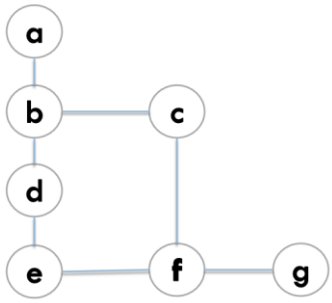
USN | | | | | | | | | | |

Department of Computer Science and Engineering
M.Tech in Computer Science and Engineering (CSE)

Course:	Advanced Data Structures and Algorithms	Course Code: 22MCE12TL	Semester: 01
28.03.2023	Duration : 90 minutes	Max Marks: 50	Staff: RS

Continuous Internal Evaluation (CIE-I)- Question Paper

Sl. No.	Answer all questions	M	* L1-L6	**CO
1a.	Solve the following recurrence relation to find the time complexity, by using iterative method: $T(n) = T(1)$ $if\ n = 1$ $T(n) = c + T\left(\frac{n}{2}\right)$ $if\ n > 1$, where, c is a constant	6	L4	CO1
1b.	(i) Suppose that we use a linked list to represent a queue and that in addition to the enqueue and dequeue functions (i.e., functions to add and remove elements from the linked list), you want to add a new operation to the queue that deletes the last element of the queue. Which linked structure do we need to use to guarantee that this operation is also executed inconstant time? Justify your answer (ii) If the characters 'D', 'C', 'B', 'A' are placed in a queue (in that order), and then removed one at a time, in what order will they be removed? (iii) Put these Big-O sets of functions into ascending order of complexity: O(n1/2), O(lg n), O(n1/3), O(n), O(1/n). (iv) If data is a circular array of CAPACITY elements, and last is an index into that array, what is the formula for the index after last	4	L2	CO1
2a.	Apply an appropriate suitable sorting algorithm to sort in ascending order, the below given numbers and also discuss the time and space complexity taken by the algorithm Input data: 455, 61, 63, 45, 67, 135, 74, 49, 15, 5	6	L4	CO2
2b.	Compare the elements of each of the two arrays given below and comment on whether the arrays have uniformly distributed data or not.	4	L3	CO2

	array1: [10, 21, 29, 41, 52] array2: [1,4,23,5,44,9,6,43]			
3a.	Apply an appropriate suitable sorting algorithm to sort in ascending order, the below given numbers and also discuss the time and space complexity taken by the algorithm. 11, 9, 21, 8,17, 19, 13, 1, 24, 12.	6	L4	CO3
3b	Suppose you have a stack ADT (i.e., an Abstract Data Type that includes operations to maintain a stack). a) Describe in words (no code) how you could implement a queue's enqueue and dequeue operations using two stacks. Also, provide the Big-O complexity figures.	4	L3	CO4
4a.	(i) Define a function length which takes a pointer to the start of the linked list and returns the number of items that are in the list. For instance, if list is the list (3, 9, 5, 6) then length(list) returns 4. (ii) Write a code to reverse a linked list using LIST ADT	6	L4	CO4
4b.	Differentiate between Stable sort and Comparision sort with an example	4	L2	CO3
5.	Perform BFS and DFS on the following graph and list the order of the nodes visted. Mention the time and space complexity for both BFS and DFS 	10	L3	CO2

****Course Outcome**

CO1: Analyze the efficiency of programs based on time complexity.
CO2: Critically think and apply appropriate design paradigm and algorithm for a specific problem.
CO3: Apply knowledge of computing and mathematics to algorithm design
CO4: Design, implement and evaluate algorithms to solve real world problems

Marks Distribution *(L1-L6)

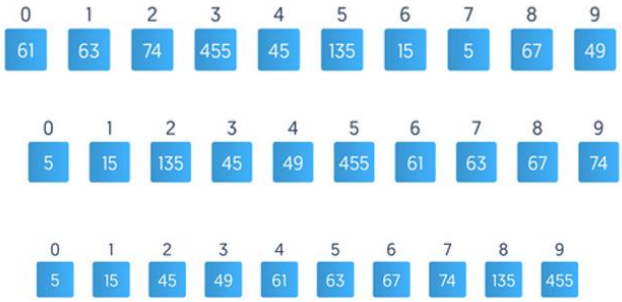
L1	L2	L3	L4	L5	L6	CO1	CO2	CO3	CO4
0	8	18	24	0	0	10	20	10	10

