1) Create Swap method

```python
def swap(self):
    if (self.Current.Next is None or self.Header is None):
        return -1
    else:

        # Nodes to swap
        current_next = self.Current.Next
        current_next_next = current_next.Next

        # Iterate until node before current node
        prev = self.Header
        while prev.Next != self.Current:
            prev = prev.Next

        # Current Node's Previous Swaps to Current_next
        prev.Next = current_next

        # Current node is now going to point to the next node that it swapped with
        self.Current.Next = current_next_next

        # Current_next is points to old current
        current_next.Next = self.Current

        return 0
```

## 2) Asymptotic Notations - Computing the Complexity

Answer the following questions explaining in a short sentence your rationale to find the answer. Consider that all relevant tasks to each algorithm is

**a) A given algorithm A is an iterative one that has two loops disposed sequentially (one after the other) each going over the n iterations. What is the complexity of A?**

- The complexity is O(n). When you are performing asymptotic analysis, sequential loops would result in $T(n) = c_1n + c_2n$. When you simplify this equation, it would result in $T(n) = n(c_3)$, which is linear in Big-Oh notation

**b) A given algorithm B is an iterative one that has two nested loops (one inside the other) each going over the n iterations. What is the complexity of B?**

- The complexity is $O(n^2)$. The outer loop would run n times and the inner loop would generate pairs which = $(n(n+1))/2$. The inner loop would dominate the time complexity resulting in the Big-Oh notation being $O(n^2)$.

**c) A given algorithm C is a recursive one that for a problem of size n executes O(n) recursive calls and to each recursive call it executes a certain number of tasks adding up a $O(n^2)$ complexity each. What is the complexity of C?**

- The complexity would be $O(n^2)$. When deciding complexity of recursive algorithms, you must look at T(n) = work outside recursive calls + work of recursive calls. Since the work outside of the recursive calls is larger, it dominates the complexity leading to $O(n^2)$ Big-Oh complexity.

**d) A given algorithm D is an iterative one that for a problem of size n executes $O(n^2)$ calls of a function that has complexity O(log n). What is the complexity of D?**

- The complexity would be $O(n^2 * \log n)$. The loop calls the function $O(n^2)$ times and for each iteration, it also does O(logn) work.

3) **Brute-Force Algorithm - Create the Difference of Two Sets**

Given two arrays of Integers A and B with len(A) = n and len(B) = m, create a third array C that includes all elements of A that are not in B. We write this operation as “A – B”; we call the operation set difference; and we call the result the difference of the two sets A and B (or simply “A minus B”). Assume that in each array, each element is listed only once (there are no duplicates within the same array), but the elements are not sorted.

a) Write a brute force function that uses nested for loops to repeatedly check if each element in A matches any of the elements in B. If the element from A does not match any element in B, then copy it into the next available slot of array C. Do not sort any of the arrays at any time. Example: With A = [2, 4, 6] and B = [3, 4, 5], your algorithm should produce C = [2, 6].

```python
def arrayDiff(A,B):
    # Array to hold set difference
    C = []

    # For each element in A, check it against each element in B
    # If the number of differences = the length of B, it is not in B
    for a in A:
        count = 0
        for b in B:
            if a != b:
                count += 1
            else:
                break # break early if there is one match
        if count == len(B):
            C.append(a)

    return C
```

