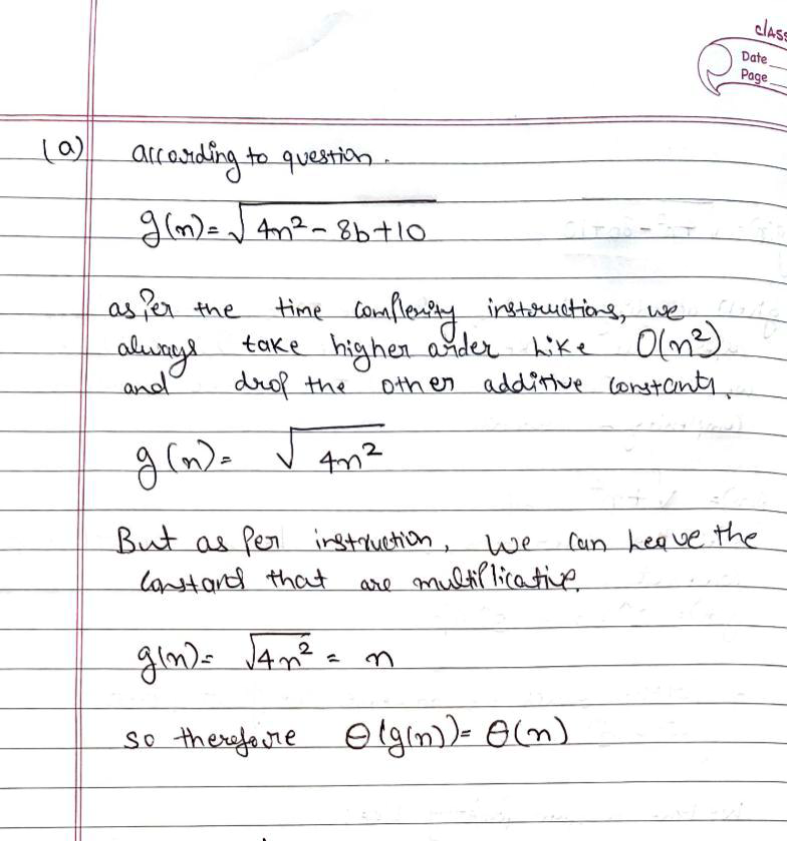
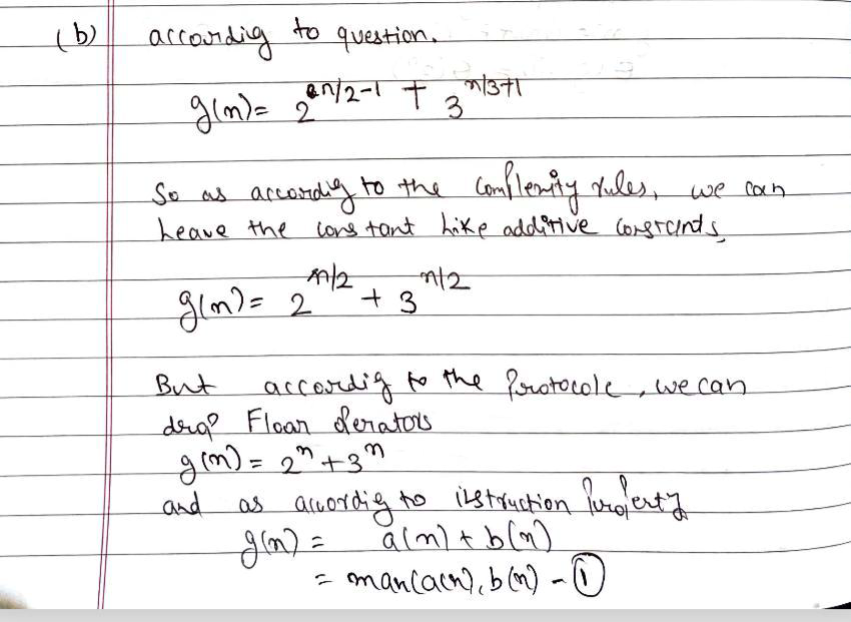
Assignment 1

1. Indicate the class Θ

For each of the following functions, indicate the class Θ(g(n)) the function belongs to. Apply all possible rules to simply g(n) as much as possible. Prove your assertions either using definitions or the limit theorem.

* 1.   
     
  2.   
       
