

Name: _____ Alpha: _____ Class Period: _____

INSTRUCTIONS:

- Read every question carefully before answering.
- Write clearly and neatly. If your handwriting cannot be easily read, you risk losing credit for your answer.
- This exam is closed-book, closed-notes, and no electronic devices are permitted.

Page	Points	Score
2	10	
3	8	
4	7	
5	12	
6	5	
7	7	
8	5	
9	6	
Total:	60	

1. Fill in the following table with the correct run times (using Big-O notation and amortized solutions where appropriate) for each method of each ADT, as implemented with each data structure. Assume everything is implemented the best way we know how, and explicitly state where amortization is applied. [10 pts]

		Array	Linked List
STACK	push(e)	$O(1)$ amortized	$O(1)$
	pop()	$O(1)$	$O(1)$
	peek()	$O(1)$	$O(1)$
QUEUE	enqueue(e)	$O(1)$ amortized	$O(1)$
	dequeue()	$O(1)$	$O(1)$
	head()/peek()	$O(1)$	$O(1)$
LIST	get(i)	$O(1)$	$O(n)$
	set(i,e)	$O(1)$	$O(n)$
	add(i,e)	$O(n)$	$O(n)*$
	remove(i)	$O(n)$	$O(n)*$

* Note: $O(1)$ for Linked List if reference to target node is already given.

2. Circle **True** or **False** for each of the following statements based on concepts covered in class.

- (A) In Python List, the expression `a is b` evaluates to **True** if and only if `a` and `b` store the exact same memory address (identity). **True** / **False** [1 pts]
- (B) In a singly linked list, deleting the last element (tail) when given only the head pointer takes $O(1)$ time. **True** / **False** [1 pts]
- (C) A function $f(n) = 4n^2 + 100n$ belongs to $O(n^3)$, mathematically speaking. **True** / **False** [1 pts]
- (D) Every recursive function must contain both a base case and a recursive case to prevent infinite recursion and stack overflow. **True** / **False** [1 pts]
- (E) Amortized analysis measures the performance of an operation assuming that inputs are randomly distributed (average-case probability). **True** / **False** [1 pts]

3. Consider the standard linear search function below that searches for an element in an unsorted list:

```
def search(L, target):
    for i in range(len(L)):
        if L[i] == target:
            return i
    return -1
```

- (A) What does n represent in the context of analyzing this algorithm? [1 pts]

Solution: n represents the total number of elements in the input list `L` (`len(L)`).

- (B) What circumstances constitute the algorithm's **best case**, and what is its step count / Big-O runtime in that case? [1 pts]

Solution: The target element is found at the very first index (`L[0]`). The runtime is $O(1)$ (constant time).

- (C) What circumstances constitute the algorithm's **worst case**, and what is its Big-O runtime? [1 pts]

Solution: The target element is at the very last index (`L[n-1]`) or is not present in the list at all. The algorithm must inspect every element, resulting in an $O(n)$ runtime (linear time).

4. Rank the following Big-O growth rates in order from **fastest growing (slowest execution time)** to **slowest growing (fastest execution time)**, with 1 being the absolute slowest runtime (worst scalability) and 6 being the fastest runtime (best scalability): [3 pts]

2 $O(n^2)$ (Quadratic)

6 $O(1)$ (Constant)

1 $O(2^n)$ (Exponential)

4 $O(n)$ (Linear)

5 $O(\log n)$ (Logarithmic)

3 $O(n \log n)$ (Linearithmic)

5. If $f(n) = 8n^2 + 12n \log_2(n) + 100$, circle **ALL** definitions of $g(n)$ such that $f(n) \in O(g(n))$ (mathematically speaking). [2 pts]

a. $g(n) = n$

b. $g(n) = n \log(n)$

c. $g(n) = n^2$

d. $g(n) = n^3$

e. $g(n) = 2^n$

6. If $f(n) = 15 \log_2(n) + 42$, circle **ALL** definitions of $g(n)$ such that $f(n) \in O(g(n))$ (mathematically speaking). [2 pts]

a. $g(n) = 1$

b. $g(n) = \log(n)$

c. $g(n) = n$

d. $g(n) = n^2$

e. $g(n) = 2^n$

7. Consider a standard implementation of a singly linked list storing integers. Complete the following method inside the `LinkedList` class that calculates and returns the **sum of all even integers** present in the list. [6 pts]

For example, if the list contains $1 \rightarrow 4 \rightarrow 7 \rightarrow 8 \rightarrow 10 \rightarrow \text{None}$, the function should return 22 ($4 + 8 + 10$). If no even integers exist or the list is empty, return 0. Do not assume the list is sorted. Your solution **must be iterative**.

```
def sumEvens(self):
```

Solution:

```
def sumEvens(self):
    total = 0
    current = self.head
    while current is not None:
        if current.data % 2 == 0:
            total += current.data
        current = current.next
    return total
```

8. Your task is the same as in Question 7, but now your solution **must be recursive**. [6 pts]

```
def sumEvens(self):
    return self._sumEvens(self.head)

def _sumEvens(self, node):
```

Solution:

```
def _sumEvens(self, node):
    if node is None:
        return 0
    if node.data % 2 == 0:
        return node.data + self._sumEvens(node.next)
    return self._sumEvens(node.next)
```

