AddisCoder Final Exam

Congratulations on reaching the final stage of the AddisCoder 2025 program!

This exam will cover material from the entire 4-week course. You have 2 hours to complete this quiz.

`Student Name: _____`

`Lab _____`

Grading Table (Staff Use Only)

Question	Marks
1	
2	
3	
4	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
Total	

SOLUTIONS - PAGE 1

Answers are overlaid in blue in the response areas on the following pages.

Original question page preserved at left

Problem 1: Multiple Choice

1.1 What is the **output** of `bool("False")`?

a) `True`

b) `False`

c) `''`

d) `'True'`

1.2 How do you get the **third to last** element of a list `lst`?

a) `lst[len(lst) - 2]`

b) `lst(-3)`

c) `lst[-3]`

d) `lst[len(lst)]`

1.3 What happens when `printList([4, 1, 5, 7, 2])` is executed? (Hint: look at the syntax carefully!)

```
def printList(lst):
    for i in range(lst):
        print(lst[i])

printList([4, 1, 5, 7, 2])
```

a) All numbers in the list are printed.

b) All of the indices of the list (0 to 4) are printed.

c) Infinite loop. The code runs but does not terminate.

d) Error. The code does not run at all.

SOLUTIONS - PAGE 2

1.1 a) True 1.2 c) lst[-3]
1.3 d) Error (range expects an integer).

1.4 Which of the following sorting algorithms has the best worst-case time complexity?

a) SelectionSort

b) BubbleSort

c) MergeSort

d) InsertionSort

1.5 In QuickSort, which pivot choice is often used in practice to improve average performance and avoid worst-case scenarios?

a) The smallest element

b) The first element

c) The largest element

d) A random element

1.6 When binary search is used to search in a list, which of the following statements is false?

a) Binary search splits the list to reduce the search space at each step

b) Binary search requires the list to be sorted to work correctly

c) Binary search always finds the item in a single comparison

d) Binary search can work on lists with duplicate elements

1.7 If V is the number of vertices and E is the number of edges in a graph, then what is the time complexity of performing Breadth First Search (BFS) on the graph?

a) $O(V)$

b) $O(V^2)$

c) $O(V * E)$

d) $O(V + E)$

1.8 Which of the following has the **highest growth rate** as n becomes very large?

a) n^4

b) $n^2 \log(n)$

c) 3^n

d) $\log(n)$

SOLUTIONS - PAGE 3

1.4 c) MergeSort
1.5 d) A random element
1.6 c) 1.7 d) O(V + E)
1.8 c) 3^n

Original question page preserved at left

1.9 Which of the following statements about the Bellman-Ford algorithm is **not true**?

a) The Bellman-Ford algorithm can handle graphs with negative weight edges.

b) The Bellman-Ford algorithm has a time complexity of $O(V^2)$ where V is the number of vertices.

c) The Bellman-Ford algorithm can detect negative weight cycles in a graph.

d) The Bellman-Ford algorithm uses dynamic programming to find the shortest paths from a single source to all other vertices in a graph.

Problem 2: Recursion Tracing

What will the following recursive function return for `f([2, 4, 6])` ?

```
def f(lst):
  if len(lst) == 1:
    return lst[0]
  else:
      return lst[0] * f(lst[1:])
```

Answer: _____

SOLUTIONS - PAGE 4

1.9 b) False: Bellman-Ford is O(VE), not O(V^2) in general.
Problem 2: 2 * 4 * 6 = 48.

Original question page preserved at left

Problem 3: Building a Dictionary

Which of the following code snippets produces the result `{1: 1, 2: 8, 3: 27, 4: 64}`?

a)

```
result = {}
for x in range(1, 5):
    result[x] = x**3
```

b)

```
result = {}
for x in range(0, 3):
    result[x] = (x+1)**3
```

c)

```
result = {}
for x in range(1, 5):
    result[x**3] = x
```

d)

```
result = {}
for x in range(2, 5):
    result[x] = x**2
```

SOLUTIONS - PAGE 5

a) result[x] = x**3 for x in range(1, 5)

Problem 4: Printing

What will be printed after running the following code?

```
x = "Hello"
y = "World"
x = x + " " + y
print(x)
y = y[:3]
print(y)
print(x + " " + y)


Output:
```

Problem 5: Time Complexity

What is the time compexity of the following functions? **Circle the correct choice** for each.

