Ankit Murarka amurark@ncsu.edu 200111393

4

1. (6 points) Goal: Practice analysis of algorithms. Consider the algorithm represented by the following program fragment.

```
1     MYSTERY(int x, int n)
2     {
3          int sum = 0;
4          if x>10 then {
5               for (int i=1; i<=n^2; i++) {
6                    sum=sum*i;
7               }
8          } else {
9               for(int i=1; i<=n; i++) {
10                   sum = sum+i;
11              }
12         }
13         return x+sum;
14    }
```

a) (2 points) As a function of *x* and *n*, what value does MYSTERY(*x*,*n*) return?

=> MYSTERY(x,n) = $\begin{cases} x & x > 10 \\ \frac{n(n+1)}{2} + x & otherwise \end{cases}$

According to the 'if' condition:

When the value of x is greater than 10 then*sum = sum*i.* Since sum is initialized to 0 in line 3, anything multiplied by 0, remains 0. So sum will remain 0 when x > 10. The function returns: x+sum, which is x+0 = x.

According to the 'else' condition:

When the value of x is less than or equal to 10, then *sum = sum+i*. Since sum is initialized to 0 in line 3, we basically get the sum of first n numbers. Applying sum of an arithmetic progression:

$$S_n = \frac{n}{2}(2a + (n-1)d)$$

We get the sum as:

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

And since the function is returning n + x:

We get the function as:

$$\frac{n(n+1)}{2} + x$$

b) (2 points) Use only ARITHMETIC operations as basic operations, and assume that the size of the input is measured by *n*. Compute the worst-case running time of MYSTERY.

=> Worst case running time for the function comes out to be:

$$n^2 + 1$$

Explanation: For the worst case running time of the function MYSTERY, we will assume that the condition in the control flow, which takes more time, will be executed. We can see that in the ‘if’ condition, the for loop iterates from 1 – n^2 while in the ‘else’ condition the for loop iterates from 1 – n. So, taking the worst-case scenario, we will consider the if condition only.

Consider each arithmetic operation takes time ‘c’.
Since the for loop inside the ‘if’ condition is iterating n^2 times, the arithmetic operation in line 6 takes c1 = n^2c time.
Since the else condition is not considered to be working, we are left with final statement in line 13 which involves another arithmetic operation which takes just 1 cycle, therefore c2 = c.

So, the total running time taken is:

$$c1 + c2$$
or
$$n^2c + c \Rightarrow c(n^2+1)$$

So, the worst case running time of the above function comes out to be n^2+1

c) (2 points) Assume that *x* remains constant while *n* goes to infinity. Derive a tight, big-Oh expression (dependent on the value of *x*) for the running time of MYSTERY. Justify your solution.

=>

Let number of computational steps = f(N)

Number of computational steps = f(input data's length) in worst case scenario

Lets assume that every step that doesn't depend on the input data takes a constant time 'C'. Since x is also a constant, lets assume that even the steps involving 'only X' also takes time 'C'.

So, statement #1,2,3,5,6 all takes time C.

While statement 4,7 and 8 are the ones which are dependent on input.

Now since, we need to calculate the Big-Oh expression, which is upper bound, we will consider only the worst running time. So, taking the if condition and ignoring the else condition we are left with just statement 7 and 8

⇨ $f(N) = C + time\ taken\ by\ \#4 + time\ taken\ by\ \#8$

$$f(N) = C + n^2 + 1$$

Ignoring the constants:

$$f(N) = n^2 + 1$$

According the the big Oh notation:

$f(N) \in O(g(N))$ if there exists c, n_o> 0, then $0 \le f(N) \le cg(N)$ for all $n \ge n_o$.

Therefore,

$$n^2 + 1 \le O(n^2)$$

for all n >= 1, C_o = 2

8

2. (8 points) *Purpose: Learn about bubblesort, practice running time analysis, learn how loop invariants are used to prove the correctness of an algorithm.* Tip: re-read Section 2.1 in our textbook. Solve problem 2-2 [a-d] on page 40 of the textbook.

