1. Linked Lists - Create a Swap Method

a.

```python
def swap(self) -> int:
    """
    This method swaps the node at Current with the node next to it.
    It returns 0 if the swapp was succesful and -1 if the input list is empty or there is nothing after current.
    """
    if (self.Current is None) or (self.Current.Next is None):
        return -1

    first = self.Current
    second = self.Current.Next

    if (first is self.Header):

        self.Header = second
    else:
        prev = self.Header
        while (prev is not None) and (prev.Next is not first):
            prev = prev.Next
        if (prev is None):
            return -1
        prev.Next = second

    first.Next = second.Next
    second.Next = first

    # Setting current to the next node
    self.Current = second
    return 0
```

b.

```
==== List created
Empty Linked List
==== Inserting 76 at Beginning
76 Current: 76
==== Inserting 88 at Beginning
88 76 Current: 76
==== Inserting 11 at Beginning
11 88 76 Current: 76
==== Inserting 34 at Beginning
34 11 88 76 Current: 76
==== Inserting 56 at Beginning
56 34 11 88 76 Current: 76
==== Inserting 91 at Beginning
91 56 34 11 88 76 Current: 76
==== Reseting the Current
91 56 34 11 88 76 Current: 91
==== Moving the Current to the next (circularly)
91 56 34 11 88 76 Current: 56
==== Moving the Current to the next (circularly)
91 56 34 11 88 76 Current: 34
The current is: 34
==== swapped current
91 56 11 34 88 76 Current: 11
```

2. Asymptotic Notations - Computing the Complexity
    a. O(n)
        i. Two loops run one after the other. Each runs n times, which simplifies to O(n).
    b. O(n^2)
        i. One loop is inside the other. So whatever n times the outer loop runs, the inner loop runs n times, so n * n would be n^2.
    c. O(n^3)
        i. There are about n recursive calls, and each call's tasks add up to about n^2 work, so n * n^2 = n^3.
    d. O(n^2 log n)
        i. We do about n^2 calls to a function that is O(log n) each, so we are looking at n^2 * log n.
3. Brute-Force Algorithm - Create the Difference of Two Sets

a.

In [17]:

```python
A = [20, 40, 70,30, 10, 80, 50, 90, 60]
B = [35, 45, 55, 60, 50, 40]
C = []

```

In [18]:

```python

def bruteforce_diff(A, B):
    for a in A:
        exists = False
        for b in B:
            if a == b:
                print(f"A={a}: checking B:{B}.\nMatch found so skipping")

                exists = True
                break
        if not exists:
            print(f"A={a}: checking B:{B}.\nNo match found adding A:{a} to C:{C}")
            C.append(a)
    return C


```

In [19]:

```python
bruteforce_diff(A, B)
```

```
A=20: checking B:[35, 45, 55, 60, 50, 40].
No match found adding A:20 to C:[]
A=40: checking B:[35, 45, 55, 60, 50, 40].
Match found so skipping
A=70: checking B:[35, 45, 55, 60, 50, 40].
No match found adding A:70 to C:[20]
A=30: checking B:[35, 45, 55, 60, 50, 40].
No match found adding A:30 to C:[20, 70]
A=10: checking B:[35, 45, 55, 60, 50, 40].
No match found adding A:10 to C:[20, 70, 30]
A=80: checking B:[35, 45, 55, 60, 50, 40].
No match found adding A:80 to C:[20, 70, 30, 10]
A=50: checking B:[35, 45, 55, 60, 50, 40].
Match found so skipping
A=90: checking B:[35, 45, 55, 60, 50, 40].
No match found adding A:90 to C:[20, 70, 30, 10, 80]
A=60: checking B:[35, 45, 55, 60, 50, 40].
Match found so skipping
```

b. Worst case scenario is if every element in A did not exist in B. That means every n element in A would be compared to every M element in B. O(n*m)

4. Recursion - Breadth First Search and Depth First Search

a.
```
adjacency_list = [
    ["B","D"], #A
    ["A","C","G"], #B
    ["A","B"], #C
    ["E","F"], #D
    ["F"], #E
    ["B"], #F
    ["C","F"], #g
]
```
b. DFS

   i.
```
DFS called for vertex A
Vertex A visited and received the stamp 0, current array: [0, -1, -1, -1, -1, -1, -1]
DFS called for vertex B
Vertex B visited and received the stamp 1, current array: [0, 1, -1, -1, -1, -1, -1]
DFS called for vertex C
Vertex C visited and received the stamp 2, current array: [0, 1, 2, -1, -1, -1, -1]
DFS called for vertex G
Vertex G visited and received the stamp 3, current array: [0, 1, 2, -1, -1, -1, 3]
DFS called for vertex F
Vertex F visited and received the stamp 4, current array: [0, 1, 2, -1, -1, 4, 3]
DFS called for vertex D
Vertex D visited and received the stamp 5, current array: [0, 1, 2, 5, -1, 4, 3]
DFS called for vertex E
Vertex E visited and received the stamp 6, current array: [0, 1, 2, 5, 6, 4, 3]
[0, 1, 2, 5, 6, 4, 3]
```

