Maharishi International University (MIU)

MIDTERM

Course Title and Code: *CS 435 - Design and Analysis of Algorithms*
Instructor: *Dr. Emdad Khan*
Date: *Friday 05/28/2021*
Duration: *10am – 12:30 pm*

Student Name:

ALVIN ABAHO

Student ID:

612418

Total Mark
43.5
46

1. This is a closed book exam. Do not use any notes or books!
2. Show your work. Partial credit will be given. Grading will be based on correctness, clarity and neatness.
3. We suggest you to read the whole exam before beginning to work on any problem.
4. There are 4 questions worth a total of 46 points, on 10 pages (including this one)
5. You can use all back pages for scratch paper (may use for answers if you have used up the designated space for answer).
6. You can use a basic calculator (No smart phone unless network is disabled).

8

Question 1: 10 points (3 + 4 + 3)

a. Show the running time of the following code using big O notation (*show your work*):

```
for (int j = 4; j < n; j=j+2) {  <- from j=4 to n -> O(n)
val = 0;  <- constant op
        for (int i = 0; i < j; ++i)  <- from i to j but j is up to n -> O(n^2) because it is nested
        {
            val = val + i * j;  <- constant op
            for (int k = 0; k < n; ++k)  <- loop from k to n -> O(n^3) because it is nested
                {
                    val++;  <- constant op
                }
        }
}
```

$\therefore$ We have 3 loops, the first from 4 to n which is $O(n)$, the second from $i=0$ to j (but j is up to n) which is $O(n^2)$ and the third which is $k=0$ to n which is $O(n^3)$

$\therefore$ The running time is $O(n^3)$

b. Determine whether f is O(g), Theta (g) or Omega (g). Mention all that applies. Show your work.

i) Is $2^{n+1} = O(2^n)$? Find c and n_0 to show your result.

let $f(n) = 2^{n+1}$ $g(n) = (2^n)$ for $f \in O(g(n))$, $f(n) \leq c\, g(n)$

$\therefore 2^{n+1} \leq c\, 2^n$

$\cancel{2^n} \cdot 2^1 \leq c \cdot \cancel{2^n}$ for all n, 2^n is $\neq 0$,

then $c \geq 2$, for all $n \geq 0$

$\therefore 2^{n+1}$ is $O(2^n)$ for $c \geq 2$ and for all n_0

ii) $\mathbf{f = n^{lgc}}$, $\mathbf{g = c^{lgn}}$

please turn over

Question 1: (continued)

c. Give a Big O estimate for $f(x) = \underbrace{(x^3 + 4)}_{A} \underbrace{\log(x^2 + 4)}_{B} + \underbrace{4x^3}_{C}$

for A: let $f(x) = x^3 + 4$, $g(x) = x^3$

$$\lim_{x \to \infty} \left| \frac{f(x)}{g(x)} \right| = \frac{x^3+4}{x^3}$$
$$= 1 + 4/x^3$$
$$= 1 + 4/\infty$$
$$= 1$$

$\therefore$ x^3+4 is $O(x^3)$ since limit is finite

for B: let $f(x) = \log(x^2+4)$ $g(x) = \log x$

$$\lim_{x\to\infty} \left| \frac{\log(x^2+4)}{\log x} \right|$$

from L'Hopital's rule and chain rule of differentiation,

$\frac{d}{dx}(\log x) = \frac{1}{x \cdot \ln 2}$

$\frac{d}{dx}(\log(x^2+4)) = 2x \cdot \frac{1}{(x^2+4)\cdot \ln 2}$

by chain Rule

$$= \lim_{x\to\infty} \left| \frac{2x}{C\,(x^2+4)} \div \frac{1}{C\cdot x} \right|$$

$$= \lim_{x\to\infty} \left| \frac{2x \times x \cdot \cancel{C}}{\cancel{C}(x^2+4)} \right|$$

$$= \lim_{x\to\infty} \left| \frac{2x^2}{x^2+4} \right|$$

$$= \lim_{x\to\infty} \left| \frac{2}{1+4/x^2} \right|$$

$$= \frac{2}{1+4/\infty}$$

$= 2$ which is finite

$\therefore$ $\log(x^2+4)$ is $O(\log x)$

Please turn over

for c: $4x^3$

let $f(x) = 4x^3$ and $g(x) = x^3$

$$\lim_{x \to \infty} \left| \frac{4x^3}{x^3} \right|$$

$= 4$ which is finite

$\therefore$ Big O estimate of $(x^3+4)\log(x^2+4) + 4x^3$

$$= O\left(O(x^3) \cdot O(\log x) + O(x^3)\right)$$

$$= \max\left[O(x^3 \log x), O(x^3)\right]$$

$$= O(x^3 \log x)$$

12

Question 2: 12 points (4 + 3 + 5)