     

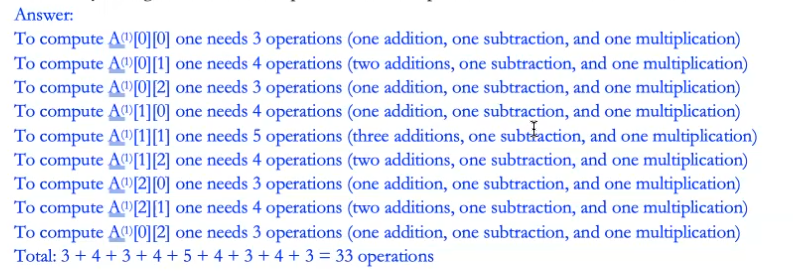
1. Parallel matrix update of size 3x3  
   Given a matrix A of size 3x3, each cell (i.e. an element of the matrix) will be updated based on the values of (existing) immediate neighboring cells. This update must be done in parallel, this means A[0,0] will be updated in parallel with A[0,1], etc.. For a cell A[i,j], the update will be of the form:

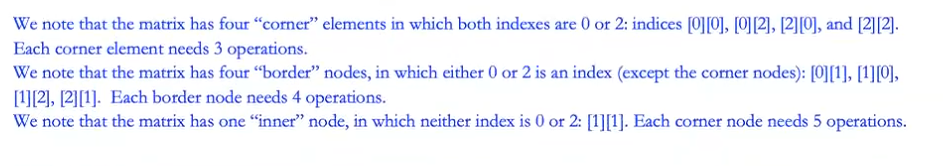
A[i][j] = A[i-1][j] + A[i+1][j] + A[i][j-1] + A[i][j+1] - 4\*A[i][j]

Some of the terms may be missing if the element is on the border of the matrix: i=0, j=0, i=2, j=2.

For example A[0][0] = A[1][0] + A[0][1] - 4\*A[0][0].

In other terms, if A(0) is the initial matrix at time 0, then A(1) will be computed entirely based on the cells of A(0) at time 1, A(2) will be computed entirely based on the cells of A(1) at time 2, and so on. To compute A(1)[0][0] one needs 3 operations (one addition, one subtraction, and one multiplication). You need to state the number of simple mathematical operations needed to compute all the cells at time. This value is obtained by adding the number of simple mathematical operations for each cell in the matrix.





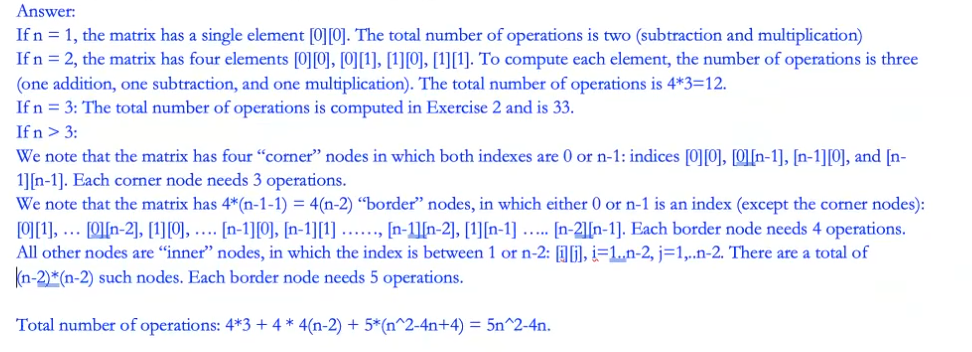
1. Parallel matrix update of size 3x3  
   Given a matrix A of size n x n, each cell (i.e. an element of the matrix) will be updated based on the values of (existing) immediate neighboring cells. This update must be done in parallel, this means A[0,0] will be updated in parallel with A[0,1], etc.. For a cell A[i,j], the update will be of the form:

A[i][j] = A[i-1][j] + A[i+1][j] + A[i][j-1] + A[i][j+1] - 4\*A[i][j]

Some of the terms may be missing if the element is on the border of the matrix: i=0, j=0, i=n-1, j=n-1.

For example A[0][0] = A[1][0] + A[0][1] - 4\*A[0][0].

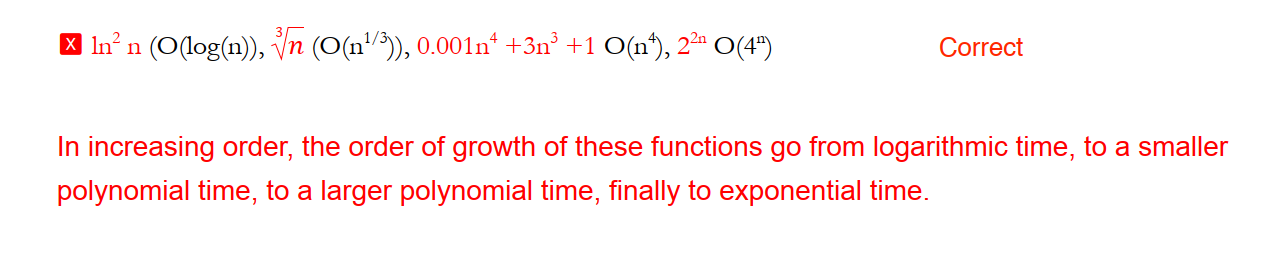
In other terms, if A(0) is the initial matrix at time 0, then A(1) will be computed entirely based on the cells of A(0) at time 1, A(2) will be computed entirely based on the cells of A(1) at time 2, and so on. To compute A(1)[0][0] one needs 3 operations (one addition, one subtraction, and one multiplication). You need to state the number of simple mathematical operations needed to compute all the cells at time 1. This value is obtained by adding the number of simple mathematical operations for each cell in the matrix.