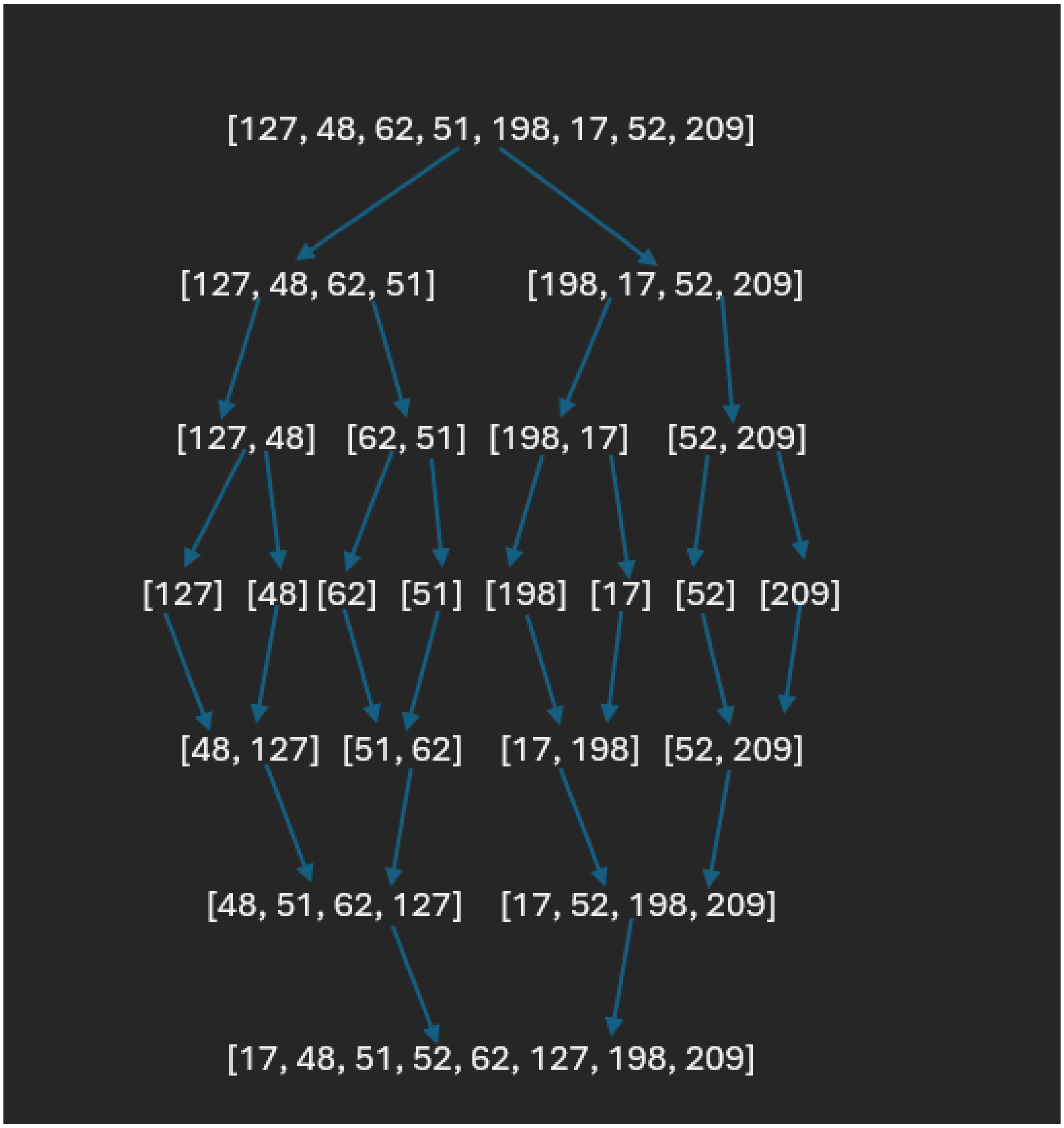
5. Recursion - Master Method
   a. T(n) = 4T(n/2) + n^3
      i. Since n^3 > n^2
      ii. O(n^3)
   b. T(n) = 4T(n/2) + n^2
      i. Since n^2 = n^2
      ii. O(n^2 log n)
   c. T(n) = 4T(n/2) + n
      i. Since n < n^2
      ii. O(n^2)
6. Decrease-and-Conquer Algorithm – Maximum Element in Array

a.

```python
def maximum(A, right):
    if right == 0:
        return A[0]

    prev = Maximum(A, right - 1)

    if A[right] >= prev:
        print(f"right={right}: {A[right]} greater than or equal to {prev} so keep {A[right]}")
        return A[right]
    else:
        print(f"right={right}: {A[right]} less than or equal to {prev} so keep {prev}")
        return prev


A = [17, 62, 49, 73, 26, 51]
max_num = maximum(A, len(A) - 1)
print("Max Number:", max_num)
```

```
right=1: 62 greater than or equal to 17 so keep 62
right=2: 49 less than or equal to 62 so keep 62
right=3: 73 greater than or equal to 62 so keep 73
right=4: 26 less than or equal to 73 so keep 73
right=5: 51 less than or equal to 73 so keep 73
Max Number: 73
```

7. Divide-and-Conquer Algorithms – Mergesort and Quicksort
    a. Worst case Big O:
        i. Mergesort: O(n log n)
        ii. Quicksort: O(n^2)
    b. Average case Big O:
        i. Mergesort: O(n log n)
        ii. Quicksort: O(n log n)



c.

Step	0	1	2	3	4	5	6	7
Original Array	127	48	62	51	198	17	52	209
Step 1 (l=0, r=7)	127	48	62	51	198	17	52	**209**
Step 2 (l=0, r=6)	127	48	62	51	198	17	**52**	
Step 3 (l=0, r=2)	48	51	**17**					
Step 4 (l=1, r=2)		51	**48**					
Step 5 (l=4, r=6)					198	62	**127**	
Sorted	17	48	51	52	62	127	198	209

d.

8.