b) Trace the algorithm with A = [20, 40, 70, 30, 10, 80, 50, 90, 60] B = [35, 45, 55, 60, 50, 40]

```
Array A element: 20 | Array B element 35  - Elements do not match. Increment Counter
Array A element: 20 | Array B element 45  - Elements do not match. Increment Counter
Array A element: 20 | Array B element 55  - Elements do not match. Increment Counter
Array A element: 20 | Array B element 60  - Elements do not match. Increment Counter
Array A element: 20 | Array B element 50  - Elements do not match. Increment Counter
Array A element: 20 | Array B element 40  - Elements do not match. Increment Counter
20 is not within B:[35, 45, 55, 60, 50, 40], adding to C | Array C = [20]

Array A element: 40 | Array B element 35  - Elements do not match. Increment Counter
Array A element: 40 | Array B element 45  - Elements do not match. Increment Counter
Array A element: 40 | Array B element 55  - Elements do not match. Increment Counter
Array A element: 40 | Array B element 60  - Elements do not match. Increment Counter
Array A element: 40 | Array B element 50  - Elements do not match. Increment Counter
Array A element: 40 | Array B element 40  - 40 is within B:[35, 45, 55, 60, 50, 40]

Array A element: 70 | Array B element 35  - Elements do not match. Increment Counter
Array A element: 70 | Array B element 45  - Elements do not match. Increment Counter
Array A element: 70 | Array B element 55  - Elements do not match. Increment Counter
Array A element: 70 | Array B element 60  - Elements do not match. Increment Counter
Array A element: 70 | Array B element 50  - Elements do not match. Increment Counter
Array A element: 70 | Array B element 40  - Elements do not match. Increment Counter
70 is not within B:[35, 45, 55, 60, 50, 40], adding to C | Array C = [20, 70]

Array A element: 30 | Array B element 35  - Elements do not match. Increment Counter
Array A element: 30 | Array B element 45  - Elements do not match. Increment Counter
Array A element: 30 | Array B element 55  - Elements do not match. Increment Counter
Array A element: 30 | Array B element 60  - Elements do not match. Increment Counter
Array A element: 30 | Array B element 50  - Elements do not match. Increment Counter
Array A element: 30 | Array B element 40  - Elements do not match. Increment Counter
30 is not within B:[35, 45, 55, 60, 50, 40], adding to C | Array C = [20, 70, 30]

Array A element: 10 | Array B element 35  - Elements do not match. Increment Counter
Array A element: 10 | Array B element 45  - Elements do not match. Increment Counter
Array A element: 10 | Array B element 55  - Elements do not match. Increment Counter
Array A element: 10 | Array B element 60  - Elements do not match. Increment Counter
Array A element: 10 | Array B element 50  - Elements do not match. Increment Counter
Array A element: 10 | Array B element 40  - Elements do not match. Increment Counter
10 is not within B:[35, 45, 55, 60, 50, 40], adding to C | Array C = [20, 70, 30, 10]

Array A element: 80 | Array B element 35  - Elements do not match. Increment Counter
Array A element: 80 | Array B element 45  - Elements do not match. Increment Counter
Array A element: 80 | Array B element 55  - Elements do not match. Increment Counter
Array A element: 80 | Array B element 60  - Elements do not match. Increment Counter
Array A element: 80 | Array B element 50  - Elements do not match. Increment Counter
Array A element: 80 | Array B element 40  - Elements do not match. Increment Counter
80 is not within B:[35, 45, 55, 60, 50, 40], adding to C | Array C = [20, 70, 30, 10, 80]

Array A element: 50 | Array B element 35  - Elements do not match. Increment Counter
Array A element: 50 | Array B element 45  - Elements do not match. Increment Counter
Array A element: 50 | Array B element 55  - Elements do not match. Increment Counter
Array A element: 50 | Array B element 60  - Elements do not match. Increment Counter
Array A element: 50 | Array B element 50  - 50 is within B:[35, 45, 55, 60, 50, 40]

Array A element: 90 | Array B element 35  - Elements do not match. Increment Counter
Array A element: 90 | Array B element 45  - Elements do not match. Increment Counter
Array A element: 90 | Array B element 55  - Elements do not match. Increment Counter
Array A element: 90 | Array B element 60  - Elements do not match. Increment Counter
Array A element: 90 | Array B element 50  - Elements do not match. Increment Counter
Array A element: 90 | Array B element 40  - Elements do not match. Increment Counter
90 is not within B:[35, 45, 55, 60, 50, 40], adding to C | Array C = [20, 70, 30, 10, 80, 90]

Array A element: 60 | Array B element 35  - Elements do not match. Increment Counter
Array A element: 60 | Array B element 45  - Elements do not match. Increment Counter
Array A element: 60 | Array B element 55  - Elements do not match. Increment Counter
Array A element: 60 | Array B element 60  - 60 is within B:[35, 45, 55, 60, 50, 40]

Final Array C: [20, 70, 30, 10, 80, 90]
```