5.1

Note: Assume that multiplication is a constant time operation.

```
def sum_squares_to_one(n):
    if n == 1:
        return 1
    return n**2 + sum_squares_to_one(n - 1)
```

a) $O(1)$

b) $O(n)$

c) $O(n^2)$

d) $O(2^n)$

SOLUTIONS - PAGE 6

Problem 4 output:
Hello World
Wor
Hello World Wor

5.1 b) O(n)

Original question page preserved at left

5.2

Note: **LinearFun(n)** has linear time compexity.

```
def func2(n):
    x = 0
    for i in range(n):
        LinearFun(n)
        x += i
    return x
```

a) $O(\log(n))$

b) $O(n \log(n))$

c) $O(n)$

d) $O(n^2)$

5.3

Assume `lst` contains any integer number.

```
def func3(lst):
    total = 0
    for i in range(len(lst)):
        total += lst[i]
        if total > 100:
            return total
    return total
```

a) $O(1)$

b) $O(n)$

c) $O(n^2)$

d) $O(\log(n))$

SOLUTIONS - PAGE 7

5.2 d) O(n^2)
5.3 b) O(n)

5.4

Note: **LinearFun(n)** has linear time compexity.

```
def func4(n):
    x = n
    LinearFun(n)
    while x > 0:
        LinearFun(n)
        x = x // 4
```

a) $O(\log(n))$

b) $O(n)$

c) $O(n \log(n))$

d) $O(n^2)$

5.5

Assume that addition, dictionary lookup, and dictionary update are constant-time operations.

```
def func_5(n, mem={}):
    if n in mem:
        return mem[n]

    if n == 0 or n == 1:
        return 1
    if n == 2:
        return 2

    mem[n] = func_5(n - 1, mem) + func_5(n - 2, mem) + func_5(n - 3, mem)
    return mem[n]
```

a) $O(n)$

b) $O(3^n)$

c) $O(n^3)$

d) $O(2^n)$

SOLUTIONS - PAGE 8

5.4 c) O(n log n)
5.5 a) O(n) with memoization

Original question page preserved at left

Problem 6: Fix the Bug

The following function takes in an input list of integers `lst` and intends to check if all of the elements in the list are negative. But there is a bug!

```
def check_all_negative(lst):
    for i in range(len(lst)):
        if lst[i] >= 0:
            return False
        else:
            return True
```

6.1 What is the bug with the above code?

Answer:

6.2 Fix the bug and write the correct function below.

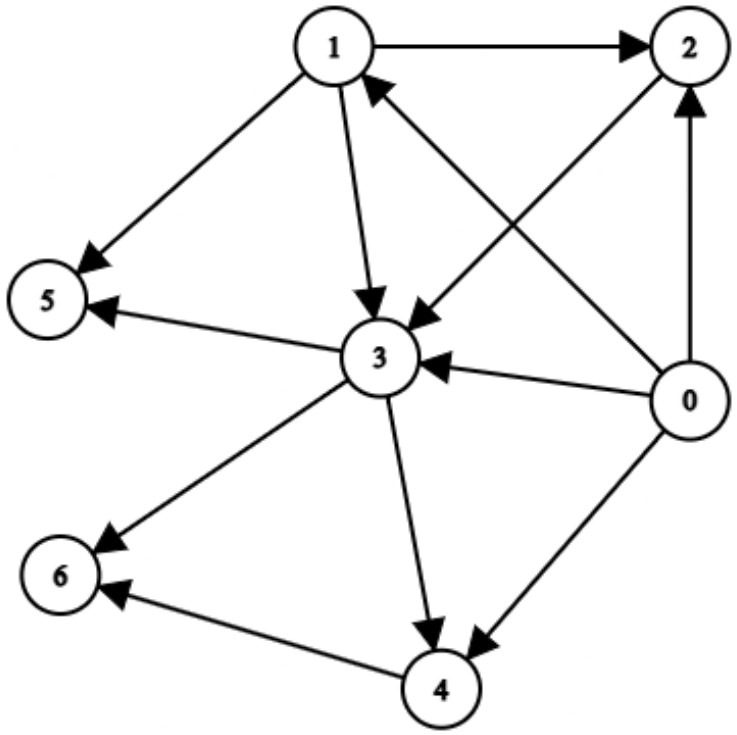
SOLUTIONS - PAGE 9

The else is inside the loop, so the function returns True after checking only
the first negative item. Move return True after the loop.

```
def check_all_negative(lst):
    for x in lst:
        if x >= 0:
            return False
    return True
```