9. For each of the following code snippets, choose the correct Big- O runtime.

(A) `total = 0`

[1 pts]

```
i = n
while i > 1:
    for j in range(n):
        total += 1
    i = i // 2
```

- a. $O(n)$
- b. $O(n \log n)$**
- c. $O(\log n)$
- d. $O(n^2)$
- e. $O(2^n)$

(B) `total = 0`

[1 pts]

```
for i in range(n):
    for j in range(50):
        total += 1
for k in range(2 * n):
    total += 1
```

- a. $O(1)$
- b. $O(n)$**
- c. $O(n \log n)$
- d. $O(n^2)$
- e. $O(n^3)$

(C) `total = 0`

[1 pts]

```
for i in range(n):
    for j in range(i, n):
        total += 1
```

- a. $O(n)$
- b. $O(n \log n)$
- c. $O(\log n)$
- d. $O(n^2)$**
- e. $O(2^n)$

10. Write the recurrence relation for the following algorithm. Assume `node` points to a standard Linked List Node:

[2 pts]

```
def process_list(node):
    if node is None:
        return 0
    return node.data + process_list(node.next)
```

Solution:

$$T(0) = c_0$$
$$T(n) = T(n-1) + c_1$$

11. Write the recurrence relation $T(n)$ for the following recursive function:

[2 pts]

```
def mystery(arr, low, high):
    if high - low <= 1:
        return arr[low]
    mid = (low + high) // 2
    left_val = mystery(arr, low, mid)
    right_val = mystery(arr, mid, high)
    return max(left_val, right_val)
```

Solution:

$$T(1) = c_0$$

$$T(n) = 2T(n/2) + c_1$$

12. Find the Big- O bound for the following recurrence relation by repeatedly expanding terms. **SHOW YOUR WORK.**

[5 pts]

$$T(1) = c_0$$

$$T(n) = T(n-1) + c_1 \quad \text{for } n > 1$$

Solution: Unrolling the terms step by step:

$$T(n) = T(n-1) + c_1$$

$$T(n) = (T(n-2) + c_1) + c_1 = T(n-2) + 2c_1$$

$$T(n) = (T(n-3) + c_1) + 2c_1 = T(n-3) + 3c_1$$

$$\vdots$$

$$T(n) = T(n-k) + k \cdot c_1$$

To reach the base case $T(1)$, set $n - k = 1 \implies k = n - 1$:

$$T(n) = T(1) + (n-1)c_1$$

$$T(n) = c_0 + n \cdot c_1 - c_1$$

$$T(n) \in O(n)$$

13. Assume a `Node` and `LinkedList` class implemented as usual. Implement the method `remove_front(self)` inside the `LinkedList` class, which removes the front node and returns its data value. If the list is empty, return `None`.

[3 pts]

```
def remove_front(self):
```

Solution:

```
def remove_front(self):
    if self.head is None:
        return None
    val = self.head.data
    self.head = self.head.next
    return val
```

14. Assume an empty Stack `s`. What output will be printed by the following sequence of operations?

[1 pts]

```
s.push(10)
s.push(20)
x = s.pop()
s.push(30)
s.push(40)
y = s.peek()
z = s.pop()
print(x, y, z)
```

Solution: Output: 20 40 40

Trace: $x = 20$ (stack has [10]). Push 30, 40 (stack has [10, 30, 40]). $y = 40$ (peek). $z = 40$ (pop).

15. Assume an empty Queue `q`. What output will be printed by the following sequence of operations?

[1 pts]

```
q.enqueue(10)
q.enqueue(20)
x = q.dequeue()
q.enqueue(30)
y = q.peek()
z = q.dequeue()
print(x, y, z)
```

Solution: Output: 10 20 20

Trace: $x = 10$ (queue has [20]). Enqueue 30 (queue has [20, 30]). $y = 20$ (peek front). $z = 20$ (dequeue front).

16. A Queue is implemented using an **Array** of capacity 8. The queue currently contains 5 elements (**size** = 5), and the first item (**front**) is located at index 5. [3 pts]

The current array layout is shown below, where **X** indicates a filled slot and **–** indicates an empty slot:

0	1	2	3	4	5	6	7
X	X	–	–	–	X	X	X
					↑ front		

- (A) State the array index where the next **enqueue** operation will insert a new element. [1 pts]

Solution: Index **2**.

- (B) Write down the general formula (using **front**, **size**, and **capacity**) used to calculate this insert index in a circular array. [1 pts]

Solution:

$$\text{insert_index} = (\text{front} + \text{size}) \pmod{\text{capacity}}$$

Calculation for this example: $(5 + 5) \pmod{8} = 10 \pmod{8} = 2$.

- (C) Explain what changes will occur to the queue's state variables (**front**, **size**) immediately after the new element is successfully enqueued. [1 pts]

Solution: The **size** variable will increment from 5 to **6**. The **front** pointer remains unchanged at index **5** (since **front** only moves during **dequeue** operations).

17. What was your effort level when preparing for this 6-week exam, expressed in Big-*O* notation ($O(1)$, $O(\log n)$, $O(n)$, $O(n^2)$, or $O(2^n)$)? Briefly explain your answer in 1 sentence. **NO RIGHT OR WRONG**