c) Perform asymptotic analysis to determine the maximum number of comparisons of array elements that are needed. What is the Big-Oh class for this algorithm in terms of m and n?

```
1   def arrayDiff(A,B):
2       # Array to hold set difference
3       C = []
4
5       # For each element in A, check it against each element in B
6       # If the number of differences = the length of B, it is not in B
7       for a in A:
8           count = 0
9           for b in B:
10              if a != b:
11                  count += 1
12              else:
13                  break # break early if there is one match
14          if count == len(B):
15              C.append(a)
16
17      return C
```

| Line | Cost | Count |
|---|---|---|
| 3 | C1 | 1 |
| 7 - 8 | C2 | n |
| 9 - 11 | C4 | n*m |
| 12 – 13 | C5 | 0 |
| 14 – 15 | C6 | n |
| 17 | C7 | 1 |

$T(n) = C_1 + nC_2 + n*mC_4 + C_6n + C_7$

$T(n) = C_1 + C_7 + n( C_2 + C_6) + n*mC_4$

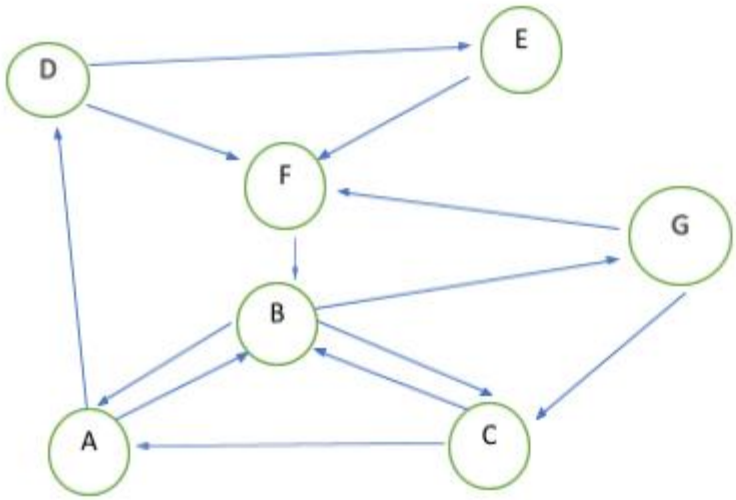
$T(n) = C_8 + nC_9 + n*mC_4$

$T(n) <= n*mC_8 + n*mC_9 + n*mC_4$

$T(n) <= n*m(C_{10})$

O(nm)

#### 4) Recursion - Breadth First Search and Depth First Search




a) Represent this graph using adjacency lists. Arrange the neighbors of each vertex in alphabetical order.

- (A,B,1), (A,D,1)
- (B,A,1), (B,C,1), (B,G,1)
- (C,A,1), (C,B,1)
- (D,E,1), (D,F,1)
- (E,F,1)
- (F,B,1)
- (G,C,1), (G,F,1)