Problem 7: Different Ways to Search

7.1 Write the graph below as an adjacency list.



Answer:

SOLUTIONS - PAGE 10

0: [1, 2, 3, 4]
1: [2, 3, 5]
2: [3]
3: [4, 5, 6]
4: [6]
5: []
6: []

7.2 Write the order in which **depth-first search** would visit nodes in the same directed graph above, starting from vertex 0. Break ties by assuming neighbors are visited in increasing order (smaller numbers are visited before larger numbers).

Answer: _____

7.3 Write the order in which **breadth-first search** would pop nodes off the queue in the same directed graph above, starting from vertex 0. Break ties by assuming neighbors are looped over in increasing order.

Answer: _____

SOLUTIONS - PAGE 11

7.2 DFS: 0, 1, 2, 3, 4, 6, 5
7.3 BFS dequeue order: 0, 1, 2, 3, 4, 5, 6

Original question page preserved at left

Problem 8: Using Memoization

Let's say you want to write a program that returns the nth Padovan number. Each term is the sum of the term two steps before and the term three steps before it: `P(n) = P(n - 2) + P(n - 3)`. The first three numbers are `1`, `1`, and `1`. Below is a function that finds the nth Padovan number.

```
def padovan(n):
    if n == 0 or n == 1 or n == 2:
        return 1
    else:
        return padovan(n - 2) + padovan(n - 3)
```

8.1 What is the time complexity of this function?

a) linear

b) quadratic

c) cubic

d) exponential

8.2 Write the function `padovan_memo(n)` that uses memoization to speed up the function above.

SOLUTIONS - PAGE 12

8.1 d) exponential

```
def padovan_memo(n, mem=None):
    if mem is None:
        mem = {}
    if n <= 2:
        return 1
    if n not in mem:
        mem[n] = padovan_memo(n-2, mem) +
          padovan_memo(n-3, mem)
    return mem[n]
```

Problem 9: Special Numbers

9.1 Write a function `special(n)` that takes an integer `n` and does the following:

- If `n` is less than 100 or greater than 999, print `"OUT OF RANGE"`
- If the first digit is equal to the last digit (e.g. `121`, `585`, `464`, etc.), print `"SPECIAL!"`
- In all other cases, print "NORMAL: " followed by the number `n`

Example:

```
special(121) → "SPECIAL!"

special(385) → Normal: 385

special(99) → "OUT OF RANGE"
```

9.2 Write a program that runs special(n) on numbers from 400 to 800, exclusive of both 400 and 800.

SOLUTIONS - PAGE 13

```
def special(n):
    if n < 100 or n > 999:
        print("OUT OF RANGE")
    elif n // 100 == n % 10:
        print("SPECIAL!")
    else:
        print("NORMAL:", n)

for n in range(401, 800):
    special(n)
```

Original question page preserved at left

Problem 10: Merge Sort

Write `merge_sort(lst)` to implement the Merge Sort algorithm. Assume that `lst` is an unsorted list of integers and return the sorted list.

What is the time complexity of your program if `lst` is of length `n`?

SOLUTIONS - PAGE 14

```
def merge(left, right):
    out = []
    i = j = 0
    while i < len(left) and j < len(right):
        if left[i] <= right[j]:
            out.append(left[i]); i += 1
        else:
            out.append(right[j]); j += 1
    return out + left[i:] + right[j:]

def merge_sort(lst):
    if len(lst) <= 1:
        return lst[:]
    mid = len(lst) // 2
    return merge(merge_sort(lst[:mid]),
      merge_sort(lst[mid:]))
```

Time complexity: O(n log n)

Original question page preserved at left

Answer: _____

Problem 11: Maximum Difference

Write a function `max_diff(lst)` that takes a list of integers and returns the largest difference between two numbers, where the larger number appears before the smaller number in the list.

If no such pair exists, the function should return 0.