For each of the following recurrences, derive an expression for the running time using iterative, substitution or Master Theorem.

a. Consider the following recurrence algorithm **[Use Master Theorem – See LAST Page]**

```
long power(long x, long n)
  if (n==0) return 1;     → 2
  if (n==1) return x;     → 2
  if ((n % 2) == 0)       → 2
    return power(x*x, n/2);
  else                    → 4 + T(n/2)
    return power(x*x, n/2) * x;
```

Assume n is power of 2.

i. (2 points) Write a recurrence equation for T(n)

$T(n) = \{4 + T(n/2),\ T(0) = 1\}$ ~~T(1)=~~ ?

ii. (2 points) Solve recurrence equation using Master's method i.e. give an expression for the runtime T(n).

From the master's formula, $a = 1$, $b = 2$, $k = 0$.

$\therefore$ ~~$a < b^k$~~ ~~$1 < 2$~~ ~~$a^k = 1$~~ $a = b^k$ $\therefore$ Runtime is

$1 = 2^0$

$\Theta(n^0 \log n)$

$= \Theta(\log n)$

b. Use **Iterative method**

$$\begin{cases} T(n) = 3T(n-1) + 1 \\ T(1) = 0 \end{cases}$$

$T(n) = 3T(n-1) + 1$

$T(n) = 3[3T(n-2) + 1] + 1$

$= 3^2 T(n-2) + 3 + 3^0$

$T(n) = 3^2[3T(n-3) + 1] + 3^1 + 3^0$

$= 3^3 T(n-3) + 3^2 + 3^1 + 3^0$

for k,

$T(n) = 3^k T(n-k) + 3^{k-1} + 3^{k-2} + \dots + 3^1 + 3^0$

$T(n) = 3^k T(n-k) + \sum_{i=0}^{k-1} 3^i$

please turn over

$$T(n) = 3^k T(n-k) + \sum_{i=0}^{k-1} 3^i$$

for the base case, $n - k = 1$ and $T(1) = 0$

$$\Rightarrow k = n-1$$

$$\therefore T(n) = 3^k \cdot T(1) + \sum_{i=0}^{n-1-1} 3^i$$

$$T(n) = 3 \times 0 + \sum_{i=0}^{n-2} 3^i$$

from $\sum_{j=0}^{N} r^j = \frac{r^{N+1} - 1}{r-1}$,

$$T(n) = \sum_{i=0}^{n-2} 3^i \equiv \frac{3^{n-2+1} - 1}{3-1}$$

$$= \frac{3^{n-1} - 1}{2}$$

$$T(n) = \frac{\frac{3^n}{3} - 1}{2}$$

$\therefore$ Runtime for $T(n)$ is $\underline{O(3^n)}$

Question 2: (continued)

c. Use Induction to show that

$$D(n) = \begin{cases} 0 & \text{if } n = 1, \\ D(n/2) + \lg n & \text{if } n = 2^k \text{ and } k \geq 1, \end{cases}$$

has the solution $D(n) = (\lg n)(\lg n + 1)/2$.