# Assignment 2

1. List the following functions according to their order of growth from the lowest to the highest:

22n, 0.001n4 +3n3 +1, ln2 n, 3 𝑛

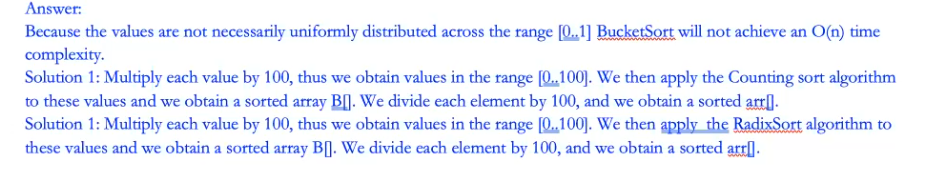


1. Consider the problem of sorting an array ofn floating point numbers which are in range from 0 to 1 and have at most two digits after the decimal point. The elements in the array are not necessarily uniformly distributed across the range [0..1]. Write a linear time O(n) algorithm to sort these n values.

Example:

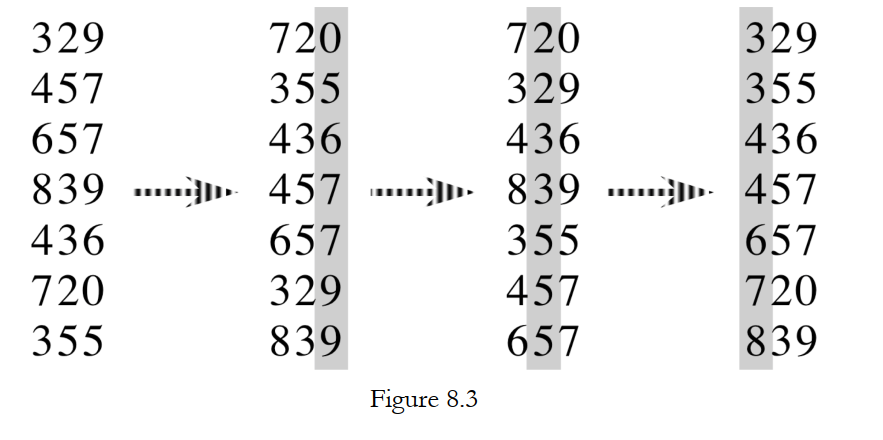
Input : arr[] = { 0.89, 0.7, 0.56, 0.5, 0.65, 0.6, 0.12, 0.34, 0, 0.34, 0.4 }

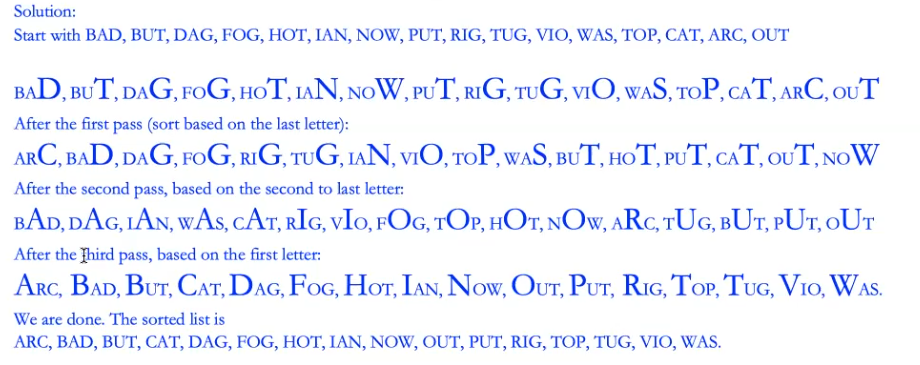
Output : 0 0.12 0.34 0.34 0.4 0.5 0.56 0.6 0.65 0.7 0.89



1. (Forming the largest number,)

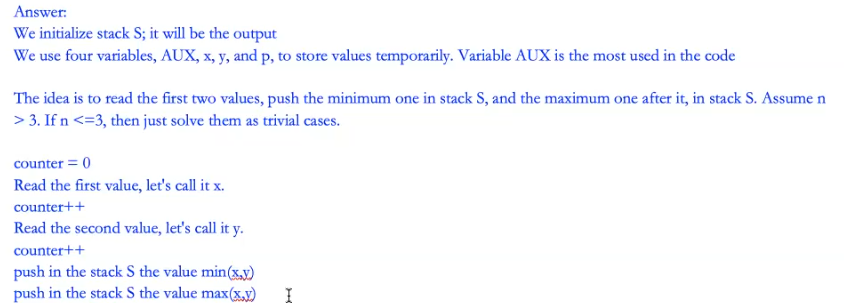
Using Figure 8.3 (below) as a model, illustrate the operation of RADIX-SORT on the following list of English words: BAD, BUT, DAG, FOG, HOT, IAN, NOW, PUT, RIG, TUG, VIO, WAS, TOP, CAT, ARC, OUT.

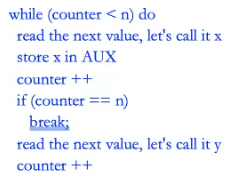


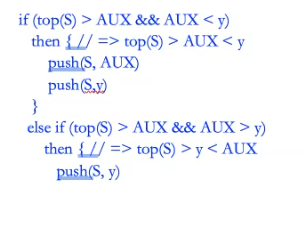


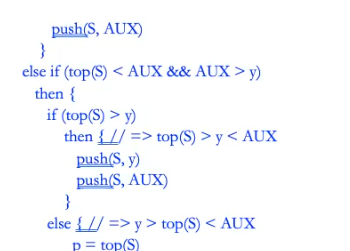
# Assignment 3

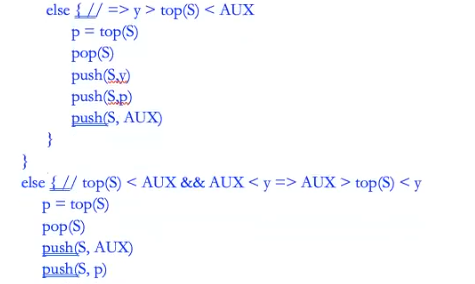
1. A wiggle sequence is a sequence where the differences between successive numbers strictly alternate between positive and negative. The first difference (if one exists) may be either positive or negative. A sequence with one element and a sequence with two non-equal elements are trivially wiggle sequences. For example, [1, 7, 4, 9, 2, 5] is a wiggle sequence because the differences (6, -3, 5, -7, 3) alternate between positive and negative. In contrast, [1, 4, 7, 2, 5] and [1, 7, 4, 5, 5] are not wiggle sequences. The first is not because its first two differences are positive, and the second is not because its last difference is zero. Design an algorithm that reads n numerical values from the user, one by one, and inserts these values as a wiggle sequence into a stack in O(n) time. You can use one additional stack or one additional queue, to store values temporarily but no other data structures. For stack and queues, only push/enqueue and pop/dequeue are allowed, and one cannot access any elements other than the top/front. Note: You can assume that no two consecutive values are the same, but one can have identical values, just not consecutive ones.

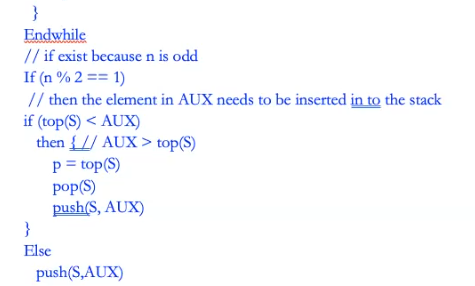












1. Given an array of positive integers, arrange them in a way that yields the largest ODD value.

Example 1: If the given numbers are {1, 34, 3, 98, 9, 76, 45, 4}, then the arrangement 998764543431 (coming from 9 98 76 45 4 34 3 1) gives the largest odd value.

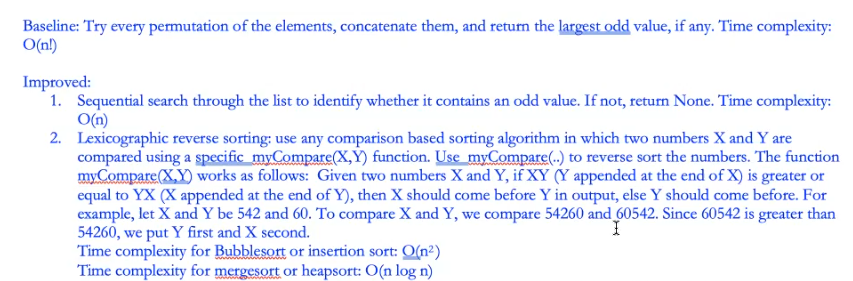
Example 2:

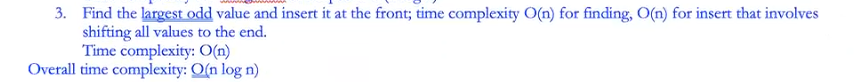
If the given numbers are {53, 516, 648, 60}, the arrangement 6486051653 gives the largest odd value.

Example 3:

If the given numbers are {54, 546, 548, 60}, then arrangement 6054854654 gives the largest odd value. The output is NONE

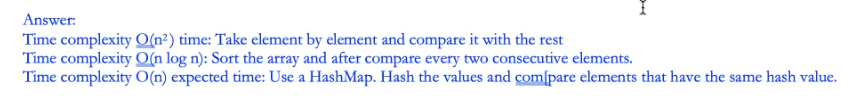
Your goal: baseline O(n!) time, improved to O(n log n).





