1. Create Swap method

A screen shot of a computer program

Description automatically generated

**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) = c1n + c2n. When you simplify this equation, it would result in T(n) = n(c3), 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(n2). 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(n2).

**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(n2) complexity each. What is the complexity of C?**

* The complexity would be O(n2). 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(n2) Big-Oh complexity.

**d) A given algorithm D is an iterative one that for a problem of size n executes O(n2 ) calls of a function that has complexity O(log n). What is the complexity of D?**

* The complexity would be O(n2\*logn). The loop calls the function O(n2) 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].

A screenshot of a computer program

Description automatically generated

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

A white sheet with blue text

Description automatically generated

A white paper with blue text

Description automatically generated

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?

A screenshot of a computer program

Description automatically generated

|  |  |  |
| --- | --- | --- |
| 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) = C1 + nC2 + n\*mC4 + C6n + C7

T(n) = C1 + C7 + n( C2 + C6) + n\*mC4

T(n) = C8 + nC9 + n\*mC4

T(n) <= n\*mC8 + n\*mC9 + n\*mC4

T(n) <= n\*m(C10)

O(nm)

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

A diagram of a network

Description automatically generated

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

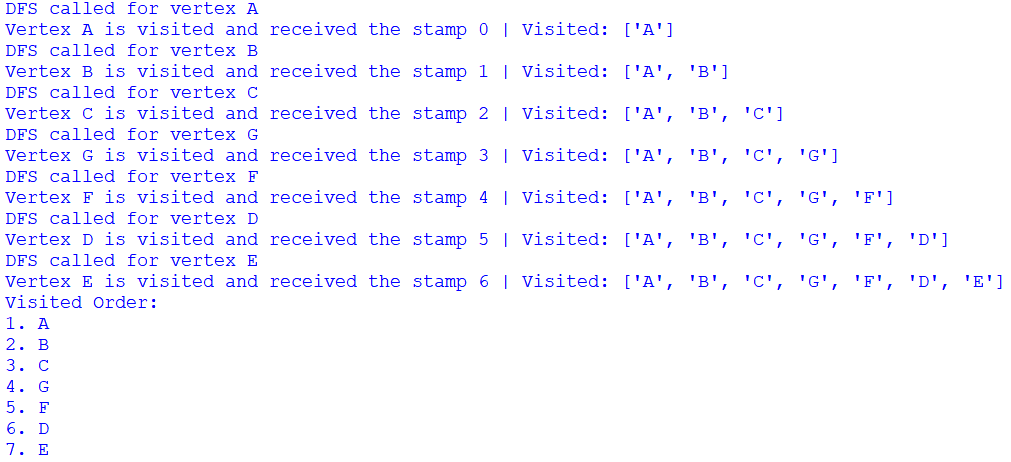
* (A,B,1), (A,D,1)
* (B,A,1), (B,C,1), (B,G,1)
* (C,A,1), (C,B,1)
* (D,E,1), (D,F,1)
* (E,F,1)
* (F,B,1)
* (G,C,1), (G,F,1)

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.

A screenshot of a computer program

Description automatically generated

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.



**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.

1. T(n) = 4T(n/2) + n3

**Variables**

**a = 4**

**b = 2**

**f(n) = n3**

**Compare f(n) to nd**

**nlogba = nlog24 = n2**

**n3 > n2**

**Big-Oh Complexity**

**T(n) = O(f(n)) = O(n3)**

1. T(n) = 4T(n/2) + n2

**Variables**

**a = 4**

**b = 2**

**f(n) = n2**

**Compare f(n) to nd**

**nlogba = nlog24 = n2**

**n2 = n2**

**Big-Oh Complexity**

**T(n) = O(nlogbalog(n)) = O(n2logn)**

1. 4T(n/2) + n

**Variables**

**a = 4**

**b = 2**

**f(n) = n**

**Compare f(n) to nd**

**nlogba = nlog24 = n2**

**n1 < n2**

**Big-Oh Complexity**

**T(n) = O(nlogba) = O(n2)**

**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.

A computer screen shot of text

Description automatically generated

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

A screenshot of a computer

Description automatically generated

1. **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(n2)

