Department of Computer Science

UET Lahore, New Campus

Name:

Registration No:

EXAM: QUIZ 1 | CS-272 Design and Analysis of Algorithms | Time Limit: 50 minutes | Total Marks: 20

Semester: SPRING 2025 | Marks Obtained:

NOTE: Attempt all the questions on Question paper. NO RETAKE [CLO1, CLO2, CLO3, CLO4]

Q No. **Solve the following questions and write the answers in the space provided. Show your work. The correct answers without any work will result in zero marks.**

1. [1 point] List the following functions in increasing order of their order of growth

$(n-2)!,\ 5\lg(n+100)^{10},\ 2^{2n},\ 0.001n^4+3n^3+1,\ \ln^2 n,\ \sqrt{n},\ 3^n.$

order of n: $0 < \ln^2 n < 5\lg(n+100)^{10} < 0.001n^4+3n^3+1 < 2^{2n} < 3^n < (n-2)!$

2. [2 points] For each of the following pairs of functions, indicate the class $\Theta(g(n))$ the function belongs to (Use simplest g(n)). Prove your assertions

1. $2^{n+1}+3^{n+1} \Rightarrow O(3^n)$ $\Omega(3^n)$

The dominant term grows faster

$2^{n+1}+3^{n+1} \le 3^{n+1}+3^{n+1} \le 2\cdot 3^{n+1}$

$\le 3^n \cdot 2 \cdot 3 = 6\cdot 3^n$

$c \quad g(n)$

$2^{n+1}+3^{n+1} \ge 3^{n+1}$

$\ge 3^n \cdot 3$

$g(n) \quad c$

$n>0$

$2^{n+1} \le 3^{n+1}$

$2^{n+1}+3^{n+1} \le 3^{n+1}+3^{n+1} = O(3^n)$

$\Theta(3^n)$.

2. $\lfloor \log_2 n \rfloor$ its $O(\lg_2 n)$ and $\Omega(\lg_2 n)$ so $\Theta(\lfloor \lg_2 n \rfloor)$

3. [2 points] Prove that $n^4+20n^3+1000 = \Theta(n^4)$. Find c and n_0. Yes

$0 \le c_1 n^4 \le n^4+20n^3+1000 \le c_2 n^4 \quad n>n_0$

$c_1 = 1021 \quad c_2 = 1$

$n \ge 1$

Big O

$n^4+20n^3+1000 \le n^4+20n^4+1000n^4$

$= 1021n^4$

$c \quad g(n)$

$n \ge 1$

$20n^3 \le 20n^4$

$1 \le n$

$1000 \le 1000n^4$

$1 \le n$

Big Ω

$n^4+20n^3+1000 \ge n^4$

$n^4 \ge n^4 \quad c=1$

$n>0$

4. [4 points] Analyze the complexity of the code in terms of asymptotic notations

```
function p(n)
 if n is even
   count ← n * log2(n)
 else
  count ← n²
 for i ← 1 to count do
  print "*"
```

depends upon count.

for even (n) — count = $n\lg n$

for odd (n) — count = n^2

What will be the best- and worst-case input of the algorithm.

(Best case will be very small value of n, $n=1$, $n=2$) but This algorithm will run max for any value of n.

Is it true to say that the run time of the function has different upper and lower bounds specific to either the best case or worst case?

The worst case of this algorithm has upper bound of n^2 and lower bound of $n\lg n$.

Since for any 'n' value either even or odd loop will be executed max. count value.

Yes not specified in code as separate case loop will be executed count+1 times hence best can be said.

5. [1 point] Consider the following algorithm

ALGORITHM *Mystery(n)*

```
//Input: A nonnegative integer n
S ← 0
for i ← 1 to n do
    S ← S + i * i
return S
```

a. What does this algorithm compute? Sum of Squares

b. What is its basic operation? Multiplication.

