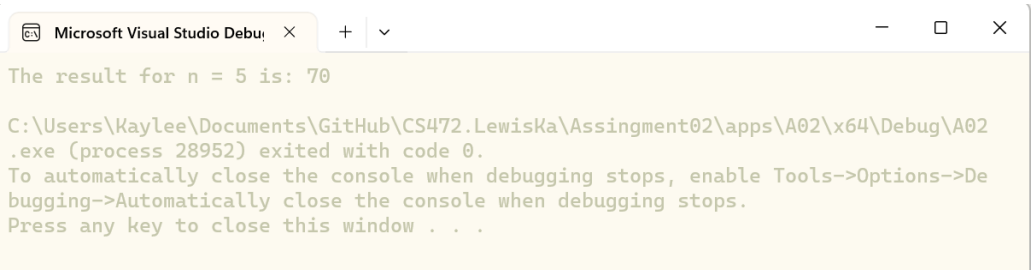
The value returned by this algorithm is the number of times the inner loop runs. The inner loop runs i + j times, so the value returned by this algorithm is the sum of the first i natural numbers. (n*(n+1)*(n+2))/2 is the value returned by this algorithm, expressed as a function of n.

What is the 0(n) of this algorithm? Justify your answer.

The outer loop runs n times, O(n), the middle loop runs i times, O(n^2), and the inner loop runs i + j times, O(n^3). So the O(n) of this algorithm is O(n^3).

```
20
21  #include <iostream>
22
23  // FunkyIndexes algorithm
24  //  This algorithm is O(n^3)
25  int algorithm ( int n )
26  {
27      // r is the result, O(1)
28      int r = 0;
29
30      // The outer loop runs n times, O(n)
31      for ( int i = 1; i <= n; ++i )
32      {
33          // The middle loop runs i times O(n^2)
34          for ( int j = 1; j <= i; ++j )
35          {
36              // The inner loop runs i + j times, O(n^3)
37              for ( int k = j; k <= i + j; ++k )
38              {
39                  // r is incremented by 1
40                  r += 1;
41
42              }
43          }
44      }
45
46      //  The result is returned
47      return r;
48  }
```

```
50  int main ( )
51  {
52      // Example with n = 5
53      int n = 5;
54      int const result = algorithm ( n );
55
56      // Print the result
57      std::cout << "The result for n = " << n << " is: " << result << std::endl;
58
59      return 0;
60  }
```



2. (15 points) Show directly that f (n) = n^2 + 3n^3 ∈ Θ(n^3). That is, use the definitions of O and Ω to show that f (n) is in both O(n3) and Ω(n3).

f(n) = n^2 + 3n^3 ∈ O(n^3) use constants c and n0

$0 \leq f(n) \leq c \cdot g(n), n \geq n0$, g(n)=n^3

$0 \leq n^2 + 3n^3 \leq c \cdot n^3$, divide by n^3

$0 \leq (n^2 / n^3) + 3 \leq c$

$0 \leq (1/n) + 3 \leq c$

$c \geq 3$ and n0=1 , $n \geq 1$

f(n) is in O(n^3)

and f(n) = n^2 + 3n^3 ∈ Ω(n^3) use constants c′ and n0′

$0 \leq c' \cdot g(n) \leq f(n)$, $n \geq n0'$.

$0 \leq c' \cdot n^3 \leq n^2 + 3n^3$, divide by n^3

$0 \leq c' \leq (1/n) + 3$

$c' \leq 3$ and n0′=1 , $n \geq 1$.

f(n) is in Ω(n3)

f (n) is in both O(n3) and Ω(n3) so it is in Θ(n^3).

3. (15 points) The lectures contain a proof that 2(n+1) → Θ(2^n). Can you generalize this thought?

