Name: _____

1. Variable Tracing

For each of the following questions, write the output without running the code.

a. What will the following code print?

```
numbers = [4, 9, 2, 7]

a = numbers[1] - numbers[0]
b = a + numbers[2]
numbers[0] = b
c = numbers[0] + numbers[3]
a = c - b

print(a)
print(b)
print(c)
print(numbers)
```

b. What will the following code print?

```
values = [6, 3, 8, 1]

x = values[0]
y = values[2] - values[1]
values[3] = x + y
z = values[3] - values[0]
x = z + values[1]

print(x)
print(y)
print(z)
print(values)
```

2. Code Tracing

For each part, write exactly what the code prints. If the code prints a list, include the brackets, commas, and quotation marks exactly as Python would display them.

a. What will the following code print?

```
values = [3, 6, 9, 12, 15, 18, 21]
print(values[1:6:2])
print(values[5:1:-2])
```

b. What will the following code print?

```
def mystery(n):
    if n <= 1:
        print(n)
        return
    print(n)
    mystery(n - 2)
    print(n + 1)
mystery(5)
```

c. What will the following code print?

```
def puzzle(values):
    if len(values) == 0:
        return 0
    result = values[0] + puzzle(values[2:])
    print(result)
    return result
print("Final:", puzzle([2, 4, 6, 8, 10]))
```

3. Binary Search

Suppose we had the following sorted list:

```
numbers = [1, 4, 6, 10, 17, 19, 26, 35, 41, 48, 52, 60, 67, 71, 85]
```

a. Write the indices that are visited when searching for `67`.

b. Write the indices that are visited when searching for `18`.

c. Write the indices that are visited when searching for `4`.

d. Write the indices that are visited when searching for `100`.

e. What is the **best-case** runtime of binary search? Briefly explain your answer.

f. What is the **worst-case** runtime of binary search? Briefly explain your answer.

4. Complexity

For each of the following questions, analyze the runtime in terms of `N`.

```
def mystery(N):
    for i in range(N):
        print(i)
```

Runtime: $O($_____$)$

```
def mystery(N):
    for i in range(N):
        for j in range(N):
            print(i + j)
```

Runtime: $O($_____$)$

```
def mystery(N):
    for i in range(N):
        print(i)

    for j in range(N):
        print(j)
```

Runtime: $O($_____$)$

```
def mystery(N):
    while N > 1:
        print(N)
        N //= 2
```

Runtime: $O($_____$)$

Give the best-case and worst-case runtime in terms of `N`. Assume `ping(i)` runs in $O(1)$ and returns a `bool`.

```
def mystery(N):
    for i in range(N):
        if ping(i):
            break
```

Best Case Runtime: $O($_____$)$

Worst Case Runtime: $O($_____$)$

5. Find First

Write a function `findFirst(numbers, target)` that returns the index of the ***first*** occurrence of `target` in the list. If `target` does not appear in the list, return `-1`.

Examples:

- `findFirst([4, 8, 2, 8], 8)` returns `1`
- `findFirst([5, 7, 9], 4)` returns `-1`
- `findFirst([], 3)` returns `-1`

Part I: Iterative

Complete the function below using iteration.

```
def findFirst(numbers, target):

    for _____________________________________:

        if ______________________________:

            return ________________________

    return ______________________________
```

Part II: Recursive

Now, complete the recursive helper function below. Your solution must not use loops.

```
def findFirst(numbers, target):

    return helper(numbers, target, 0)

def helper(numbers, target, index):

    if ___________________________________:

        return ___________________________

    if ___________________________________:

        return ___________________________

    return ________________________________________
```