b) Show the steps of a breadth first search with the graph using the technique given in the class notes. Use the adjacency lists representation that you created. Start at vertex A. As part of your answer, produce a graph that has the vertices numbered according to the order in which they were processed/visited.

```
Vertex A enqueued, Queue: ['A']
Vertex A visited,  Visited: ['A']
Vertex B enqueued, Queue: ['A', 'B']
Vertex B visited,  Visited: ['A', 'B']
Vertex D enqueued, Queue: ['A', 'B', 'D']
Vertex D visited,  Visited: ['A', 'B', 'D']
Vertex A dequeued, Queue: ['B', 'D']
Vertex C enqueued, Queue: ['B', 'D', 'C']
Vertex C visited,  Visited: ['A', 'B', 'D', 'C']
Vertex G enqueued, Queue: ['B', 'D', 'C', 'G']
Vertex G visited,  Visited: ['A', 'B', 'D', 'C', 'G']
Vertex B dequeued, Queue: ['D', 'C', 'G']
Vertex E enqueued, Queue: ['D', 'C', 'G', 'E']
Vertex E visited,  Visited: ['A', 'B', 'D', 'C', 'G', 'E']
Vertex F enqueued, Queue: ['D', 'C', 'G', 'E', 'F']
Vertex F visited,  Visited: ['A', 'B', 'D', 'C', 'G', 'E', 'F']
Vertex D dequeued, Queue: ['C', 'G', 'E', 'F']
Vertex C dequeued, Queue: ['G', 'E', 'F']
Vertex G dequeued, Queue: ['E', 'F']
Vertex E dequeued, Queue: ['F']
Vertex F dequeued, Queue: []
Visited Order:
1. A
2. B
3. D
4. C
5. G
6. E
7. F
```

c) Show the steps of a depth first search with the graph using the technique given in the class notes. Use the adjacency lists representation that you created. Start at vertex A. As part of your answer, produce a graph that has the vertices numbered according to the order in which they were processed/visited.

```
DFS called for vertex A
Vertex A is visited and received the stamp 0 | Visited: ['A']
DFS called for vertex B
Vertex B is visited and received the stamp 1 | Visited: ['A', 'B']
DFS called for vertex C
Vertex C is visited and received the stamp 2 | Visited: ['A', 'B', 'C']
DFS called for vertex G
Vertex G is visited and received the stamp 3 | Visited: ['A', 'B', 'C', 'G']
DFS called for vertex F
Vertex F is visited and received the stamp 4 | Visited: ['A', 'B', 'C', 'G', 'F']
DFS called for vertex D
Vertex D is visited and received the stamp 5 | Visited: ['A', 'B', 'C', 'G', 'F', 'D']
DFS called for vertex E
Vertex E is visited and received the stamp 6 | Visited: ['A', 'B', 'C', 'G', 'F', 'D', 'E']
Visited Order:
1. A
2. B
3. C
4. G
5. F
6. D
7. E
```

**5) Recursion - Master Method**
Use the master method to determine the Big-Oh class for an algorithm whose worst-case performance is given by each of these recurrence relations.

a) $T(n) = 4T(n/2) + n^3$

**Variables**
**$a = 4$**
**$b = 2$**
**$f(n) = n^3$**

**Compare $f(n)$ to $n^d$**
**$n^{\log_b a} = n^{\log_2 4} = n^2$**

**$n^3 > n^2$**

**Big-Oh Complexity**
**$T(n) = O(f(n)) = O(n^3)$**

b) $T(n) = 4T(n/2) + n^2$

**Variables**
**$a = 4$**
**$b = 2$**
**$f(n) = n^2$**

**Compare $f(n)$ to $n^d$**
**$n^{\log_b a} = n^{\log_2 4} = n^2$**

**$n^2 = n^2$**

**Big-Oh Complexity**
**$T(n) = O(n^{\log_b a}\log(n)) = O(n^2 \log n)$**

c) $4T(n/2) + n$

**<u>Variables</u>**

**$a = 4$**

**$b = 2$**

**$f(n) = n$**

**<u>Compare f(n) to $n^d$</u>**

**$n^{\log_b a} = n^{\log_2 4} = n^2$**

**$n^1 < n^2$**

**<u>Big-Oh Complexity</u>**

**$T(n) = O(n^{\log_b a}) = O(n^2)$**

**6) Decrease-and-Conquer Algorithm – Maximum Element in Array**

a) Write a recursive decrease-and-conquer algorithm to calculate the maximum element in a non-empty array of real numbers. Your algorithm should work by comparing the last element in the array with the maximum of the "remaining front end" of the array.