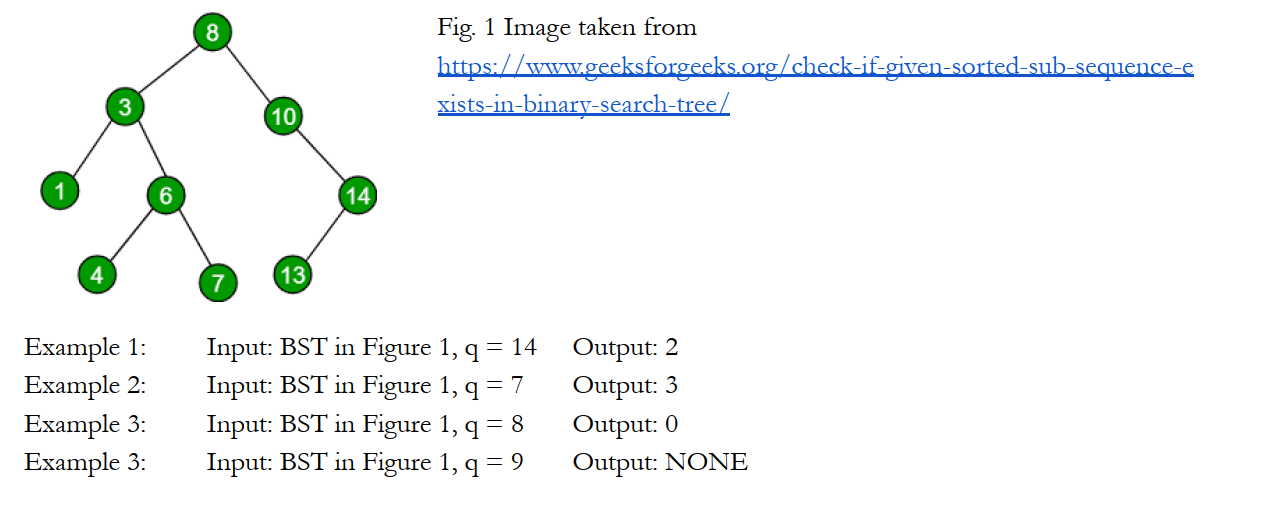
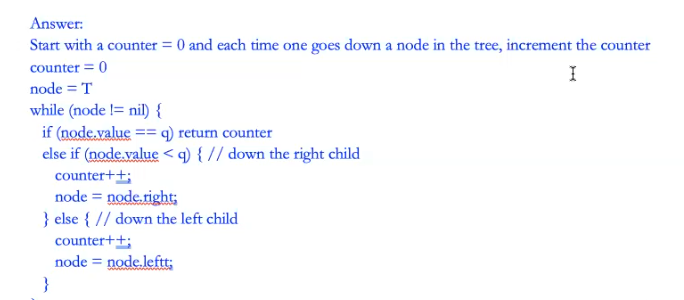
1. Design an algorithm that, given a set 𝑆of 𝑛 integers, identifies all elements that repeat.

Your goal: baseline O(n2 ) time, improved to either O(n log n) worst-case time or O(n) expected time.

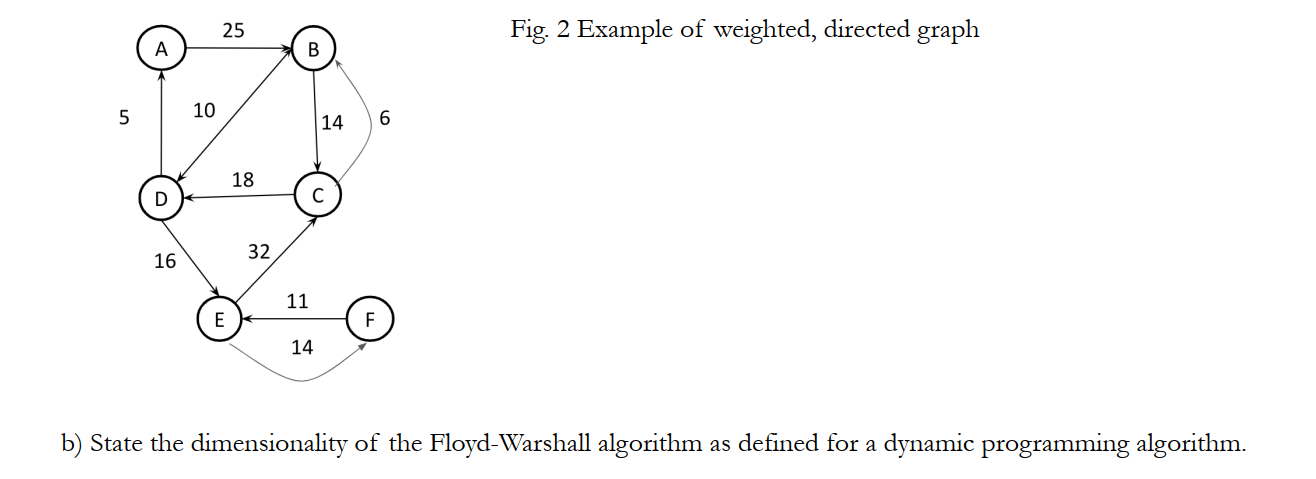


# Assignment 4

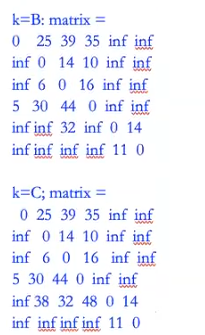
1. Given a binary search tree with n nodes (n > 3) and a query value q, design an algorithm that returns NONE if the query value is not in the tree, or the depth of the node that holds the query value. The depth of the root node is 0.