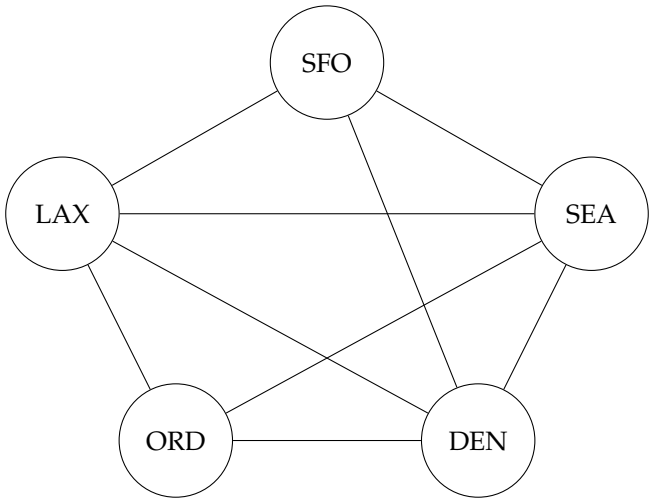
6. Flight Network Graph

Five airports are connected by direct flights:

- Los Angeles (LAX)
- San Francisco (SFO)
- Seattle (SEA)
- Denver (DEN)
- Chicago (ORD)

An **edge** (a line) between two airports means there is a direct flight between them.

Consider the following undirected graph:

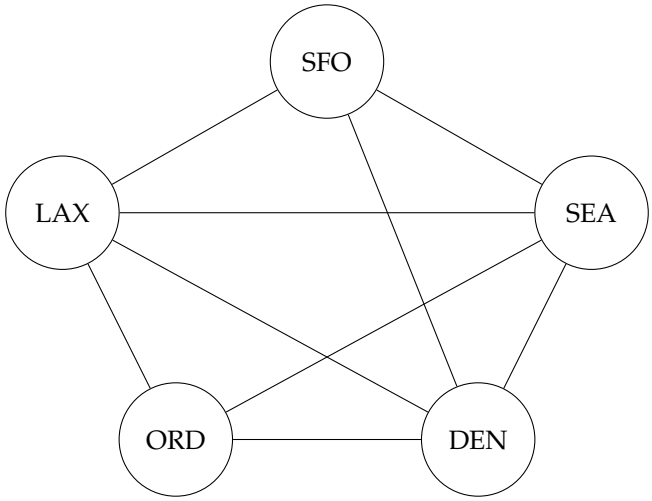


a. How many direct flights leave `SFO`?

b. Complete the adjacency list for the flight network.

```
LAX: ____________________________________________

SFO: ____________________________________________

SEA: ____________________________________________

DEN: ____________________________________________

ORD: ____________________________________________
```

The graph is shown again for ease of reference.



c. Complete the adjacency matrix. Write `1` if there is a direct flight between two airports and `0` otherwise.

	LAX	SFO	SEA	DEN	ORD
LAX					
SFO					
SEA					
DEN					
ORD					

d. Give one cycle in the graph. Write the airports in the order they are visited.

e. Perform a depth-first search (DFS) on the graph starting from the airport `SFO`. Write down the order in which vertices are visited. Break ties by selecting the airport that comes first alphabetically.

DFS Order: _____ _____ _____ _____ _____

d. Perform a breadth-first search (BFS) starting at `SFO` and list the order in which vertices are visited. Break ties by selecting the airport that comes first alphabetically.

BFS Order: _____ _____ _____ _____ _____

7. Sorting Algorithms

Complete each of the sorting algorithms below by filling in the blanks.

Part I: Selection Sort

Fill in each blank to complete the Selection Sort implementation below.

```
def selection_sort(L):

    for i in range(__________________):
        min_index = i
        for j in range(__________________):
            if ___________________:
                min_index = ___________________

        temp = ___________________
        L[i] = ___________________
        L[min_index] = ___________________

    return A
```

Part II: Merge Sort

Merge Sort recursively divides a list into two halves, recursively sorts each half, and then merges the two sorted halves together.

Assume the helper function `merge(left, right)` has already been implemented. It accepts two sorted lists and returns a single sorted list containing all of their elements.

Fill in each blank to complete the implementation below.

```
def merge_sort(L):

    if _______________________________:

        return ________________________

    mid = ____________________________

    left = ____________________________

    right = ___________________________

    sorted_left = ____________________________________

    sorted_right = ___________________________________

    return ___________________________________________
```