Maharishi International University (MIU)

MIDTERM

Course Title and Code: *CS 435 - Design and Analysis of Algorithms*
Instructor: *Dr. Emdad Khan*
Date: *Friday 07/30/2021*
Duration: *9am – 11:30 pm*

Student Name:
SSEMUJU BENEDICT

Student ID:
612756

Total Mark
[redacted]
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 correctne and neatness.
3. We suggest you to read the whole exam before beginning to work on any proble
4. There are 4 questions worth a total of 46 points, on 9 pages (including this one)
5. You can use all back pages for scratch paper (may use for answers if you have u designated space for answer).
6. You can use a basic calculator (No smart phone unless network is disabled).

7.5

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 i = 0; i < n + 100; ++i) — O(n)
{
    for (int j = 0; j < i * n ; ++j) — O(n)
    {
        sum = sum + j;
    }                                          } O(2n).
    for (int k = 0; k < n + n + n; ++k) — O(n)
    {
        c[k] = c[k] + sum;
    }
}
```

-2

Ans.

The outer for loop will run for O(n). The inner for loops will both run for O(n) each. And they are in parallel. So there is an outer loop and two inner loops nested inside. This gives $O(n^2)$
Inner loops are O(2n) nested in O(n). Hence $O(n^2)$.

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

i) $f = 2^{2n}$, $g = 2^n$

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

$\Rightarrow f$ is $\omega(g)$.

g is $o(f)$ ~ small-o

ii) $f = n^{\lg m}$ and $g = m^{\lg n}$; **Show your reasoning / work.**

Taking $\log(f/g) \Rightarrow \log\left(\frac{n^{\log m}}{m^{\log n}}\right) = \log m \log n - \log m \log n = 0$.

.5

Since $\log(f/g) = 0$. This implies $(f/g) = 1$. Since $\log 1 = 0$.

Since its a constant f is $O(g)$.
g is $\Omega(f)$ $\Rightarrow$ f is Theta of g

c. Give a Big O estimate for the following function

$(2^n + n^2)(n^3 + 3^n)$.

2^n is $O(2^n)$ since $2^n \le c2^n$. n^2 is $O(n^2)$ since $n^2 \le cn^2$.
Since its addition, we take max of $O(2^n)$ and $O(n^2)$. Therefore $(2^n + n^2)$ is $O(2^n)$.
n^3 is $O(n^3)$ since $n^3 \le cn^3$. 3^n is $O(3^n)$ since $3^n \le c3^n$. Therefore $(n^3 + 3^n)$ is $O(3^n)$.
$(2^n + n^2)(n^3 + 3^n)$ is $O(2^n \times 3^n) = O(2^n \cdot 3^n) = O(6^n)$

12

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

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;
   if (n==1) return x;
   if ((n % 2) == 0)
      return power(x*x, n/2);
   else
      return power(x*x, n/2) * x;
```

Assume n is power of 2.

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

$T(0) = 2$

$T(1) = 2$

$T(n) = T(n/2) + 6$

ii. (2 points) Solve recurrence equation using Master's method i.e. give an expression for the runtime T(n). $a = 1$; $b = 2$; $c = 6$; $k = 0$.

$\Rightarrow a = 1$; $b^k = 2^0 = 1$.

Since $a = b^k \Rightarrow T(n) = \Theta(n^0 \log n) = \Theta(\log n)$

b. Use **Iterative method**

$\cancel{T(n) = T\left(\frac{n}{2}\right) + T(n-2) + c}$ $for\ n > 0$ [Use Iterative method] $T(n) = 3T(n-1) + 1$

$T(1) = 0$.

T(n) = 1 for n = 0.

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

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

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

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

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

At K^{th} step;

$T(n) = 3^K T(n-K) + 3^{K-1} + \ldots + 3^1 + 3^0$.

To reach base condition $n - K = 1 \Rightarrow n = K + 1$.