6. [4 points] Set up and solve a recurrence relation for the number of times the algorithm's basic operation is executed

ALGORITHM *S(n)*

```
//Input: A positive integer n
//Output: The sum of the first n cubes
if n = 1 return 1
else return S(n − 1) + n * n * n
```

$$M(n) = \begin{cases} 0 & n = 1 \\ M(n-1) + 2 & \end{cases}$$

$$M(n) = M(n-1) + 2$$
$$= M(n-2) + 2 + 2$$
$$= M(n-3) + 2 + 2 + 2$$
$$= M(n-4) + 2 + 2 + 2 + 2$$

$$M(n-i) = M(n-i) + 2i$$
$$= M(n-n+1) + 2(n-1)$$
$$= M(1) + 2(n-1)$$
$$= 2(n-1) = O(n)$$

7. [6 points] From algorithmic design techniques one technique is Brute Force Technique. It is a straightforward approach to solving a problem, usually directly based on the problem statement and definitions of the concepts involved. The term force in this strategy refers to the computational power of a computer rather than human intellect. Another way to describe the brute-force approach is "Just do it!" since it relies on exhaustive computation rather than clever problem-solving. In many cases, brute force is the simplest and most straightforward strategy to implement.

You are familiar with two sorting algorithms that process all elements exhaustively without taking advantage of existing order, i.e. the Brute Force Approach. Can you name those two sorting algorithms?

Selection Sort and Bubble Sort.

State the loop invariants for both algorithms.

Selection Sort

Initialization: before iteration: Sorted section is empty (trivially sorted)

Maintainence: Smallest element in unsorted portion as found and swapped with first $(i+1)$ element in sorted section. Hence elements are sorted.

Termination: $(n-1)$ elements are sorted after $n-1$ iterations. last element is already in place.

Bubble Sort

Swap adjacent elements if they are in wrong order. The largest element are in their correct position, at the end of array

I: before iteration no elements are in their final position.

M: largest element is bubbled up to correct position at the end of array ensuring $i+1$ elements are sorted

T: $(n-1)$ iterations, largest $(n-1)$ elements are in their correct position one (smallest) is already in place.

Department of Computer Science

UET Lahore, New Campus

Name:

Registration No:

EXAM: QUIZ I	CSC-208 Design and Analysis of Algorithms	Time Limit:	Total Marks: 20 [5*4]
Semester: SPRING 2025		40 minutes	Marks Obtained:

NOTE: Attempt all the questions on Question paper. **[CLO1, CLO2, CLO3, CLO4]**

Q No. **Solve the following questions and write the answers in the space provided. Show your work. The correct answers without any work will result in zero marks.**

1. Suppose that for inputs of size n on a particular computer, insertion sort runs in $8n^2$ steps and merge sort runs in $64n\lg n$ steps. For which values of n does insertion sort beat merge sort?

2. The while loop of lines 5-7 of the INSERTION-SORT procedure uses a linear search to scan (backward) through the sorted subarray A [1: j-1]. What if insertion sort used a binary search instead of a linear search? Would that improve the overall worst-case running time of insertion sort to Θ (n lg n)?

```
INSERTION-SORT(A, n)
1  for i = 2 to n
2      key = A[i]
3      // Insert A[i] into the sorted subarray A[1 .. i − 1]
4      j = i − 1
5      while j > 0 and A[j] > key
6          A[j + 1] = A[j]
7          j = j − 1
8      A[j + 1] = key
```

3. Consider the following algorithm.

```
ALGORITHM Mystery(n)
    //Input: A nonnegative integer n
    S ← 0
    for i ← 1 to n do
        S ← S + i * i
    return S
```

a. What does this algorithm compute?
b. What is its basic operation?
c. How many times is the basic operation executed?
d. What is the efficiency of this algorithm?
e. Suggest an improvement, or a better algorithm altogether, and indicate its efficiency class. If you cannot do it, try to prove that, in fact, it cannot be done.

4. Consider the following recursive algorithm for computing the sum of the first n cubes:
$S(n) = 1^3 + 2^3 + \ldots + n^3$.

```
ALGORITHM S(n)
    //Input: A positive integer n
    //Output: The sum of the first n cubes
    if n = 1 return 1
    else return S(n − 1) + n * n * n
```

a. Set up and solve a recurrence relation for the number of times the algorithm's basic operation is executed.
b. How does this algorithm compare with the straightforward non recursive algorithm for computing this sum?