Example:

```
max_diff([22, 14, 17, 5, 13]) # returns 17 (22 - 5)
max_diff([1, 22, 10, 5, 100]) # returns 17 (22 - 5)
```

SOLUTIONS - PAGE 15

```
def max_diff(lst):
    if not lst:
        return 0
    largest_seen = lst[0]
    best = 0
    for x in lst[1:]:
        best = max(best, largest_seen - x)
        largest_seen = max(largest_seen, x)
    return best
```

Time complexity: O(n); extra space: O(1).

Problem 12: Updating the Cache

Consider the following program:

```
fact_cache = [-1] * 7

def factorial(n, mem):
    if mem[n] != -1:
        return mem[n]

    if n == 0 or n == 1:
        return 1

    mem[n] = n * factorial(n - 1, mem)
    return mem[n]

factorial(5, fact_cache)
```

What is the value of `fact_cache` after this program runs?

Answer: _____

SOLUTIONS - PAGE 16

[-1, -1, 2, 6, 24, 120, -1]
The base cases return 1 without storing it in indices 0 or 1.

Problem 13: Shortest Path

You are given an N × M 2D maze from a video game, represented as a list of lists. Each cell maze[i][j] contains one of the following:

- 'O': an empty cell
- 'X': a wall
- 'S': the start
- 'E': the end

You want to use **Breadth-First Search (BFS)** to find the shortest path from the start to the end, or determine if it's impossible. You can move up, down, left, or right into adjacent empty cells, and moving into 'S' and 'E' is also allowed.

Answer the following scenarios:

1. **Basic case :** How would you define the vertices and edges of the graph? How many vertices are there?

2. **Bombs :** Suppose you start with 3 lives, and some cells contain bombs 'B'. Entering a bomb cell decreases your lives by 1; reaching 0 means death. How does this change your definition of vertices and edges, and how many vertices are possible now?

3. **Keys and doors :** Now add up to 3 locked doors ('F', 'G', 'H') and 3 matching keys ('f', 'g', 'h'). You can only pass through a door if you've collected its corresponding key. How do vertices and edges need to be defined in this case, and what is the maximum number of vertices?

SOLUTIONS - PAGE 17

1. Basic: a vertex is a traversable cell (r, c). Edges connect orthogonally adjacent traversable cells. At most N*M vertices.

2. Bombs: use state (r, c, lives), where lives is 1, 2, or 3. An edge moves to a neighbor and subtracts 1 if it is B; moves leaving 0 lives are not allowed. At most 3*N*M vertices.

3. Keys/doors: use state (r, c, key_mask), a 3-bit mask. Moving onto a key
sets its bit; entering F/G/H requires the matching bit. At most 8*N*M vertices. Edges represent legal one-step moves with the updated key mask.

Original question page preserved at left

Problem 14: Find the Missing Cube

You're given a list lst of length n, which contains the cubes of integers from 0 to n inclusive, but with one cube missing.

Write a `function missing_cube(lst)` that returns a pair `(number, cube)`, where `number` is the integer whose cube is missing, and `cube` is its cube.

Example:

```
missing_cube([0, 1, 8, 64])  # returns (3, 27) because 3^3 is missing
missing_cube([1, 0])         # returns (2, 8) because 2^3 is missing
```

What is the time complexity of your solution if `lst` is of length `n`?

SOLUTIONS - PAGE 18

```
def missing_cube(lst):
    n = len(lst)
    present = set(lst)
    for number in range(n + 1):
        cube = number ** 3
        if cube not in present:
            return (number, cube)
```

Time complexity: O(n) expected; extra space: O(n).

```
Answer: _____
```

Problem 15: Counting Distinct Substrings

Write a function `count_distinct_substrings_k(s, k)` that takes in a string s and an integer k, and returns the number of distinct substrings of length k in the string.

Note: A substring is a continuous block of characters from the string.

Example:

```
count_distinct_substrings_k("ababa", 2)  # returns 2
# The distinct substrings of length 2 are: "ab", "ba"

count_distinct_substrings_k("abcdef", 4)
# returns 3
# Distinct substrings of length 4 are: "abcd", "bcde", "cdef"
```

SOLUTIONS - PAGE 19

```
def count_distinct_substrings_k(s, k):
    if k < 0 or k > len(s):
        return 0
    return len({s[i:i+k] for i in range(len(s) - k + 1)})
```

Time: O((n-k+1)*k) with ordinary slicing/hashing; O(n) when k is fixed.
Extra space: O(number of distinct substrings * k).