$T(n) = 3^K T(1) + 3^{K-1} + \ldots + 3^1 + 3^0$.

but $T(1) = 0$ and $K = n - 1$

PTO.

Question 2: (continued)

c. **Use Induction to show that**

$$\sum_{r=1}^{n} r(r+1) = \frac{1}{3}n(n+1)(n+2)$$

Base case: L.H.S $r(r+1) = (1)(1+1) = 2$.

R.H.S $= \frac{1}{3}(1)(1+1)(1+2) = \frac{1}{3}(1)(2)(3) = 2$ As required

Induction Step:

Assume true for $r=k$ and prove for $r = k+1$

$$\Rightarrow \sum_{r=1}^{k} k(k+1) = \frac{1}{3}k(k+1)(k+2).$$

$$\sum_{r=1}^{k+1} r(r+1) = \sum_{r=1}^{k} r(r+1) + (k+1)(k+1+1)$$

$$= \frac{1}{3}k(k+1)(k+2) + (k+1)(k+2).$$

$$= \frac{1}{3}(k+1)\{k(k+2) + 3(k+2)\}$$

$$= \frac{(k+1)(k^2+2k+3k+6)}{3}$$

$$= \frac{(k+1)(k(k+2)+3(k+2))}{3}$$

$$= \frac{(k+1)(k+2)(k+3)}{3}$$

$$= \frac{(k+1)(k+1+1)(k+1+2)}{3}$$

If $k' = k+1$

$$= \frac{1}{3}k'(k'+1)(k'+2)$$ As required.

11

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

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.

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

I can do back up every after n insertions. This can be guaranteed by charging 2 cyber dollars per insertion. This implies there is a profit of 1 dollar from every insertion. After n operations, I have profit of n cyber dollars to offset the backup cost. For n operations, this $O(n)$.
Per operation, this $\frac{O(n)}{n}$ which is $O(1)$.

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

Actual cost; Amortized Cost

$C_{insert} = 1$ $\hat{C}_{insert} = 2$

$C_{backup} = n$. $\hat{C}_{backup} = 0$.

After n operations, Total Number of insertions is n; $\Rightarrow \sum_{i=1}^{n} c_i = 2n$.

The total Amortized cost $\Rightarrow \sum_{i=1}^{n} \hat{c}_i = 2n$. This implies $\sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \hat{c}_i$. Profit is n

Since total Amortized cost is 2n. This is $O(n)$
Per Operation $O(n)/n = 1$

(b) Use the QuickSelect algorithm to manually compute the 5th smallest element of the array [1, 5, 23, 0, 8, 4, 33]. Assume that the rightmost element is used as the pivot in each case. Show what happens in each self-call, indicating the new input array and the current value of *k*.

[1, 5, 23, 0, 8, 4, 33] $k = 5$

L E G

[1, 5, 23, 0, 8, 4] [33] [ ] $k = 5$

L E G

[1, 0] [4] [5, 23, 8] $k' = k - |L| - |E|$
$= k - 1 - 2 = 2$

L E G

[5] [8] [23] At this point $|L| < k \leq |L| + |E|$ is satisfied

L E G

[ ] [8] [ ]

Therefore fifth element is 8

Question 3: (continued)

c. Use RadixSort, with two bucket arrays and radix = 12, to sort the following array: [63, 1, 48, 53, 24, 10, 12, 30, 100, 141, 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? What would be the running time if you used ONE bucket?

Remainder	12 24 48	1		63	100	17 53	30			141	10	
Bucket	0	1	2	3	4	5	6	7	8	9	10	11

Quotient	10 1	17 12	30 24		53 48	63			100			141
Bucket	0	1	2	3	4	5	6	7	8	9	10	11

Sorted Array [1, 10, 12, 17, 24, 30, 48, 53, 63, 100, 141]

→ Explanation for time $O(n)$:

The time taken to write into Remainder Bucket is $O(n)$.

Time taken to write into quotient Bucket is $O(n)$.

Time taken to read sorted array from quotient bucket is $O(n)$.

This results in running time $O(n)$.

→ If one bucket was used, the range would have to run from 1 to 141 which is $\simeq 12^2 = 144$.

Using one Bucket would increase running time from $O(n)$ to $O(n^2)$.

13

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

a. What is the worst case running time of Quicksort? Can you improve the worst case running time of quicksort? If so, describe to what value and how.

The worst running time of QuickSort is $O(n^2)$. This occurs when pivot is chosen as the unique minimum or maximum value of array.

The worst case running time can be improved to $O(n \log n)$.

This is done using the super quick sort method in which a pivot is checked to identify if its a good pivot or a bad pivot. If its a bad pivot, its discarded and another is picked. The process of assessing pivot run like the split of binary search and results in a tree of maximum length $\log n$ resulting in $O(n \log n)$ since each level running time is $O(n)$

A Good pivot is one that has L and G < $3/4 n$.

b. (i) 2 points - Explain with an example what is meant by "Mathematics is not sound".

Mathematics being sound means only the true statements can be proved. However. Mathematics is not sound since false statements can be Proved. For example P OR $\bar{P}$ Always gives true and this can be proved. However, P AND $\bar{P}$ can also always be proved. and it's Always False. This means Mathematics is not sound.

(ii) **3 points** - Show how Quicksort is not stable by using in-place random partitioning algorithm and the following 4 numbers {4a, 4b, 4c, 4d} (show all steps).

Take 4d as Pivot

$4a \quad 4b \quad 4c \quad \underline{4d}$

$\underset{i}{\uparrow} \qquad \underset{j}{\uparrow}$

Swap 4a & 4c

$4_c \quad 4_b \quad 4_a \quad \underline{4_d}$

$\underset{i}{\uparrow}\underset{j}{\uparrow}$

$4_c \quad 4_b \quad 4_a \quad \underline{4_d}$

$\underset{j}{\uparrow} \quad \underset{i}{\uparrow}$

Swap i with Pivot

$4_c \quad 4_b \quad 4_d \quad 4_a$

Since the original order has changed, Quicksort is Unstable.

Question 4: (continued)

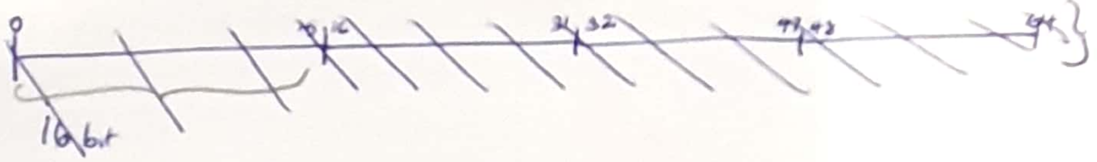
c. You would like to determine which of your Facebook friends are early adopters. So, you have decided to sort them using Facebook account ids which are 64-bit numbers. Which sorting algorithm will be most appropriate – **Insertion sort, Merge sort, Quicksort, Counting sort or Radix sort? Explain why.**

I would use Radix Sort because it gives running time $O(n)$.

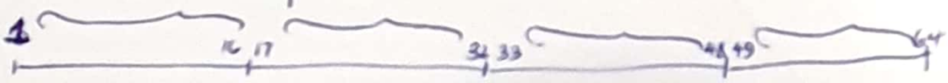
The rest would give $O(n^2)$ and $O(n\log n)$.

Insertion gives $O(n^2)$; Merge Sort $O(n\log n)$, Quicksort $O(n\log n)$ and $O(n^2)$ in worst case. The Number is too big so counting sort not advisable.

I use Radix sort and Divide the number into 4 passes each with 16 bits.



16 bit number passes



I would sort the number from the Least Significant Digit to Most Significant.

This would give a running time of $O(n)$.