(a) Let A' denote the output of the BUBBLESORT(A). To prove that BUBBLESORT is correct, we need to prove that it terminates and that

$$A'[1] \le A'[2] \le \cdots \le A'[n]$$

where n = A.length. In order to show that BUBBLESORT actually sorts, what else do we need to prove?

⇨ To analyze if any algorithm is correct, we need to prove:
1. If it actually produces the output in finite time.
2. If it is actually doing what it was designed to do.
3. The output contains the same set of input.

1 and 2 are already mentioned, as its already stated that we need to prove that the Bubblesort algorithm terminates and the output is in ascending order.
The 3rd thing is that the data set doesn't change after applying the algo. So, the input should be the same as the output.
In this case, elements in A' = elements in A, but in a different order.

(b) State precisely a loop invariant for the **for**loop in lines 2-4, and prove that this loop invariant holds. Your proof should use the structure of the loop invariant proof presented in this chapter

⇨ *At the start of each iteration of the **for** loop of line 2-4, the subarray $A[j \dots]$has the smallest element at the **j**th position that is, the element $A[j]$ is the smallest element in the array $A[j \dots]$.*

- **Initialization:** The loop invariant holds prior to the first iteration of the loop, when j = A.length. The subarray $A[j \dots]$ consists of just one element. So A[j] is the smallest element in the array $A[j \dots]$.

- **Maintenance:** After each iteration, the loop invariant holds true because we replace $A[j]$ with $A[j-1]$ if in case $A[j-1]$ is smaller than $A[j]$. So $A[j]$ basically still remains the smallest element in the subarray $A[j \dots]$.

- **Termination:** Finally, once the loop terminates and j becomes equal to i, since we are trying to get $A[j]$ as the smallest element in the subarray $A[j \dots]$ and at then termination of the loop we replace it with $A[j-1]$. We get the smallest number at the $A[i]$ position.

(c) Using the termination condition of the loop invariant proved in part (b), state a loop invariant for the **for**loop in lines 1-4 that will allow you to prove inequality (2.3). Your proof should use the structure of the loop invariant proof presented in the chapter.

⇨ At the start of each iteration of the **for** loop in line 1-4, the subarray $A[1 \dots i-1]$ consists of elements which are sorted. Also, these elements are all smaller than or equal to elements in subarray $A[i \dots]$.

- **Initialization:** The loop invariant holds prior to the first iteration of the loop as the array is empty.
- **Maintenance:** After each iteration, the loop invariant hold true because the inner loop gives us the smallest value from the array $A[j \dots]$ and puts it at the ith position. $A[i] \leq A[j \dots]$.
- **Termination:** By the end of this loop, when it gets terminated, i has iterated from 1 to n, where n is the length of the array. And the array is sorted.

(d) What is the worst case running time of bubblesort? How does it compare to the running time of insertion sort?

⇨ In this case the outer for loop runs (n-1) times. It's the inner loop which is trickier, because the no. of iterations of the inner loop depends on the count of outer loop.
For example: In the first iteration of the outer loop, where i =1, the inner loop runs from n….i+1 which is n….2. So it runs n-1 times in the first iteration of the outer loop. The no. of iterations of the inner keeps on decreasing as the count of the outer loop iteration increases:

1st outer loop iteration : Inner loop runs n-1 times
2nd outer loop iteration : Inner loop runs n-2 times
.
.
.
n–1th outer loop iteration: Inner loop runs 1 time only.

So we can conclude that the inner loop is running 1+2+3+……(n-1) times.

Using the sum of arithmetic progression, we get:

$$S_n = \frac{n}{2}(2a + (n-1)d)$$

So, we get:

$$S_{n-1} = \frac{n-1}{2}(2a + (n-1-1)d)$$

For a,d = 1, we get:

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

which gives:

$$S_{n-1} = \frac{n(n-1)}{2}$$

So, the comparison and swap operations will work $\frac{n(n-1)}{2}$ times. Assuming that they take a combined time of C. So, the total time taken by them is:

$$T(n) = C.\frac{n(n-1)}{2}$$

$$T(n) = \frac{Cn^2 - cn}{2}$$

To calculate the worst case running time of bubble sort, we use Big-Oh notation, which shows that

$$\frac{n^2-n}{2} \in O(n^2)$$

or

$$\frac{n^2-n}{2} \leq cn^2$$

In comparison with the worst case running time of insertion sort, bubble sorts comes up equal in order.