**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.**

**A white text with blue text

Description automatically generated**

**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 diagram of numbers and arrows

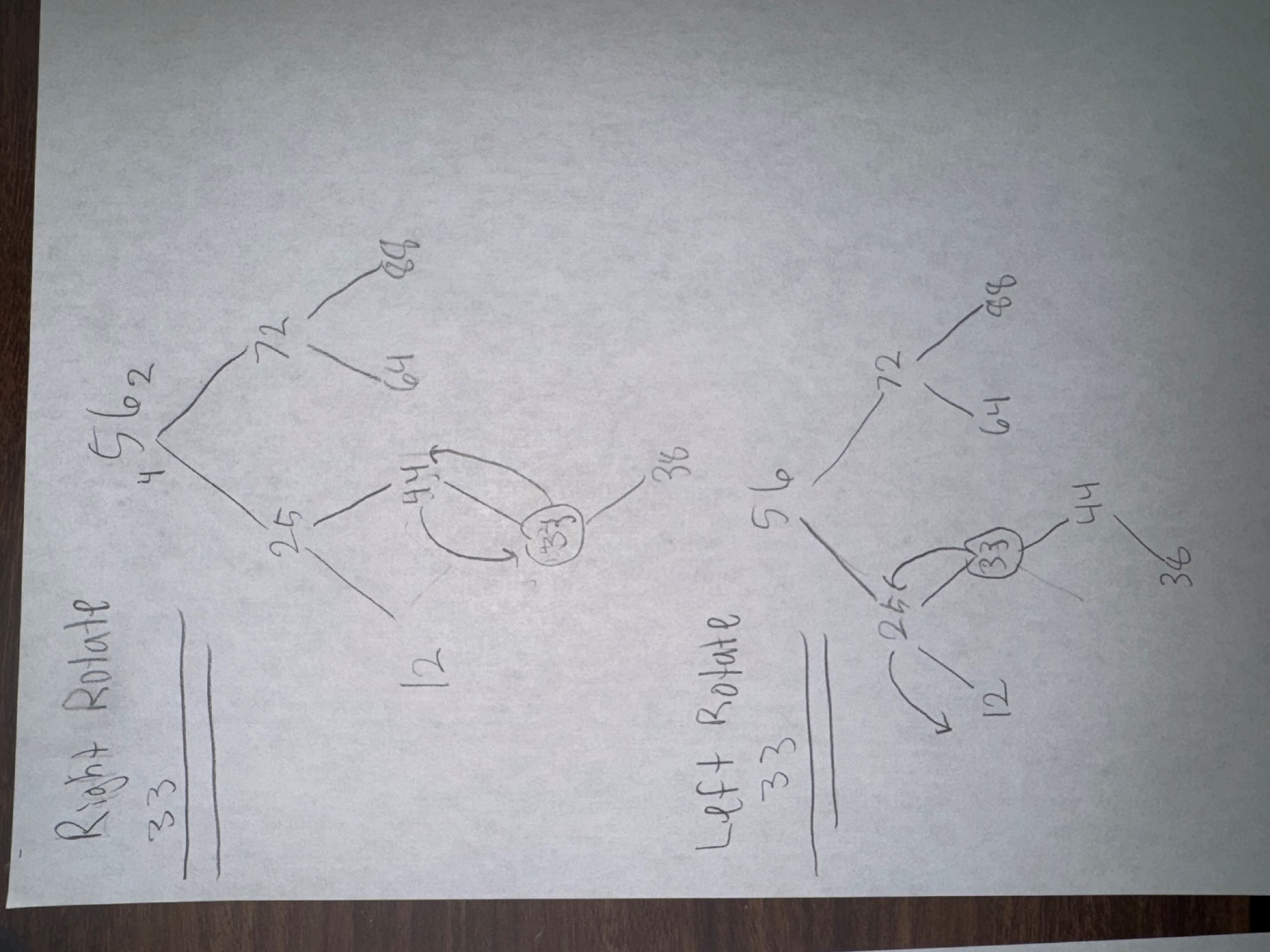
Description automatically generated

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

A graph on a white paper

Description automatically generated

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).