Q no 2 : Insertion Sort takes the key and insed the key into the sorted section. Sorted section is always shifted to place the key in its right position. The binary search procedure can find the the key place (position) in sorted section in less number of comparisons but still n elements are to be shifted in $\Theta(n)$ So the run time of worst case will remain $\Theta(n^2)$.

Since we know that while loop takes max time in worst case by iterating most (or shifting all the elements to create a space - In linear search its $O(j)$ $j \Rightarrow n-1$ to 0

So if binary search is used, to find correct position $O(\lg j)$ time is taken

$\cong O(\lg n)$ and for n elements it will be $O(n \lg n)$

But shifting takes $O(j) \cong O(n)$ time in worst case and for n elements it will be $O(n^2)$

So total time for ~~while loop~~ worst case is $\Rightarrow O(n \lg n) + O(n^2)$

finding the key position ↑ shifting ↑

hence $O(n^2)$.

for loop ($i=0 \longrightarrow n$)

while loop : j ($n-1 \longleftarrow 0$)

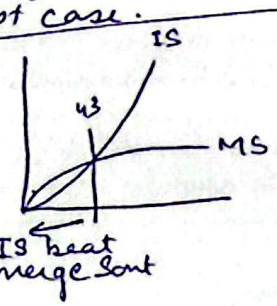
→ find index for key. (shift operation dominates)

→ Shift operation.

* Basic operation is key comparison so runtime will not be affected in worst case.

Binary search create no improvement.

Q no. 1



$8n^2 \leq 64 n \lg n$ | So $n \geq 2$

$n \leq 8 \lg n$ as $\lg n \neq 0$ for $n = 1$.

$n \leq 8 \lg 2$ $1 \neq 0$

$2 \leq 8 \rightarrow$ True

Ans $n = 2 \rightarrow 43$

Trying for powers of 2^n

2^2 $n \leq 8 \lg n$

$4 \leq 8 \lg 4$

$4 \leq 16$ True

2^3 $8 \leq 8 \lg 8$

$8 \leq 24$ True

2^4 $16 \leq 8 \lg 2^4$

$16 \leq 32$ True

2^5 $32 \leq 8 \lg 2^5$

$32 \leq 40$ True

2^6 $64 \leq 8 \lg 2^6$

$64 \not\leq 48$ false.

So n is b/w 32-64

trying (40) $40 \leq 8 \log 40$

$40 \leq 42.56$ app

(44) $44 \leq 8 \log 44$ false $44 \leq 43.67$

(42) $42 \leq 8 \log 42$

$42 \leq 43.13$ app

(43) $43 \leq 8 \log 43$

$43 \leq 43.4$

So $n = 43$. before 43 IS always beat MS after 43 MS beat IS.

Qno.3 a) Compute sum of squares $S(n) = \sum_{i=1}^{n} i^2$

b) Basic operation is Multiplication Addition operation also take same amount of time.

c) $C(n) = \sum_{i=1}^{n} 1 = n$.

d) $O(n)$ linear.

e) Func (n)
return $n\frac{(n+1)(2n+1)}{6}$ ⟹ Constant time.

Qno. 4

$$M(n) = \begin{cases} 0 & n = 1 \\ M(n-1)+2 & n > 1 \end{cases}$$

Using backward Substitution.

$M(n) = M(n-1) + 2$
$= M(n-2) + 2 + 2$
$= M(n-3) + 2 + 2 + 2$
$= M(n-4) + 2 + 2 + 2 + 2$
$= M(n-i) + 2i$

$n - i = 1$
$i = n - 1$
$= M(n - n + 1) + 2(n-1)$
$= M(1) + 2(n-1)$
$= 2(n-1) = O(n)$

b) non-recursive. Sum(n)
S ← 0
for i → 1 to n do
S ← S + i × i × i
return S → $O(n)$

Optimal Soln. Sum (n)
return $\left(\frac{n(n+1)}{2}\right)^2 \rightarrow O(1)$