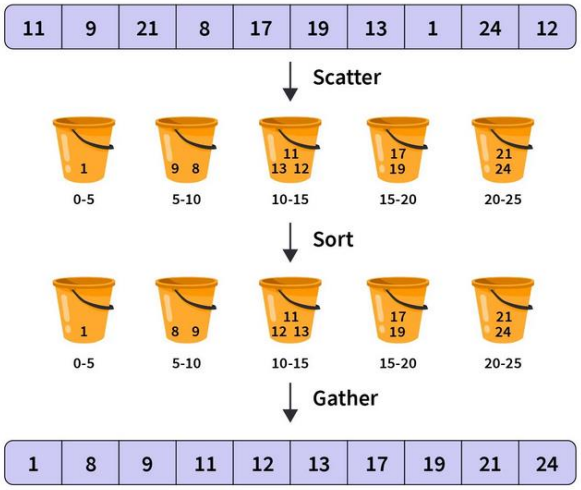
Department of Computer Science and Engineering
M.Tech in Computer Science and Engineering (CSE)

Course:	Advanced Data Structures and Algorithms	Course Code: 22MCE12TL	Semester : 01
28.03.2023	Duration : 90 minutes	Max Marks: 50	Staff : RS

Continuous Internal Evaluation (CIE-I)- Scheme and Solution

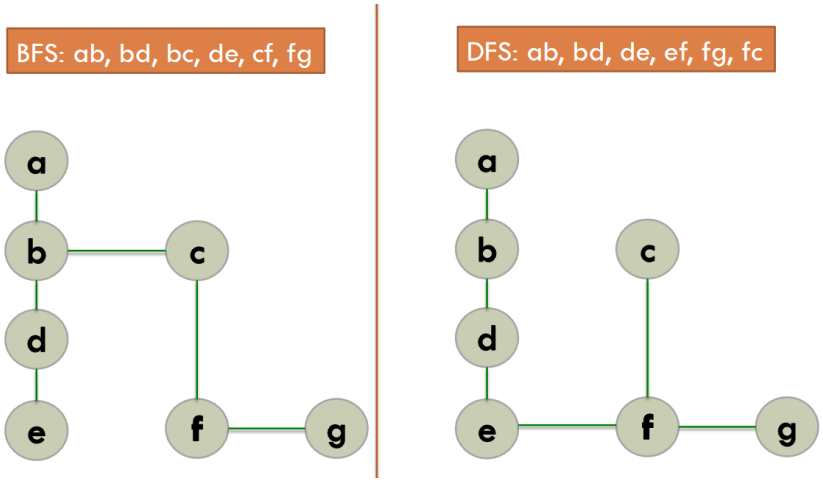
Sl. No.	Answer all questions	Marks
1a.	$T\left(\frac{n}{2}\right) = c + T\left(\frac{n}{4}\right)$ $T(n) = c + T\left(\frac{n}{2}\right)$ $T(n) = c + \underbrace{\left(c + T\left(\frac{n}{4}\right)\right)}_{\text{value of } T\left(\frac{n}{2}\right)}$ $T\left(\frac{n}{4}\right) = c + T\left(\frac{n}{8}\right)$ $T(n) = c + c + c + T\left(\frac{n}{8}\right)$ $T(n) = 3c + T\left(\frac{n}{2^3}\right)$ $T(n) = \underbrace{c + c + \ldots + c}_{k \text{ times}} + T\left(\frac{n}{2^k}\right)$ Since $n = 2^k$, $T(n) = \underbrace{c + c + \ldots + c}_{k \text{ times}} + T\left(\frac{2^k}{2^k}\right)$ $= kc + T(1)$ $n = 2^k$ $\Rightarrow \log_2(n) = log_2(2^k)$ $\Rightarrow \lg(n) = k$ $T(n) = c\lg(n) + T(1)$ $= \Theta(\lg(n))$	5