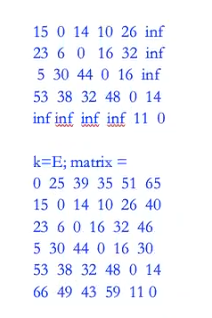
Baseline: time complexity O(n), improved to O(height) time complexity  
  


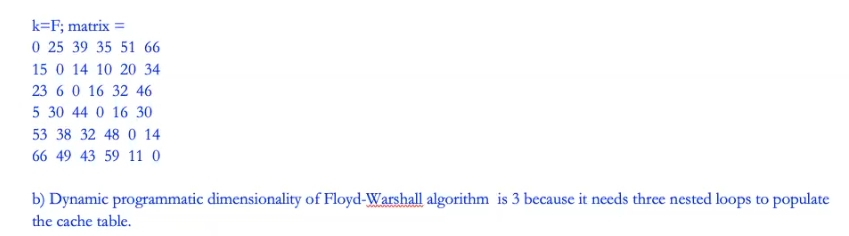


1. a) Run the Floyd-Warshall algorithm on the weighted, directed graph shown below. Show the matrix D(k) that results for each iteration of the outer loop. Show your work at every step. I will deduct points if you do not show your work and you skip steps.  
   

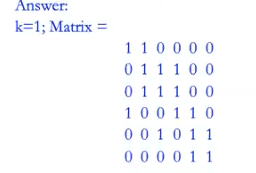


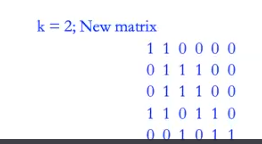


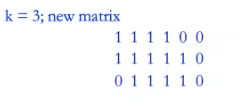


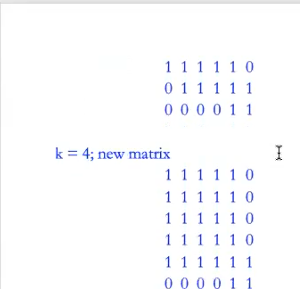


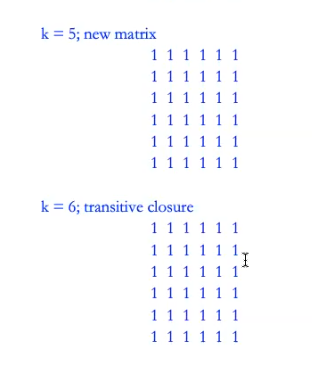
1. Compute the transitive closure of the graph above. Show your work at every step. I will deduct points if you do not show your work and you skip steps.