For example, to find the largest element in the array [5, 13, 9, 10] your algorithm should call itself to find the maximum of [5, 13, 9] and return either 10 or the result of the recursive call, whichever is larger.

- Do not use Python's built-in max() function.
- Do not rearrange the elements of the array by sorting or partially sorting them.
- Do not use any loops.

You can assume that the array has at least one element in it.

Your function call should call should be

**Maximum(A, right)**

where the two input parameters are the array and right index. With these input parameters, the function should return the maximum array element from A[0] to A[right]. Return the value of the array element, not the index where it occurs in the array.

```python
def Maximum(A, right):
    if right == 0:
        print(f"Base Case Reached. Return {A[0]}")
        return A[0]
    else:
        max_remaining = Maximum(A, right-1)
        print(f"Compare {max_remaining} and {A[right]}")
        max_so_far = max_remaining if max_remaining > A[right] else A[right]
        print(f"Return {max_so_far}")
        return max_so_far
```

**b) Trace your algorithm with A = [17, 62, 49, 73, 26, 51]**

```
Base Case Reached. Return 17
Compare 17 and 62
Return 62
Compare 62 and 49
Return 62
Compare 62 and 73
Return 73
Compare 73 and 26
Return 73
Compare 73 and 51
Return 73
Maximum Element: 73
```

**c) Write a recurrence relation for the number of comparisons of array elements that are performed for a problem of size n. Then perform asymptotic analysis to determine the Big-Oh class for this algorithm**

$T(n) = 1 + T(n-1)$ and $T(1) = 0$

## Back-Substitution

**Substitute n-1**

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

$T(n-1) = 1 + T(n-2)$

$T(n) = 1 + 1 + T(n-2)$

**Substitute n-2**

$T(n-2) = 1 + T(n-2-1)$

$T(n-2) = 1 + T(n-3)$

$T(n) = 1 + 1 + 1 + T(n-3)$

**Pattern:**

$T(n) = k + T(n-k)$

$n – k = 0$

$n = k$

**Solve:**

$T(n) = n + T(0)$

$T(n) = n$

**Big Oh Complexity**

**O(n)**

## 7) Divide-and-Conquer Algorithms – Mergesort and Quicksort

**a) For each of these two sorting algorithms, what is its Big-Oh class in the worst case?**

- Mergesort = O(nlogn)
- Quicksort = $O(n^2)$

**b) For each of these two sorting algorithms, what is its Big-Oh class in the average case?**

- Mergesort = O(nlogn)
- Quicksort = O(nlogn)

**c) Trace the mergesort algorithm for the following array of values. A = [127, 48, 62, 51, 198, 17, 52, 209] Rather than keep track of the values of individual variables, follow the graph-like that was used in the slides to trace the Mergesort algorithm.**

```
Split Array: Input Array = [127, 48, 62, 51, 198, 17, 52, 209] -> Output Arrays = [127, 48, 62, 51] and [198, 17, 52, 209]
Split Array: Input Array = [127, 48, 62, 51] -> Output Arrays = [127, 48] and [62, 51]
Split Array: Input Array = [127, 48] -> Output Arrays = [127] and [48]
Base Case Reached: 127 returned
Base Case Reached: 48 returned
Merge Arrays: Input Arrays = [127] and [48] -> Output Array = [48, 127]
Split Array: Input Array = [62, 51] -> Output Arrays = [62] and [51]
Base Case Reached: 62 returned
Base Case Reached: 51 returned
Merge Arrays: Input Arrays = [62] and [51] -> Output Array = [51, 62]
Merge Arrays: Input Arrays = [48, 127] and [51, 62] -> Output Array = [48, 51, 62, 127]
Split Array: Input Array = [198, 17, 52, 209] -> Output Arrays = [198, 17] and [52, 209]
Split Array: Input Array = [198, 17] -> Output Arrays = [198] and [17]
Base Case Reached: 198 returned
Base Case Reached: 17 returned
Merge Arrays: Input Arrays = [198] and [17] -> Output Array = [17, 198]
Split Array: Input Array = [52, 209] -> Output Arrays = [52] and [209]
Base Case Reached: 52 returned
Base Case Reached: 209 returned
Merge Arrays: Input Arrays = [52] and [209] -> Output Array = [52, 209]
Merge Arrays: Input Arrays = [17, 198] and [52, 209] -> Output Array = [17, 52, 198, 209]
Merge Arrays: Input Arrays = [48, 51, 62, 127] and [17, 52, 198, 209] -> Output Array = [17, 48, 51, 52, 62, 127, 198, 209]
```

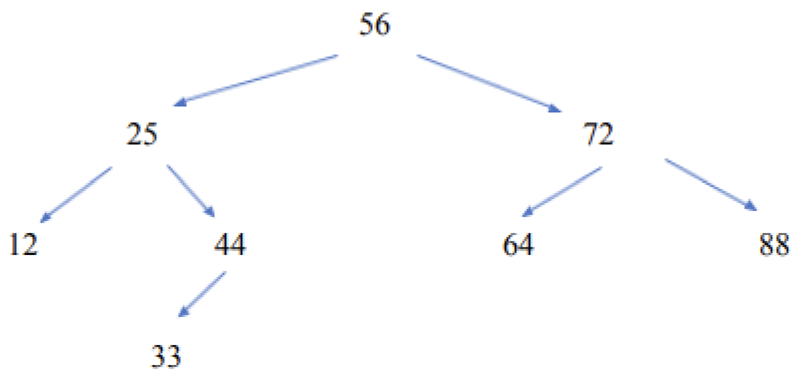
**d) Trace the Quicksort algorithm for the same array of values. A = [127, 48, 62, 51, 198, 17, 52, 209] Indicate the pivots in red as was done in the class notes.**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 127 | 48 | 62 | 51 | 198 | 17 | 52 | 209 |
| 48 | 51 | 17 | 52 | 198 | 62 | 127 | |
| 17 | 51 | 48 | | 62 | 127 | 198 | |
| | 48 | 51 | | 62 | | 198 | |
| | | 51 | | | | | |

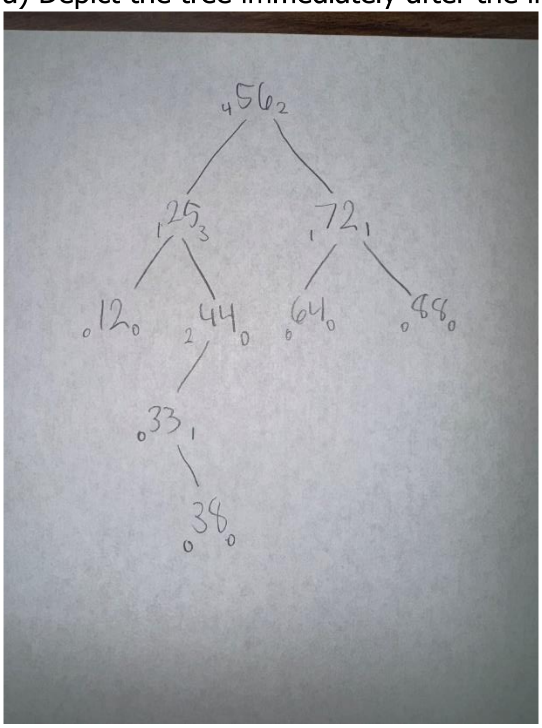
Sorted Array:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 17 | 48 | 51 | 52 | 62 | 127 | 198 | 209 |