~~*If* $T(n) = rT(n-1) + a$, $T(0) = b$, *and* $r \neq 1$ *then*~~

$$T(n) = r^n b + a\frac{1-r^n}{1-r}$$

~~For all nonnegative integer a.~~

$\boxed{\log 1 = 0}$

for Base case,

$n = 1$

$\therefore D(1) = \log 1 \frac{(\log 1 + 1)}{2}$

$D(1) = 0 \frac{(0+1)}{2}$

$D(1) = 0$

$\therefore$ Assuming $D(n) = (\log n)(\log n + 1)/2$ for $n = 2^k$,

$D(n) = D(n/2) + \log n$ for $n = 2^k$

we need to prove for a certain $n' = 2^{k+1}$

from the recurrence formula,

$\therefore D(n') = D(n'/2) + \log n'$

$D(n') = D\left(\frac{2^{k+1}}{2}\right) + \log 2^{k+1}$

$= D\left(\frac{2^k \cdot 2}{2}\right) + \log 2^{k+1}$

$= D(2^k) + \log 2^{k+1}$

but from the assumption $D(n) = \log n \frac{(\log n + 1)}{2}$,

$D(2^k) = \log 2^k \frac{(\log 2^k + 1)}{2}$

$\therefore D(n') = \frac{1}{2} \log 2^k (\log 2^k + 1) + \log 2^{k+1}$

please turn over

$$D(n') = \frac{1}{2} \log 2^k (\log 2^k + 1) + \log 2^{k+1}$$

but $\log 2 = 1$

$$= \frac{1}{2} \log 2^k (\log 2^k + \log 2) + \log 2^{k+1}$$

from rules of adding logs

$$= \frac{1}{2} \log 2^k (\log(2^k \cdot 2^1)) + \log 2^{k+1}$$

$$= \frac{1}{2} \log 2^k (\log 2^{k+1}) + \log 2^{k+1}$$

$$= \log 2^{k+1} \left[\frac{1}{2} \log 2^k + 1 \right]$$

$$= \log 2^{k+1} \left[\frac{\log 2^k + 2}{2} \right]$$

$2 \equiv 1 + 1$
$\equiv \log 2 + 1$

$$= \log 2^{k+1} \left[\frac{\log 2^k + \log 2 + 1}{2} \right]$$

$$= \log 2^{k+1} \left(\frac{\log(2^k \cdot 2^1) + 1}{2} \right)$$

$$D(n') = \frac{\log 2^{k+1} (\log 2^{k+1} + 1)}{2}$$

Hence the proof

$\therefore$ $D(n) = \frac{\log n (\log n + 1)}{2}$ is true since $D(n') = \frac{\log n' (\log n' + 1)}{2}$ is true for $n' = 2^{k+1}$

Question 3: 11 points (4 + 4 + 3)

~~10~~ 10.5

a. (a) Assume you are creating an array data structure that has a fixed size of n. You want to backup this array after every so many insertion operations. Unfortunately, the backup operation is quite expensive, it takes n time to do the backup. Insertions without a backup just take 1 time unit. per instruction.

(i) How frequently can you do a backup and still guarantee that the amortized cost of insertion is O(1)?

-> One can perform backups ~~every~~ every ~~at~~ after addition of n elements in the array as ~~log~~ long as there exists an amortized cost profit that is sufficient to pay for the expensive backup when the array is full.

-> if cost of backup is n, the profit must be $\geq n$

(ii) Prove that you can do backups in O(1) amortized time. Use the accounting method for your proof.

Elements	C	$\hat{C}$	Profit
1	1	2	1
2	1	2	2
3	1	2	3
4	1	2	4
⋮	⋮	⋮	⋮
K	1	2	K
Backup			n
Backup	n	0	$n-n = 0$

If a backup occurs at $K = n$

let the amortized cost for each insertion be $\hat{c} = 2$ cyber dollars

if $\hat{c} = 2$, after n additions, the total profit is n, which is sufficient to pay for the backup which costs n time

Hence the amortised cost of backup $\hat{c}_b = 0$

∴ The cost of the next insertion is $O(1)$

(b) Explain why Amortized analysis is better than Average Case analysis using probabilistic method.

Amortized analysis	Probabilistic method
- Simple calculation	- is computationally expensive $\sum x P(x)$
- Only uses worst case of parts of the algorithm	- Requires all the inputs to the algorithm in order to calculate expected run time
- Does not depend on the inputs of the algorithm	- Is not practical to get all the inputs to the algorithm.

Compared to probabilistic method which depends on all possible inputs which are hard/impossible to generate, Amortized analysis is simple and easy to calculate as it is profit based.

Total/n vs $E(X) = x P(X = x)$

Question 3: (continued)

c. Use RadixSort, with two bucket arrays and radix = 11, to sort the following array: [63, 1, 48, 53, 24, 10, 12, 30, 100, 115, 17]. Show all steps of the sorting procedure. Then explain why the running time is O(n).
What would be the running time if you used ONE bucket?

Remainder bucket

for $n=11$

	100 12 24 1	24		48	115	17		30 63	53	10
0	1	2	3	4	5	6	7	8	9	10

← run time $= O(n)$

Quotient bucket

$n=11$

-0.5

10 1	17 12	30 24		53 48	63				100	115
0	1	2	3	4	5	6	7	8	9	10

← run time $= O(n)$

Sorted array = [1, 10, 12, 17, 24, 30, 48, 53, 63, 100, 115]

Why the run time is O(n)?

Radix Sort does not depend on any element comparisons unlike other sorting algorithms that are decision based and run on O(n log n) time at best.
Radix Sort usually depends on an internal counting sort which runs in O(k+n) and if k is O(n), then the sort becomes O(n) time.