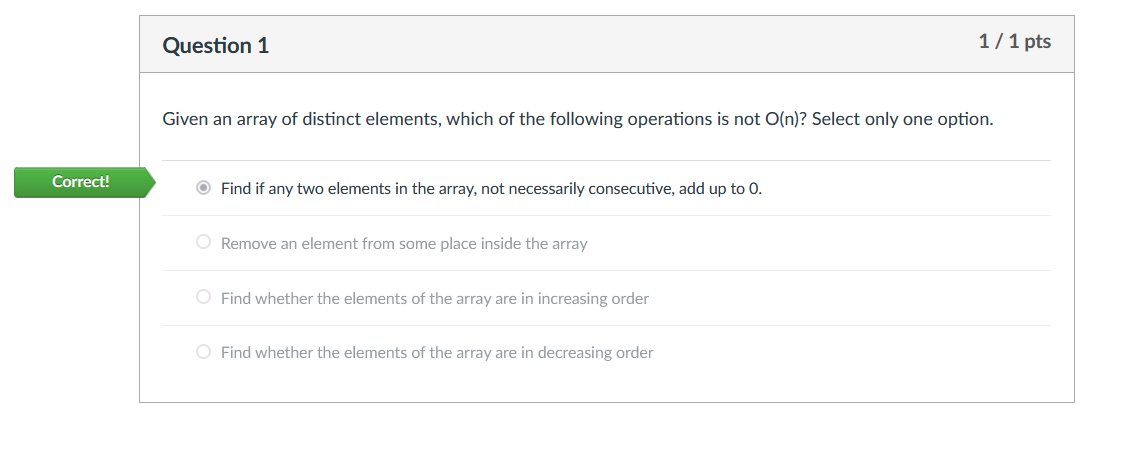
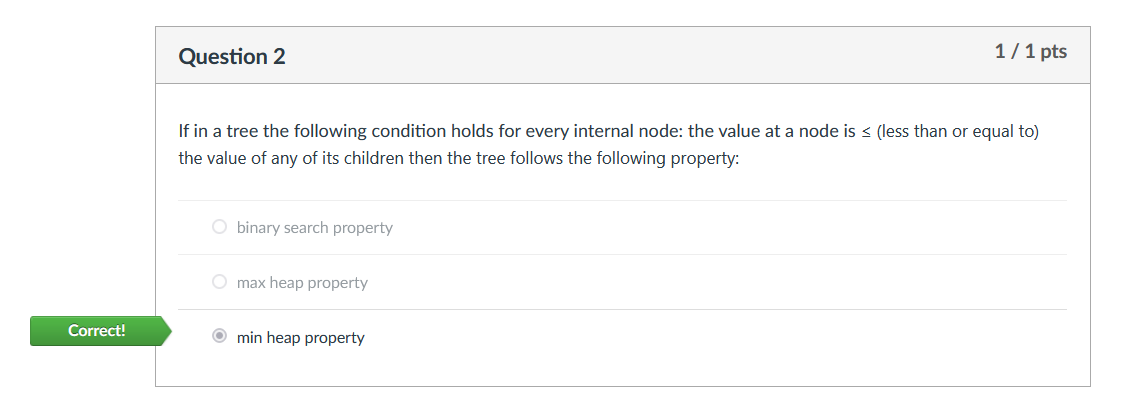
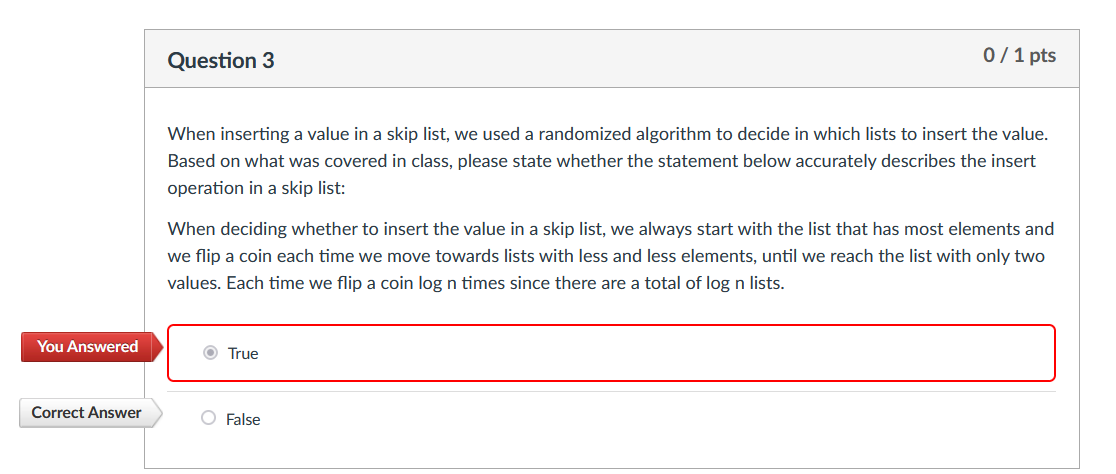
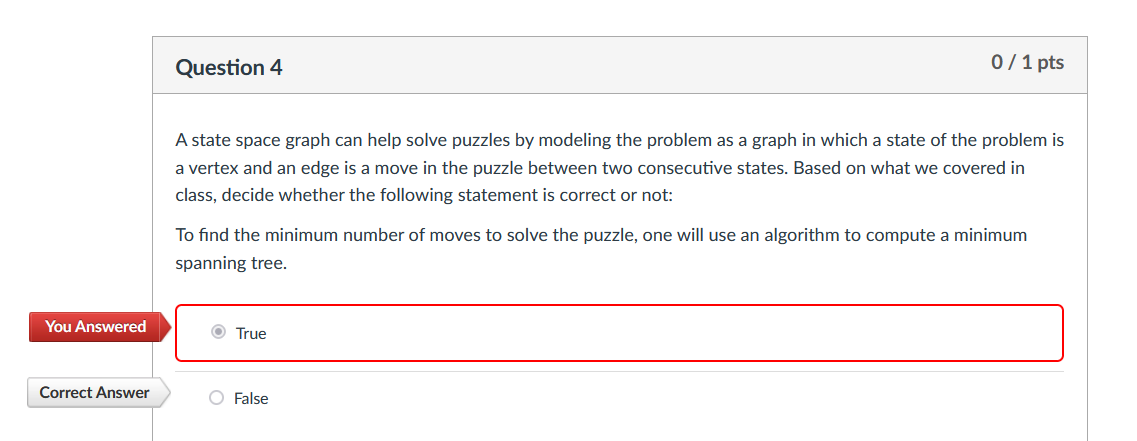