To generalize the proof established two inequalities f(n) ∈ O(g(n)) and f(n) ∈ Ω(g(n)). Find constants c and n0 and c' and n0′ respectively. If both hold, then f(n) is Θ(g(n)) so 2(n+1) → Θ(2^n)

Is an+1 → Θ(an) if a is constant? Justify your answer.

an + 1 ∈ O(an)

$0 \leq an + 1 \leq c \cdot an$, $n \geq n0$

$0 \leq an + 1 \leq c \cdot an$, subtract an

$0 \leq 1 \leq (c - 1) \cdot an$

$c \geq 2$, $(c - 1 \geq 1)$, $n0 = 1$, $n \geq 1$

an + 1 is in O(an)

and an + 1 ∈ Ω(an)

$0 \leq c' \cdot an \leq an + 1$, $n \geq n0'$

$0 \leq c' \cdot an \leq an + 1$, divide by an

$0 \leq c' \leq 1 + (1 / an)$

$c' \leq 2$ and $n\ 0' = 1$, $n \geq 1$

an + 1 is in Ω(an)

an + 1 is in both O(an) and Ω(an) so it is in Θ(an)

4. (20 points) Consider the following algorithm that checks whether a graph represented by its adjacency matrix is complete:

Algorithm 2: GraphComplete

```
Input: An zero-indexed adjacency matrix A of a undirected graph G
Output: A Boolean value of true if graph is complete, false otherwise
if n = 1 then
    return true;
else
    if NOT (GraphComplete (A[n-2],A[n-2])) then
        return false;
    else
        for j in [0..n-2] do
            if A[n-1,j] = 0 then
                return false;
```

What is this order of this algorithm in the worst case? Justify your answer.

This algorithm will continuously check subgraphs recursively until it reaches the base case where n = 1. At each level of the recursion, it will perform an iteration over the last row of the adjacency matrix. In the worst case, the algorithm will have to check every pair of vertices to see if there is an edge between them (n(n-1)/2). This means the complexity is O(n^2).

5. (35 points) The authors of modern software libraries are nice enough to include a sort() function for use in applications. These libraries claim that the sort functions use an algorithm that is at least O(n ∗ lg(n)) in nature. Let's test this claim by writing a small program.

In your language of choice, use the sort routine provided by the language to sort randomly generated

data sets of 5, 10, 50, 100, 500, 1000, 5000, and 10,000 integers. Since we are looking at average data set sizes, you will need to sort at least 10 or more random data sets for each data set size.

You should implement an algorithm that looks like this:

Algorithm 3: Test the order of the library sort

```
Output: A data set of run times for each data set size
for size ∈ [5, 10, 50, 100, 500, 1000, 5000, 100000] do
    Set totalRunTime ← 0;
    for runNumber ∈ [0 .. NumberOfRuns) do
        GenerateRandomDataSet (dataSet,size);
        Note start time;
        Sort (dataSet);
        Note end time;
        Set elapsed time to difference between end and start times;
        Add elapsed time to totalRunTime;
    Set averageRunTime = totalRunTime / NumberOfRuns;
```

The function GenerateRandomDataSet(dataSet, size) needs to generate a random set of integers of whatever size is passed into the function as an input.

Create a graph (using Excel or your favorite graphing tool) that graphs data set size versus the average run time for each size. Include a plot of data set size versus f (n) = n ∗ lg(n) as well. Submit your source code, graph, and a short (1-3 paragraph) analysis of the results (each of which contributes 10 points to the grade for this problem).

Hint: you will need to research two items for this question: (1) the functions in your language of choice to deal with function timing, and (2) how to call the sort() function in your language of choice. If you are programming in C++, review the information found at https://en.cppreference.com/w/cpp/numeric/random for more information on random number generation and
https://en.cppreference.com/w/cpp/algorithm/sort for sorting in C++.

The program tests the library sort function. The program generated random datasets of varying sizes, sorts each dataset multiple times, and records the average runtime for each dataset. It also calculated the theoretical time complexity of f(n) = n * log(n) for each dataset. The resulting print out and graphs show that the claim that the sort algorithm is at least O(n * lg(n)) and that the growth pattern is logarithmic is accurate. The program and graphs show that the sort algorithm is aligning with the expectations the libraries claim.

Microsoft Visual Studio Debu

Data set size	Average run time (ms)	f(n) = n * lg(n)
5	0.1	11.6096
10	0.1	33.2193
50	3.1	282.193
100	6.3	664.386
500	44.6	4482.89
1000	101.8	9965.78
5000	661.5	61438.6
10000	1432.6	132877