1b.	(i) We need to use a doubly-linked list, so that we can dequeue nodes from either end of the list in O(1) time. (ii) DCBA (iii) O(1/n), O(lg n), O(n1/3), O(n1/2), O(n) (iv) (last + 1) % CAPACITY	4x1=4
2a.	455, 61, 63, 45, 67, 135, 74, 49, 15, 5  Time Complexity: • **Best Case -** In the best case i.e.i.e. when the array is already sorted, we do not need to apply the algorithm instead we can check if the array is sorted in a single traversal of the array. Hence time complexity is O(n)O(n) in the best case. • **Worst Case -** In the worst case i.e.i.e. array is sorted in reverse order, we need to apply the counting Sort (an O(n)O(n) process) for kk times. Where kk is the number of digits present in the largest element present in aa. **Hence,** the overall time complexity is O(n×k)O(n×k) • **Average Case -** In the average case i.e.i.e. elements of the array are arbitrarily arranged, again we need to apply counting sort on the array for kk times. Hence in the average case also, the time complexity is O(n×k)O(n×k). **Radix Sort Space Complexity** Since we are using a temptemp array in the Counting Sort process of size nn and a countcount array of size 1010 the space complexity is O(n+b)O(n+b). Where bb is the base of elements in the original array, since in the above case we are dealing with decimals (base 10), b=10b=10.	4+1+1
2b.	Consider this array: [10, 21, 29, 41, 52] The difference between each adjacent term is almost equal to 10. Hence, this array has uniformly distributed data and can be sorted using the bucket sort algorithm. Consider another array: [1,4,23,5,44,9,6,43] This array is not uniformly distributed because the number of elements between the range [0-10] = 5 (i.e., 1,4,5,9, and 6), whereas the number of elements between [10-20] is 0, and the same for the range [30-40]. The data is scattered over different data ranges but is concentrated within some	2+2

	specific ranges, whereas sparse among the others. Hence, this array is not uniformly distributed. The complete data is scattered equally within each range and hence depicts uniformly distributed data.	
3a.	 Space Complexity for bucket sort is O(n+k), where n = number of elements in the array, and k = number of buckets formed Space taken by each bucket is O(k), and inside each bucket, we have n elements scattered. Hence, the space complexity becomes O(n+k). Time Complexity: If the array elements are not uniformly distributed, i.e., elements are concentrated within specific ranges. This will result in one or more buckets having more elements than other buckets, making bucket sort like any other sorting technique, where every element is compared to the other. Time complexity increases even further if the elements in the array are present in the reverse order. If insertion sort is used, the worst-case time complexity can go up to O(n2). If the array elements are uniformly distributed, bucket size will almost be the same for all the buckets. Hence, this will be the best case which will take up the least amount of time. Sorting time complexity will reduce even further if all the elements inside each bucket are already sorted. To create n buckets and scatter each element from the array, time complexity = O(n). If we use Insertion sort to sort each bucket, time complexity = O(k). Hence, best case time complexity for bucket sort = O(n+k), where n = number of elements, and k = number of buckets	4+1+1
3b	Use one stack as the one to hold all the entries in LIFO order, as usual. Let's call this Stack 1, and we'll maintain a pointer to its tail (only), as usual. To dequeue from Stack 1: If we want to remove the node from the front of the list, we'll need to pop off all the nodes from Stack 1 and push them onto Stack 2. The final node to be popped off of Stack 1 is the one we want. This whole operation takes O(n) time, due to 2 passes through the list: once to dequeue the nodes from Stack 1, and once to enqueue the nodes onto Stack 2. Finally, we can return all the nodes to Stack 1, which is again an O(n) operation for the reasons just described. Thus, we will use Stack 1 as the "master" list, and just	3+1

	use Stack 2 as the work list. To enqueue to Stack 1: Just push the new node onto Stack 1. This is an O(1) operation.	

4a.	(i)
	int length(struct node * head)
	{
	int count = 0;
	while (head != NULL)
	{
	count++;
	head = head->next;
	}
	return count;
	
	(ii)
	Reverse(t : node pointer): node pointer {
	rev : node pointer;
	temp: node pointer;
	rev = NULL;
	while(t !=NULL){
	temp = t.next;
	t.next = rev;
	rev = t;
	t = temp;
	}
	return (rev);
	}
		3+3

4b.	A stable sorting algorithm maintains the relative order of the items with equal sort keys. An unstable sorting algorithm does not. In other words, when a collection is sorted with a stable sorting algorithm, items with the same sort keys preserve their order after the collection is sorted.	2+2
5.	 Time Complexity: BFS: If we represent the graph G by adjacency matrix then the running time of BFS	2+2+1 +1

	algorithm is O(n), where n is the number of nodes. 2) If we represent the graph G by link lists then the running time of BFS algorithm is O(m + n), where m is the number of edges and n is the number of nodes On a **directed graph**, the average time complexity of DFS is O(V+|E|), where V is the number of vertices and E is the number of edges; on an **undirected graph**, the time complexity is O(V+2|E|)(each edge is visited twice). The average time complexity of **DFS on a tree** is O(V), where V is the number of nodes. Space Complexity The space complexity of the DFS algorithm is O(V), where V is the number of vertices.	