12

3. Purpose: Practice working with asymptotic notation. Please solve (12 points) 3-2 on page 61, (8 points) 3-4 [a-c] on page 62. For full marks justify your solutions.

=>3.1

a) Relation between $lg^k n$ and $n^\in$

We can see that if we make $n \to \infty$,

Both $lg^k n$ and $n^\in \to \infty$.

Lets apply L'Hopital Rule, which states that $\lim_{n\to\infty} \frac{s\{n]}{t\{n]} = \lim_{n\to\infty} \frac{s'\{n]}{t'\{n]}$.

So, s'(n) = $\frac{d}{dn} lg^k n \Rightarrow \frac{k}{n} lg^{k-1} n$

Similarly, t'(n) = $\frac{d}{dn} n^\in \Rightarrow \in n^{\in-1}$.

Here, we see that again s'(n) and t'(n) tends to infinity if n tends to infinity. So, we can apply L'Hopital again. Lets apply L' Hopital 'k' times.

So s'(n) = $\frac{k(k-1)\ldots.(k-(k-1))n^{k-k}}{n^k}$

And t'(n) = $[\in.(\in-1)\ldots..(\in-(k-1))]n^{\in-k}$

Applying limits now:

So, $\lim_{n\to\infty} \frac{s'(n)}{t'(n)} = \frac{k(k-1)\ldots.(k-(k-1))n^{k-k}}{n^k[\in.(\in-1)\ldots..(\in-(k-1))]n^{\in-k}} = \frac{k(k-1)\ldots.(k-(k-1))n^0}{n^k[\in.(\in-1)\ldots..(\in-(k-1))]n^{\in-k}}$

$= \lim_{n\to\infty} \frac{k(k-1)\ldots.(k-(k-1))}{n^k[\in.(\in-1)\ldots..(\in-(k-1))]n^{\in-k}} = \frac{k(k-1)\ldots.(k-(k-1))}{\infty} = 0$

Therefore:

$$s(n) \in o(t(n))$$

Which means that:

$$s(n) < c(t(n))$$

So, we can also say that:

$$s(n) \le c(t(n))$$

Therefore,

$$s(n) \in O\big(t(n)\big)$$

So, to conclude we say:

$$s(n) \in O, o\big(t(n)\big)$$

b) Relation between n^k and c^n

We can see that if we make $n \rightarrow \infty$,

Bothn^k and $c^n \rightarrow \infty$.

Lets apply L'Hopital Rule:

So, s'(n) = $\frac{d}{dn} n^k = kn^{k-1}$

And t'(n) = $\frac{d}{dn} c^n = c^n \log c$

This doesn't solve anything, lets again apply L'Hopital rule as both s'(n) and t'(n) tends to infinity.

So, s''(n) = $\frac{d}{dn}\ kn^{k-1} = k(k-1)n^{k-2}$

And t''(n) = $\frac{d}{dn} c^n logc = c^{2n} \log c$

Here, we can observe a pattern, if we keep on applying L'Hopital rule and differentiate the numerator 'k' times, we get something like this:

$$s^{'''}(n) = k(k-1)(k-2) \ldots.(k-(k-1))n^{k-k}$$

Which means, that

$$s^{'''}(n) = k(k-1)(k-2) \ldots.(k-(k-1))n^{0}$$

So, the numerator becomes:

$$s^{'''}(n) = k(k-1)(k-2) \ldots.(k-(k-1))$$

We, can observe here that numerator is no longer dependent on 'n'. At the same time the value of denominator is:

$$t'''(n) = c^{kn} \log c$$

Therefore, for k >= 1:

We can say that, the for $n \to \infty$, s'''(n) = $k(k-1)(k-2)\ldots.(k-(k-1))$, while t'''(n) = ∞.