✓

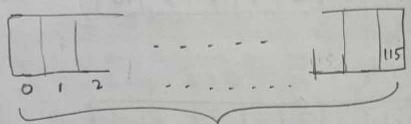
Please turn over.

What is the run time for ONE bucket?

If we use one bucket, it needs to be created based on the range of elements K in the array.

The range of elements in the array is $0 \longrightarrow 115 \approx 11^2$

Therefore there would need to be 121 approximate partitions of the bucket to accomodate the largest number (115)



115 approx. $\approx 121 = 11^2$

$\hookrightarrow n = 11$

$\therefore$ The run time for ONE bucket would be $O(n^2)$

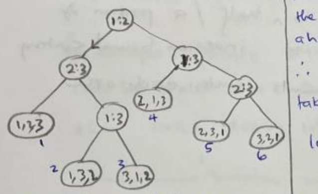
Question 4: 13 points (4 + 5 + 4) 13

a. Use Decision tree and binary tree basic ideas to prove the following theorem:

1. "Every comparison based sorting algorithm has, for each n, running on input of size n, a worst case in which its running time is $\Omega(n\log n)$".

2. How does comparison based sorting achieves $\Omega(n\log n)$ compared to $O(n^2)$ running time of inversion bound sorts like insertion sort and bubble sort? Explain your answer.

Take an example of a decision between 3 items 1, 2, 3



for 3 items → 6 decisions $= 3!$

∴ for n items the decisions are $n!$

but from a binary tree, the total number of leaves for a height h is 2^h

$\therefore n! \leq 2^h$

taking logs on both sides

$\log(n!) \leq (\log 2^h)$

$h \geq \lg n!$

From Stirling's approximation,

$n! > \left(\frac{n}{e}\right)^n$

This means that

$h \geq \lg n! > \lg\left(\frac{n}{e}\right)^n$

$h \geq \log\left(\frac{n}{e}\right)^n$

$h \geq n[\log n - \log e]$

$\therefore h$ is ~~$\Theta(n\log n)$~~ $\Omega(n\log n)$

Continued on next page.

b. (i) Explain with an example what is meant by "Mathematics is not sound".

Historically, it was widely accepted that to prove something, it has to be "true". However, mathematics has shown that it can prove something that is ~~[illegible]~~ true, and something that is false. Hence is not sound.

(ii) Is Mathematics complete? Explain your answer.

Mathematics is Not Complete. For example, mathematics defines certain rules for numbers like exponentials 2^n that are finite but fails to clearly define rules for very large numbers. Hence it lacks complete definitions for behaviours of certain assumptions across all possible inputs.

(Not fully complete)

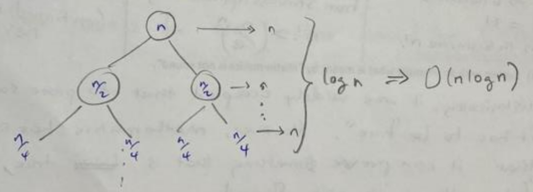
$2^0 = 1$,
$2^1 = 2$
$\therefore 2^\infty = ?\ \infty$, yet $1/0$ is also ∞?

Continuation of Q4. (a)

How does comparison based sorting achieve $\Omega(n \log n)$ compared to $O(n^2)$ of other inversion bound algorithms?

Comparison based sorting uses the divide and conquer strategy where after every recursion, the number of elements in the subsequent recursive call is for example half / a portion of the previous elements compared to inversion bound sorting that needs to compare all elements with each other hence no divide and conquer.

for comparison based,



for inversion based

$T(n) = K + n + T(n-1)$

which is $O(n^2)$

Question 4: (continued)

c. You would like to determine which of your Facebook friends are early adopters. So, you have decided to sort them using Facebook account ids which are 64-bit numbers. Which sorting algorithm will be most appropriate – **Insertion sort, Merge sort, Quicksort, Counting sort or Radix sort? Explain why.**

The most appropriate Sorting algorithm would be a Radix Sort with Buckets. ~~a Bucket Sort~~.

for example, since 64-bit numbers are large, they can be split into 4 buckets of 16-bit numbers and then sorted easily with a counting sort.

$$[\overbrace{0\ 1\ 2 \cdots 16}^{1},\ \overbrace{\cdots 32}^{2}\ \overbrace{\cdots 48}^{3}\ \overbrace{\cdots 64}^{4}]$$

in this way, the large 64 bit numbers can then be sorted in individual buckets from the least Significant (LSB) bucket, to the Most Significant bucket. (MSB)

The result is a sorted array of account ids of my Facebook friends with the earliest adopters appearing first. ✓