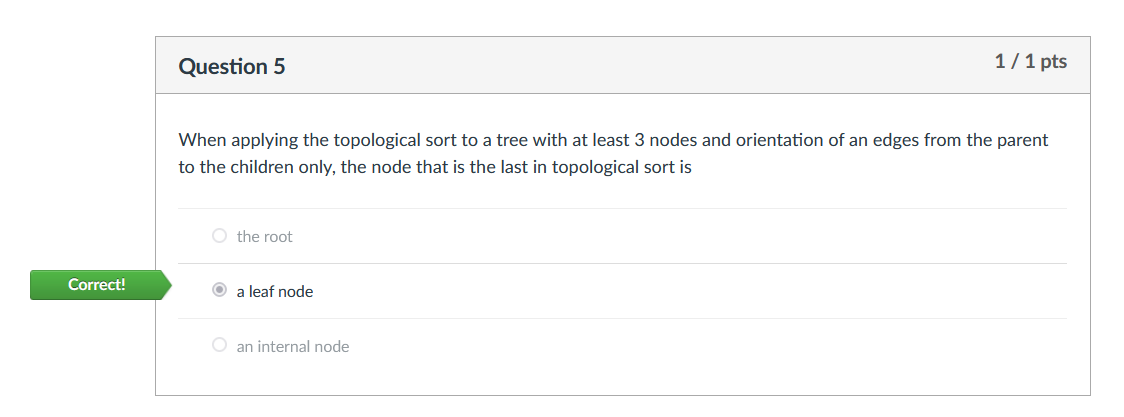
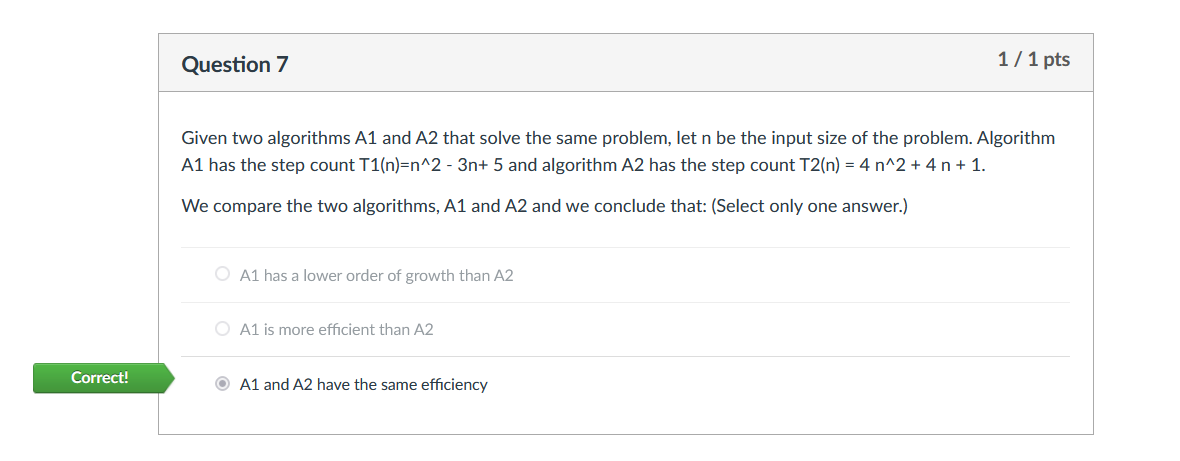
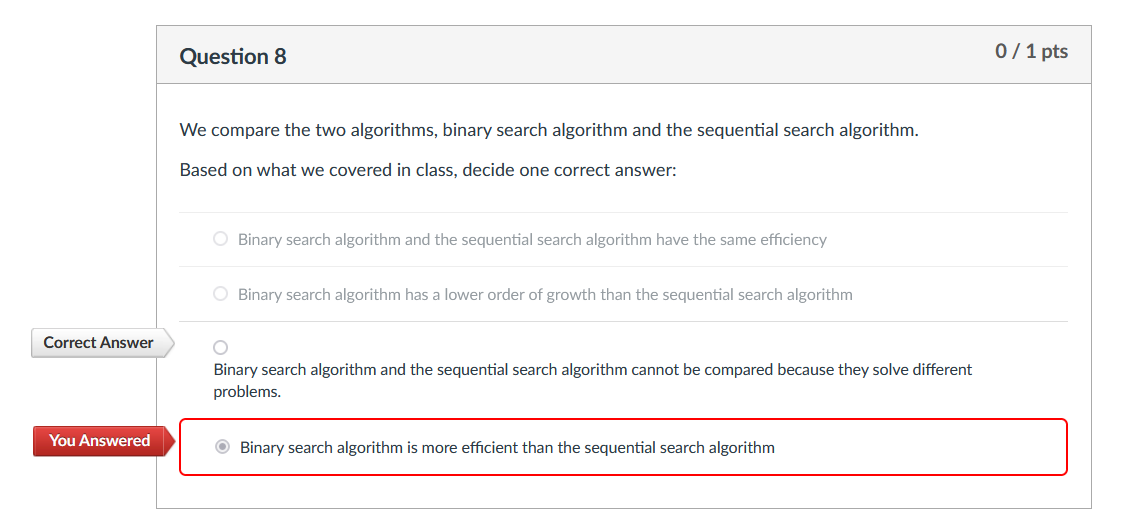








# Quiz 1

1. 
2. 
3. 
4. 
5. 
6. 
7. 
8. 
9. 