So, we get:

$$\lim_{n\to\infty} \frac{s(n)}{t(n)} = \frac{k(k-1)(k-2)\ldots.(k-(k-1))}{\infty} = 0$$

Therefore:

$$s(n) \in o(t(n))$$

Which means that:

$$s(n) < c(t(n))$$

So, we can also say that:

$$f(n) \leq c(t(n))$$

Therefore,

$$s(n) \in O(t(n))$$

So, to conclude we say:

$$s(n) \in O, o(t(n))$$

c) Since sin n is a periodic function, its value can change from -1 -> 1. So n^(sin n) can be either tend to infinity or 0. Hence there can be no relation between n^(1/2) and n^(sin n).

d) Relation between $2^n\,and\;2^{\frac{n}{2}}$.

As we can see that on applying limits for n tending to infinity, we get infinity for both the parts.

So, we can apply L' Hopital Rule.

So, s'(n) = $\frac{d}{dn}2^n = 2^n \lg 2$

And t'(n) = $\frac{d}{dn}2^{\frac{n}{2}} = \frac{2^{\frac{n}{2}}}{2}\log 2$

So, $\frac{s'(n)}{t'(n)} = \frac{2^n \lg 2}{\frac{2^{\frac{n}{2}}}{2}\log 2} = 2^{n+1-\frac{n}{2}} = 2^{\frac{n}{2}+1}$

Applying limits $n \to \infty$

$$\lim_{n\to\infty} 2^{\frac{n}{2}+1} = \infty$$

Therefore:

$$s(n) \in \omega\big(t(n)\big)$$

Which means that:

$$s(n) > c\big(t(n)\big)$$

So, we can also say that:

$$f(n) \geq c\big(t(n)\big)$$

Therefore,

$$s(n) \in \Omega\big(t(n)\big)$$

So, to conclude we say:

$$s(n) \in \Omega, \omega\big(t(n)\big)$$

e) Relation between $n^{\lg c} and\ c^{\lg n}$.

Using the property of logarithms, $a^{\log_b c} = c^{\log_b a}$

We can come to the conclusion that:

$$n^{\lg c} = c^{\lg n}$$

Which means that:

$$n^{\lg c} = 1. c^{\lg n}$$

or

$$n^{\lg c} \in \Theta(c^{\lg n})$$

Also, we can say, that:

$$n^{\lg c} \leq 1.c^{\lg n}$$

Therefore,

$$n^{\lg c} \in O\left(c^{\lg n}\right)$$

And we can also say that:

$$n^{\lg c} \geq 1.c^{\lg n}$$

Therefore, we can say that,

$$n^{\lg c} \in \Omega\left(c^{\lg n}\right)$$

So, in this case:

$$f(n) \epsilon O, \Omega, \Theta g(n)$$

f) Relation between $\lg|n!|$ *and* $\lg|n^n|$.

First we will prove that its an upper bound.

We can write lg|n!| = lg|n(n-1)(n-2)...2.1| by expanding the factorial.
Applying the logarithm property lg|ab| = lg a + lg b.

⇨ lg |n!| = lg |n| + lg|n-1| + ⋯⋯ + lg|2| + lg|1|

<= lg |n| +lg |n| +lg |n| ⋯⋯ lg |n| +lg |n|
= nlg |n|

Therefore we can say that:

lg |n!| <= n.lg|n|

or

lg |n!| $\epsilon\ On.\lg|n|$

or

lg |n!| ϵ O(lg|n^n|)

Now we will prove that its a lower bound as well.

⇨ lg |n!| = lg |n| + lg|n-1| + ⋯⋯ + lg|2| + lg|1|

Considering only the last n/2 terms.

Which gives:

lg|n!| >= lg |n| +⋯⋯+ lg |n/2|
>= n/2 lg|n/2|
= n/2(lg|n| - lg|2|)
= n/2(lg|n| - 1)
= n/2lg|n| - n/2

We can say that, $\frac{n}{2} \leq \frac{n}{2}$lg|$n$|

>= n/2lg|n| - n/2lg|n|
= n/4lg|n|
= ¼ nlg|n|

So,lg|n!| >= ¼ nlg|n|

So lg|n! | $\in \Omega$(lg|n|^n) for c = ¼.

Since lg|n! | $\in \Omega$(lg|n|^n) and lg|n!| $\in$ O(lg|n|^n), then we can say that

$$\lg|n!| \in \Theta(\lg|n|^n)$$

Therefore:

$$\lg|n!| \in O, \Omega, \Theta(\lg|n|^n)$$

3.2 =>

6 points

a) f(n) = O(g(n)) implies g(n) = O(f(n))

This is **false** because this doesn't hold true for all conditions.

For example, lets say f(n) = ln|n| and g(n) = n.

In this case we can confidently say that:

$$f(n) \in O g(n)$$

But the vice versa is definitely not true:

$$g(n) \neq O f(n)$$

Because, for no values of c > 0, can $n \leq c ln|n|$.

b) f(n) + g(n) = $\Theta(\min(f(n). g(n)))$

This is **false.**
Let us take the same values again, i.e. f(n) = ln|n| and g(n) = n.
What we are trying to prove is:

$$f(n) + g(n) \in \Theta(\min(f(n). g(n)))$$

Which gives us $\ln|n| + n = c. \ln|n|$which cannot be true for any value of c.

c) f(n) = O(g(n)) implies lg(f(n)) = O(lg(g(n)), where lg(g(n)) >= 1 and f(n) >= 1 for all sufficiently large n

This is **true**.
Since $f(n) \in O(g(n))$, this means that $f(n) \leq c. g(n)$ for c > 1. Putting log on both sides wont change the inequality. As we can see that

$$\lg|f(n)| \leq \lg|cg(n)|$$

-2; the proof is wrong

$$\lg|f(n)| \leq \lg|c| + \lg|g(n)|$$

This will always hold true for all positive values of lg|c| and lg|g(n)|.

4. *Purpose: Practice working with asymptotic notation.* Rank the following functions by order of growth.

a) (5 points) Find an arrangement f1, f2, ..., fn of the functions satisfying f1 = Ω(f2), f2= Ω(f3), ..., fn-1 = Ω(fn). (Here, lg := logarithm base 2)

4

⇨ The order is:

$$n^{-2} < 1000 < \ln|n^2| < \log|n!| < sqrt(n) < 3^{\lg|n|} < e^{2\ln|n|} < n^2$$

Comparing Sqrt(n) and ln n^2

$\lim_{n\to\infty} \frac{\sqrt{n}}{\ln(n^2)} =$

$= \lim_{n\to\infty} \frac{n^{\frac{1}{2}}}{2\ln n}$

$= \lim_{n\to\infty} \frac{n^{\frac{-1}{2}}.n}{4}$

$= \lim_{n\to\infty} \frac{n^{\frac{1}{2}}}{4} = \infty;$

⇨ Sqrt(n) = $\Omega(\ln(n^2))$

Comparing 1000 &n^{-2}

$\lim_{n\to\infty} \frac{1000}{n^{-2}}$

$= \lim_{n\to\infty} \frac{1000n^2}{1} = \infty$;

⇨ $1000 = \Omega\ (n^{-2})$

Comparing Ln n^2 and 1000

$\lim_{n\to\infty} \frac{\ln(n^2)}{1000}$, limit for n->∞ ,

$$\lim_{n\to\infty} \frac{\ln(n^2)}{1000} \to \infty$$

⇨ $\ln(n^2) = \Omega(1000)$

Comparing 3$^{\lg n}$ and sqrt(n)

$\lim_{n\to\infty} \frac{3^{lgn}}{\sqrt{n}}$

$= \lim_{n\to\infty} \frac{n^{lg\,3}}{\sqrt{n}}$

$= \lim_{n\to\infty} \frac{lg\,3.n^{lg\,3-1}}{\frac{1}{2}n^{\frac{-1}{2}}}$

$= \lim_{n\to\infty} \frac{2lg3.n^{lg3-1}.n^{\frac{1}{2}}}{1} \to \infty;$

⇨ $3^{\lg n} = \Omega$ (sqrt (n))

Now Comparing n^2 and nlg(n)

$\lim_{n\to\infty} \frac{n^2}{n\lg(n)} =$

$$\lim_{n\to\infty} \frac{n}{\lg(n)}$$

$= \lim_{n\to\infty} \frac{1}{\frac{1}{n}}$

$= \lim_{n\to\infty} \frac{n}{1} \to \infty$; implies $n^2 = \Omega$ (n lg(n))

