University of Petra		جامعة البترا - ثلاثون عاما University of Petra
Faculty of Information Technology		كلية تكنولوجيا المعلومات
Department of Computer Science		قسم علم الحاسوب

Advanced Algorithms
601326
Midterm Exam – 2024 2

Your Name: …….. **Your ID: ……**

Your Instructor Name: ……

Instructions for the Exam:

- Write your name and ID number on the exam and answer sheets.
- Write the number of the section that you enrolled in.
- Write the name of your instructor.
- Questions in the exam not allowed.
- Using any type of technology (mobiles, smart watches) not allowed
- Using extra papers or sheets not allowed
- The exam consists of Six questions.

For instructor use only:

Question number	Course ILO	Program ILO	Question weight	Student mark
Q1			4	
Q2	K2		4	
Q3	I1		7	
Q4			5	
Q5			6	
Q6			4	
Total /30				

Q1) Prove by induction that: **(4 marks)**

$$\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$$

Q1) $\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$

① Base Step:

$\sum_{i=0}^{0} 2^0 = 1$ & $2^{0+1} = 2 - 1 = 1$ ✓

② Induction Step: suppose $n = k+1$

$\sum_{i=0}^{k+1} 2^i = 2^{k+2} - 1$ ✓

$= \underbrace{2^1 + 2^2 + \dots + 2^k}_{\downarrow} + 2^{k+1}$

$= 2^{k+1} - 1 + 2^{k+1}$

$= 2^1(2^{k+1}) - 1$

$= 2^{k+2} - 1$ ✓

Q2) Consider the following algorithm. **(4 marks)**

```
Algorithm XYZ (A [0..n − 1])
//Input: An array A[0..n − 1] of n real numbers
val ← 5
res1 ← 0
res2 ← 1
for i ← 0 to n − 1 do
    if A[i] <= val
    res1 ← A[i]+res1
    if A[i] > val
    res2 ← A[i]*res2
return res1 – res2
```

a. What does this algorithm compute?
b. What is the time complexity for the algorithm?

Solution:

a. The algorithm returns (sum of elements <= 5) - (product of elements > 5).
b. Total time complexity is O(n).

Q3) Design a **divide-and-conquer** algorithm to find **Minimum** Number in an unsorted array of N elements (**without using any of the sorting algorithms covered in the class**). **Setup** a recurrence relation for your algorithm. **(7 marks)**

Solution:

```
findMin(A, left, right)
    if left == right  // Base case: single element
        return A[left]
    else
        mid = (left + right) / 2
        leftMin = findMin(A, left, mid)      // Recursively find min in left half
        rightMin = findMin(A, mid + 1, right) // Recursively find min in right half
        return min(leftMin, rightMin)        // Combine results (Student can make comparison between leftMin and rightMin)
```

$T(n)=2T(n/2)+1$ for n>1, T(n)=0 for n=1

Q4) Consider the following recursive algorithm: **(5 marks)**

```
Algorithm Q(n)
//Input: A positive integer n
if n =1
    return 1
else
    return Q(n − 1) + 2 ∗ n – 1
```

a) **Setup** a recurrence relation for the number of multiplications made by this algorithm.
b) **Solve** the recurrence relation using backward substitution and find Big O.

Solution:

a. $M(n)= 0$ for $n=1$, $M(n)= M(n-1) + 1$

b. $M(n)=M(n-1)+1$
$M(n-1)=M(n-2)+1$
$M(n-2)=M(n-3)+1$
$M(n)=[M(n-3)+1]+2=M(n-3)+3$

Using base case, when $n-k=1 \Rightarrow k=n-1$
$\Rightarrow M(n)=M(1)+(n-1)$

$=0+(n-1)$

$M(n)= n-1$

$O(n)$

Q5) Given the following array, A= -3 ,8, 4, 22, 13, 6, 20, 9 **(6 marks)**

a) Apply Quick Sort algorithm to sort the array elements in ascending order **(show detailed steps)**

b) Compare Quick sort to Insertion Sort in terms of best and worst cases.

Solution:

a.

Step 1: Initial Call (Full Array)

- **Pivot = -3 (first element)**
- **Since all elements are > -3, the partition is:**
  - **Left subarray: []**
  - **Right subarray: [8, 4, 22, 13, 6, 20, 9]**
- **Result: [-3] + sorted([8, 4, 22, 13, 6, 20, 9])**

Step 2: Sort Right Subarray [8, 4, 22, 13, 6, 20, 9]

- **Pivot = 8**
- **Partitioning:**
  - **Left (≤ 8): [4, 6]**
  - **Right (> 8): [22, 13, 20, 9]**
- **Result: [4, 6] + [8] + sorted([22, 13, 20, 9])**

Step 3: Sort [4, 6]

- **Pivot = 4**
- **Partitioning:**
  - **Left (≤ 4): []**
  - **Right (> 4): [6]**
- **Result: [4] + [6] (already sorted)**

Step 4: Sort [22, 13, 20, 9]

- **Pivot = 22**
- **Partitioning:**
  - **Left (≤ 22): [13, 20, 9]**
  - **Right (> 22): []**
- **Result: sorted([13, 20, 9]) + [22]**

Step 5: Sort [13, 20, 9]

- **Pivot = 13**
- **Partitioning:**
  - **Left (≤ 13): [9]**
  - **Right (> 13): [20]**
- **Result: [9] + [13] + [20]**

Final Sorted Array:
[-3, 4, 6, 8, 9, 13, 20, 22]

b. Quick Sort vs. Insertion Sort (Best & Worst Cases)

Aspect	Quick Sort	Insertion Sort
Best Case	O(nlogn)	O(n)
Worst Case	O(n2)	O(n2)

Q6) Show that the worst case running time of MAX-HEAPIFY on a heap of size *n* is ***log n*** (Hint: For a heap with n nodes, give node values that cause MAX-HEAPIFY to be called recursively at every node on a simple path from the root down to a leaf.) **(4 marks)**

Solution:

Constructing the Worst-Case Heap:

- **Heap Structure:**
  - A **complete binary tree** where the **root has the smallest value**, and each child is larger than its parent.
  - Example for n=7(height h=2):

```
    1
  /   \
 2     3
/ \   / \
4  5 6   7
```

    - If MAX-HEAPIFY is called on the root (1), it will **always swap with the larger child** (left or right) until it reaches a leaf.
- **Recursive Calls:**
  - At each level, the smallest element moves down, requiring **O(h) swaps**, where *h* is the height of the heap.

Time Complexity Analysis:

1. **Height of the Heap (*h*):**
   - For a heap of size *n*, the height is $h=\lfloor \log_2 n \rfloor$.
2. **Worst-Case Work:**
   - Each recursive call does $O(1)$ work (comparisons + swaps).
   - The maximum number of calls = height of the heap = $\log 2n$.
3. **Recurrence Relation:**

$T(n) \leq T(2n3)+O(1)$

   - The worst-case occurs when the **subtree sizes are as unbalanced as possible** (one subtree has 2n/3 nodes).
   - By the **Master Theorem**, this solves to $O(\log n)$.

Conclusion:

- The worst-case running time of MAX-HEAPIFY is ***O*(log*n*)** because:
  1. The recursion depth is bounded by the heap height ($\log 2n$).
  2. Each recursive call performs $O(1)$ work.

Why This is Tight:

- The example heap above forces MAX-HEAPIFY to traverse the **entire height** of the heap, proving that $\Omega(\log n)$ is a **lower bound**.
- Thus, the worst-case runtime is **$\Theta(\log n)$**.