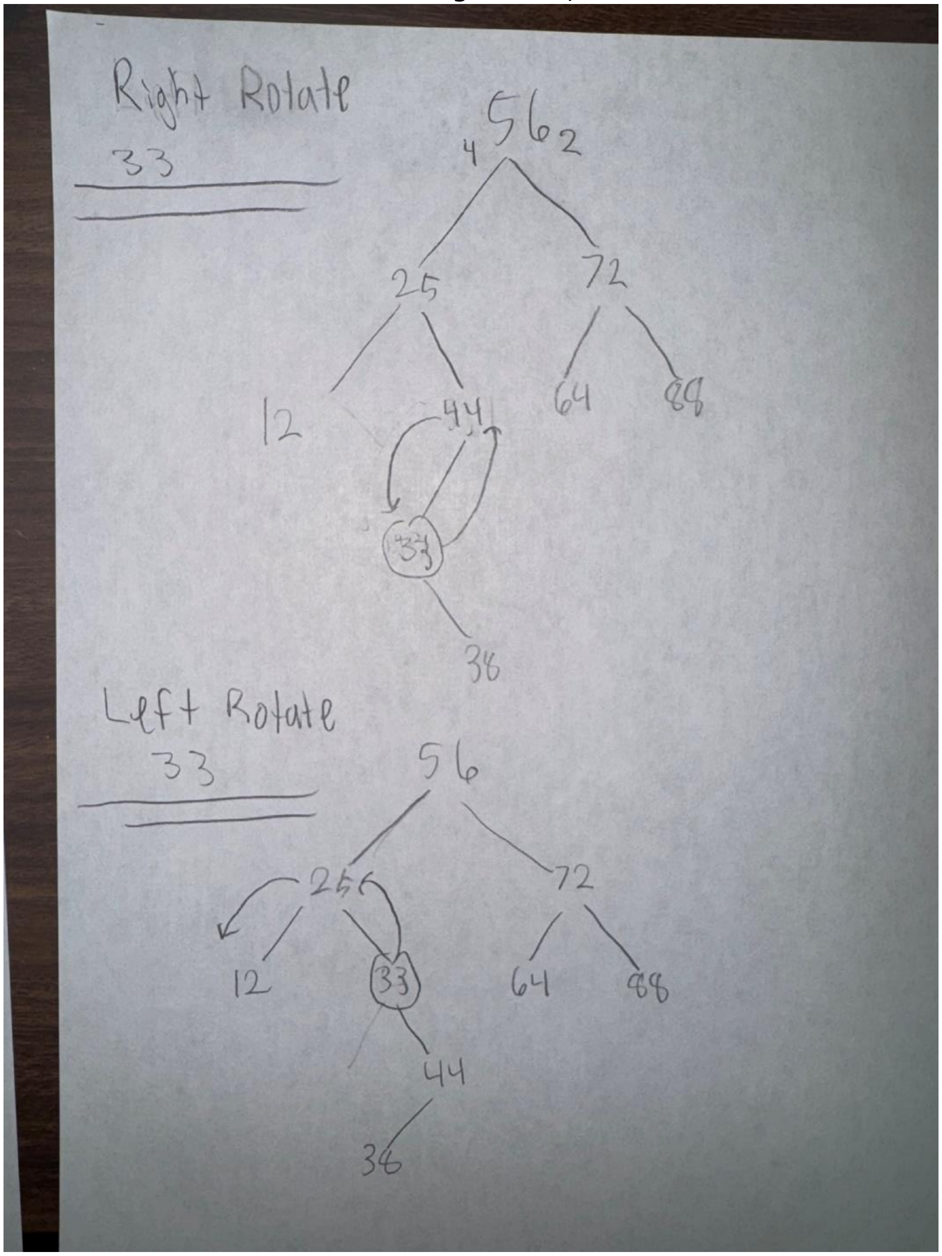
8) Transform-and-Conquer Algorithms – AVL Trees Considering the AVL Tree below, what happens if the value 38 is inserted in this tree?




a) Depict the tree immediately after the insertion of 38 (without balancing);



b) Describe what possible rotations, if necessary, need to be taken to balance the tree, indication the rotation kind and target node;



c) Depict the tree after the balancing operations (rotations).