Comparing nlg(n) and $3^{\lg n}$

$\lim_{n\to\infty} \frac{nlg(n)}{3^{lgn}} = \lim_{n\to\infty} \frac{nlg(n)}{n^{lg3}} = \lim_{n\to\infty} \frac{lg(n)+1}{lg3.n^{lg3-1}} = \lim_{n\to\infty} \frac{1}{x(lg-1)\lg(3).x^{ln3-2}} =$ => $\frac{1}{(lg-1)\lg(3)} \lim_{n\to\infty} \frac{x^{ln3-1}}{1} \to \infty$;

b) Now we need to prove which two functions can be interchanged. 2

$Since\ e^{2\ln|n|} \leq n^2$ are equal, we can say that:

$$e^{2\ln|n|} \in \Theta n^2$$

5. (2 points) Purpose: Practice working with asymptotic notation. Assume that a,b, and t are integers, and a,b,t > 0. Show $e^{an} \in \omega(\{bn\}^t)$. 2

=> To prove the above:

Lets try and prove that:

$$\lim_{n\to\infty}\frac{f(n)}{g(n)} = \infty$$

then, we can say that

$$f(n) \in \omega(g(n))$$

Applying the L' Hopital Rule

$$f'(n) = \frac{d}{dn}e^{an} = ae^{an}$$

$$g'(n) = \frac{d}{dn}b^t n^t = tb^t n^{t-1}$$

If we keep applying L'Hopital rule to g(n) and differentiate it 't' times:

Then:

$$g''''(n) = t.(t-1).(t-2)\ldots\ldots2.1b^t n^{t-t}$$

$$g''''(n) = t.(t-1).(t-2)\ldots\ldots2.1b^t n^0$$

$$g''''(n) = t.(t-1).(t-2)\ldots\ldots2.1b^t$$

So, after applying L'Hopital t times to the denominator, the term 'n' is successfully removed. And the numerator becomes:

$$f''''(n) = a^t e^{an}$$

So, for large values of 'n' [as $n \to \infty$] and since t is a positive integer greater than 0, we can say that we will get:

$$\frac{\infty}{t.(t-1).(t-2)\ldots\ldots2.1b^t} \sim \infty$$

Hence proved that:

$$e^{an} \in \omega(\{bn\}^t)$$

6 (8 points) *Purpose: Practice algorithm design and algorithm analysis.* Give pseudocode for a Θ(lg *n*) algorithm which computes a^n, given *a* and *n*. Justify the asymptotic running time of your algorithm. Do **not** assume that *n* is a power of 2. (Here, lg := logarithm base 2)

7 points

```
⇨ CALC(a,n) {
          If( n == 0) {
                    Var_2 = 1
          }
          else if(n % 2 == 0) {
                    var_1 = CALC(a,n/2)
                    var_2 = var_1*var_1
          }
          else {
                    var_1 = CALC(a,(n-1)/2)
                    var_2 = a*var_1*var_1
          }
          return var_2
  }
```

So, according to the pseudocode:

-1;to describe an algorithm, at least one worked example should be given.

- ⇨ We are dividing into subproblems of size n/2 and then solve it once. Therefore D(n) = 1.
- ⇨ Solve 'a=1' subproblems recursively. Therefore aT(n/2) = T(n/2)
- ⇨ Combine solutions to subproblem is not applicable in this case. C(n) = 0
- ⇨ Solve for n direcly, c_0= 1

Therefore,

$$T(n) \leq \begin{cases} 1\ if\ n = 0 \\ T\left(\frac{n}{2}\right) + 1\ otherwise \end{cases}$$

Applying masters theorem:

Taking the second case, we can clearly see that

$$f(n) \in \ \Theta(n \log_b a)$$

And

$$T(n) \in \Theta(n^{\log_b a} \lg n)$$

$$T(n) = \ n^{\log_2 2} \lg n$$

$$T(n) = \lg n$$

Hence proved that’s it’s running time is a $\Theta(\lg n) function$.