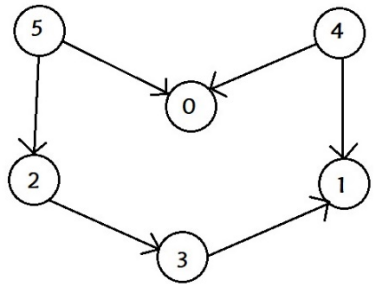
1	What are the three major variations of Decrease and Conquer design strategy? Answer: There are three major variations of decrease-and-conquer: decrease by a constant decrease by a constant factor variable size decrease In the ***decrease-by-a-constant*** variation, the size of an instance is reduced by the same constant on each iteration of the algorithm. Typically, this constant is equal to one The ***decrease-by-a-constant-factor*** technique suggests reducing a problem instance by the same constant factor on each iteration of the algorithm. In most applications, this constant factor is equal to two. Finally, in the ***variable-size-decrease*** variety of decrease-and-conquer, the size-reduction pattern varies from one iteration of an algorithm to another

2	Write the algorithm for Insertion Sort and derive its best and worst case time complexities. Explain Insertion Sort Algorithm with the following data
	12, 6, 2, 5, 11, 4, 8, 1
	
	ALGORITHM *InsertionSort*$(A[0..n-1])$
	//Sorts a given array by insertion sort
	//Input: An array $A[0..n-1]$ of n orderable elements
	//Output: Array $A[0..n-1]$ sorted in nondecreasing order
	for $i \leftarrow 1$ **to** $n-1$ **do**
	$v \leftarrow A[i]$
	$j \leftarrow i-1$
	while $j \geq 0$ **and** $A[j] > v$ **do**
	$A[j+1] \leftarrow A[j]$
	$j \leftarrow j-1$
	$A[j+1] \leftarrow v$
	
	The basic operation of the algorithm is the key comparison $A[j] > v$. (Why not $j \geq 0$? Because it is almost certainly faster than the former in an actual computer implementation. Moreover, it is not germane to the algorithm: a better implementation with a sentinel—see Problem 8 in this section's exercises—eliminates it altogether.)
	The number of key comparisons in this algorithm obviously depends on the nature of the input. In the worst case, $A[j] > v$ is executed the largest number of times, i.e., for every $j = i-1, \ldots, 0$. Since $v = A[i]$, it happens if and only if $A[j] > A[i]$ for $j = i-1, \ldots, 0$. (Note that we are using the fact that on the ith iteration of insertion sort all the elements preceding $A[i]$ are the first i elements in the input, albeit in the sorted order.) Thus, for the worst-case input, we get $A[0] > A[1]$ (for $i = 1$), $A[1] > A[2]$ (for $i = 2$), $\ldots$, $A[n-2] > A[n-1]$ (for $i = n-1$). In other words, the worst-case input is an array of strictly decreasing values. The number of key comparisons for such an input is
	$$C_{worst}(n) = \sum_{i=1}^{n-1} \sum_{j=0}^{i-1} 1 = \sum_{i=1}^{n-1} i = \frac{(n-1)n}{2} \in \Theta(n^2).$$

	In the best case, the comparison $A[j] > v$ is executed only once on every iteration of the outer loop. It happens if and only if $A[i-1] \le A[i]$ for every $i = 1, \ldots, n-1$, i.e., if the input array is already sorted in nondecreasing order. (Though it "makes sense" that the best case of an algorithm happens when the problem is already solved, it is not always the case, as you are going to see in our discussion of quicksort in Chapter 5.) Thus, for sorted arrays, the number of key comparisons is $$C_{best}(n) = \sum_{i=1}^{n-1} 1 = n - 1 \in \Theta(n).$$ 12, 6, 2, 5, 11, 4, 8, 1 6, 12 ,2 ,5, 11, 4, 8, 1 2, 6, 12, 5, 11, 4, 8, 1 2, 5, 6, 12, 11, 4, 8, 1 2, 5, 6, 11, 12, 4, 8, 1 2, 4, 5, 6, 11, 12, 8, 1 2, 4, 5, 6, 8, 11, 12, 1 1, 2, 4, 5, 6, 8, 11, 12
3	Give the worst case complexity of Insertion Sort The number of key comparisons in this algorithm obviously depends on the nature of the input. In the worst case, $A[j] > v$ is executed the largest number of times, i.e., for every $j = i-1, \ldots, 0$. Since $v = A[i]$, it happens if and only if $A[j] > A[i]$ for $j = i-1, \ldots, 0$. (Note that we are using the fact that on the ith iteration of insertion sort all the elements preceding $A[i]$ are the first i elements in the input, albeit in the sorted order.) Thus, for the worst-case input, we get $A[0] > A[1]$ (for $i = 1$), $A[1] > A[2]$ (for $i = 2$), $\ldots$, $A[n-2] > A[n-1]$ (for $i = n-1$). In other words, the worst-case input is an array of strictly decreasing values. The number of key comparisons for such an input is $$C_{worst}(n) = \sum_{i=1}^{n-1} \sum_{j=0}^{i-1} 1 = \sum_{i=1}^{n-1} i = \frac{(n-1)n}{2} \in \Theta(n^2).$$
10	Give an algorithm for Topological Sort using DFS • **Step 1**: Create a temporary stack. • **Step 2**: Recursively call topological sorting for all its adjacent vertices, then push it to the stack (when all adjacent vertices are on stack). Note this step is same as Depth First Search in a recursive way. • **Step 3**: Atlast, print contents of stack.

11



Apply Topological Sort to above graph

Ans: