Faculty of Technology and Engineering

Department of Computer Science & Engineering

Date:27/05/2024

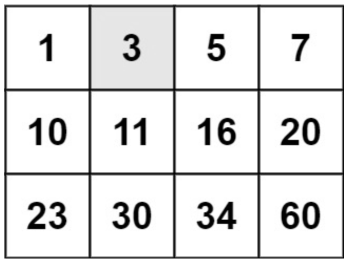
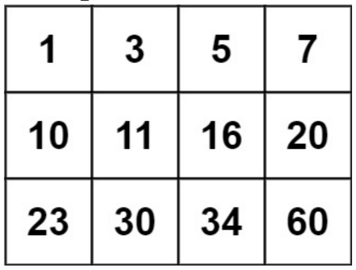
Practical List

Academic Year	:	2024-25	Semester	:	3rd
Course code	:	CSE203	Course name	:	Data Structures & Algorithms

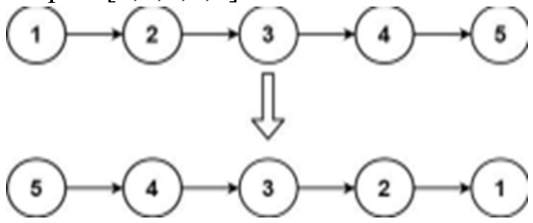
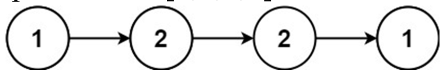
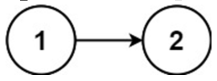
Sr. No.	Aim		Hours	CO
1.	Array data structure			1
	1.1	**Implement a program to print the pairs, if an integer array can be split into pairs such that each pair's sum of elements is divisible by a given positive integer, k.** **Example 1,** Input: arr[] = { 3, 1, 2, 6, 9, 4 } k = 5 Output: Pairs can be formed: (3, 2) (1, 9) (4, 6) Explanation: Array can be divided into pairs {(3, 2), (1, 9), (4, 6)} where the sum of elements in each pair is divisible by 5. **Example 2,** Input: arr[] = { 2, 9, 4, 1, 3, 5 } k = 6 Output: Pairs can be formed: (2, 4) (9, 3) (1, 5) Explanation: Array can be divided into pairs {(2, 4), (9, 3), (1, 5)} where the sum of elements in each pair is divisible by 6.	02 (HR)	

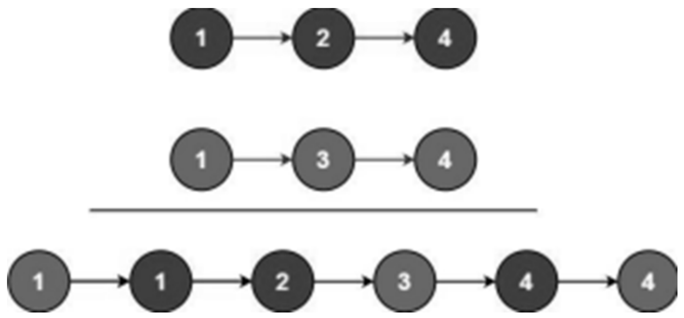
		Example 3, Input: arr[] = { 3, 1, 2, 6, 9, 4 } k = 6 Output: Pairs cannot be formed Explanation: Array cannot be divided into pairs where the sum of elements in each pair is divisible by 6.		
	1.2	**Implement a program to find the longest word in the input string and then calculate and print the number of characters in the word.** **Examples:** Input: There is a banana tree. Output : Longest word's length = 6 Input: pneumonoultramicroscopicsilicovolcanoconiosis is the longest word. Output : Longest word's length = 45		
	1.3	**Implement Linear Search and Binary Search using array data structure.** **Implement binary search for a long array of integers to find the required element. However, when the array size is quite large, the equation for finding the mid index may give the value which is out of range of integers. Implement the Binary search with a modified equation for finding mid.** **“the integer overflow problem” with binary search:** With a vast list of elements, “right” would be a very large value. Suppose your ‘left’ and ‘right’ are 16 bit unsigned integers. That means, they can only have a maximum value of 2^16 = 65536. **Example:** Consider: left = 65530 and right = 65531 left + right = 131061 (beyond the integer range). It will give garbage value It is known as an integer overflow. **Extra exercise:** **Implement Ternary search Algorithm.** It is a search algorithm that is used to find the position of a target value within a sorted array. It operates on the principle of dividing the array into three parts instead of two, as in binary search.	02	
	1.4	https://leetcode.com/problems/special-array-with-x-elements-greater-than-or-equal-x/description/?envType=daily-question&envId=2	02	

		024-05-27 You are given an array nums of non-negative integers. nums is considered special if there exists a number x such that there are exactly x numbers in nums that are greater than or equal to x. Notice that x does not have to be an element in nums. Return x if the array is special, otherwise, return -1. It can be proven that if nums is special, the value for x is unique. **Example 1**: Input: nums = [3,5] Output: 2 Explanation: There are 2 values (3 and 5) that are greater than or equal to 2. **Example 2**: Input: nums = [0,0] Output: -1 Explanation: No numbers fit the criteria for x. If x = 0, there should be 0 numbers >= x, but there are 2. If x = 1, there should be 1 number >= x, but there are 0. If x = 2, there should be 2 numbers >= x, but there are 0. x cannot be greater since there are only 2 numbers in nums. **Example 3**: Input: nums = [0,4,3,0,4] Output: 3 Explanation: There are 3 values that are greater than or equal to 3. **Constraints**: 1 <= nums.length <= 100 0 <= nums[i] <= 1000		
	1.5	**Matrix binary search:** You are given an m x n integer matrix with the following two properties: ● Each row is sorted in non-decreasing order. ● The first integer of each row is greater than the last integer of the previous row. Given an integer search key, return index if the target is in matrix or -1 otherwise. Perform this task **using the binary search algorithm**. **Example 1:**	02	

		 1 3 5 7 10 11 16 20 23 30 34 60 Input: matrix = 1 3 5 7 10 11 16 20 23 30 34 60 target = 3 Output: (0,1) **Example 2:**  1 3 5 7 10 11 16 20 23 30 34 60 Input: matrix = 1 3 5 7 10 11 16 20 23 30 34 60 target = 13 Output: -1 **Example 3:** 0 6 7 9 11 20 22 28 29 31 36 38 50 61 63 64 66 100 122 128 Target = 31 Output: (1,4)		
2.	**Sorting**			
	2.1	Implement Sorting Algorithm(s). A. Bubble Sort B. Selection Sort C. Insertion Sort **Extra exercise : implement radix sort.**	02 (VS)	

		Radix Sort is a linear sorting algorithm that sorts elements by processing them digit by digit. It is an efficient sorting algorithm for integers or strings with fixed-size keys.		
	2.2	Sort an array in **linear time** if all of its items are in ascending order except for two swapped elements. **Example:** Input: A[] = [3, 8, 6, 7, 5, 9] or [3, 5, 6, 9, 8, 7] or [3, 5, 7, 6, 8, 9] Output: A[] = [3, 5, 6, 7, 8, 9] **Note: You must solve this problem without using the library's sort function.** **Glossary**: **Linear Time** definition: The time complexity , denoted O(n), of an algorithm whose running time increases at most linearly with the size of the input. i.e. **do not use nested loops.**	02 (HR)	
	2.3	You will be given a zero-indexed array A. You need to rearrange its elements in such a way that the following conditions are satisfied: A[i] ≤ A[i+1] if i is even. A[i] ≥ A[i+1] if i is odd. In other words the following inequality should hold: A[0] ≤ A[1] ≥ A[2] ≤ A[3] ≥ A[4], and so on. Operations ≤ and ≥ should alter. **Input** The first line contains a single integer T denoting the number of test cases. The first line of each test case contains an integer N, that is the size of the array A. The second line of each test case contains the elements of array A **Output** For each test case, output a single line containing N space separated integers, which are the elements of A arranged in the required order. If there are more than one valid arrangements, you can output any of them. **Example**: 2 2 3 2 3 10 5 2	02 (HR)	

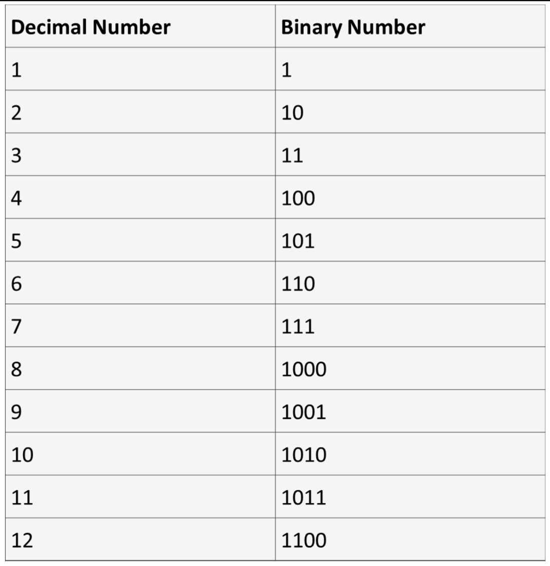
		Output: 2 3 2 10 5		
3.	Linked List			2
	3.1	Implement below operations of singly linked list. (a) Insert a node at front (b) Delete a node at last (c) **Delete Nth Node From End of List** (d) Delete all nodes of linked list **Note: Display content of linked list after each operation.**	02 (VS)	
	3.2	**(a) Reverse Linked List** Given a singly linked list, reverse the list, and return the reversed list. Input: head = [1,2,3,4,5] Output: [5,4,3,2,1]  **(b) Palindrome Linked List** Given a singly linked list, return true if it is a palindrome or false otherwise. (without using extra data structure) Example 1: Input: head = [1,2,2,1]  Output: true Example 2: Input: head = [1,2]  Output: false	02 (HR)	
	3.3	**Merge Two Sorted Lists** You are given the heads of two sorted linked lists list1 and list2. Merge the two lists in a one sorted list. The list should be made by splicing together the nodes of the first two lists. Return the merged linked list. **Example 1**:	02 (HR)	

		 Input: list1 = [1,2,4], list2 = [1,3,4] Output: [1,1,2,3,4,4]		
4.	Doubly linked list		02	
	4.1	Implement below operations of doubly linked lists. (a) Insert a node at front (b) Delete a node at last (c) Delete all nodes of linked list **Note: Display content of linked list after each operation.**	(VS)	
5.	Stack and queue			
	5.1	**(a) Implement stack using array** Implement a program to implement a Stack using Array. Your task is to use the class as shown in the comments in the code editor and complete the functions push () and pop () to implement a stack. Example 1: Input: push(2) push(3) pop() push(4) pop() Output: 3, 4 **(b) Implement Stack using Linked List** You have a linked list and you have to implement the functionalities push and pop of the stack using this given linked list. Your task is to use the class as shown in the comments in the code editor and complete the functions push() and pop() to implement a stack. **Example 1**: **Input:** push(2) push(3)	02 (HR)	

		pop()
		push(4)
		pop()
		Output: 3 4		

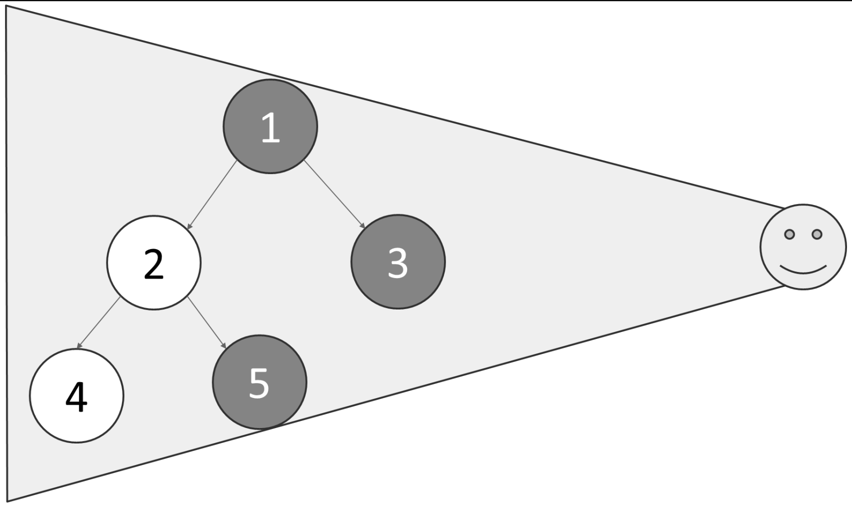
	5.2	**(a) Implement Queue using array** Implement a Queue using an Array. Queries in the Queue are of the following type: (i) 1 x (a query of this type means pushing 'x' into the queue) (ii) 2 (a query of this type means to pop element from queue and print the popped element) Example 1: Input: Q = 5 Queries = 1 2 1 3 2 1 4 2 Output: 2 3 **(b) Implement Queue using Linked List** A Query Q is of 2 Types (i) 1 x (a query of this type means pushing 'x' into the queue) (ii) 2 (a query of this type means to pop an element from the queue and print the popped element) Example 1: Input: Q = 5 Queries = 1 2 1 3 2 1 4 2 1 Output: 2 3	02 (HR)	

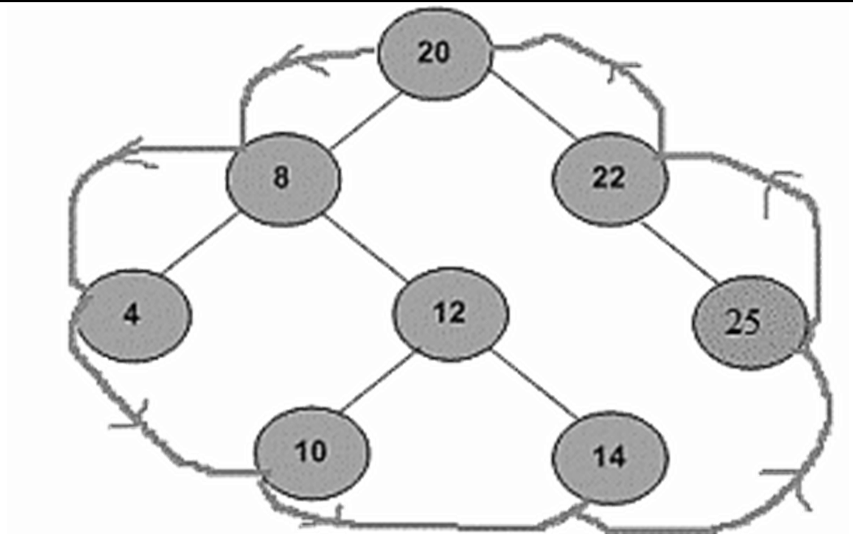
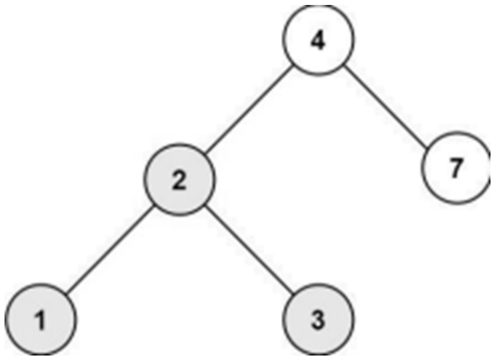
	5.3	**(a) Valid Parentheses** Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid. An input string is valid if: Open brackets must be closed by the same type of brackets. Open brackets must be closed in the correct order. Every close bracket has a corresponding open bracket of the same type. Example 1: Input: s = "()" Output: true **(b) Given an infix expression, the task is to convert it to a postfix expression.** Infix Expression: The expression of type a ‘operator’ b (a+b, where + is an operator) i.e., when the operator is between two operands. Postfix Expression: The expression of the form “a b operator” (ab+) i.e., When every pair of operands is followed by an operator. **Examples**: Input: A + B * C + D Output: ABC*+D+ Input: ((A + B) – C * (D / E)) + F Output: AB+CDE/*-F+ **(c) Given a postfix expression, the task is to evaluate the postfix expression.** Postfix expression: The expression of the form “a b operator” (ab+) i.e., when a pair of operands is followed by an operator. **Examples:** Input: str = “2 3 1 * + 9 -“ Output: -4 Explanation: If the expression is converted into an infix expression, it will be 2 + (3 * 1) – 9 = 5 – 9 = -4. **(d) Generate Binary Numbers from 1 to n using queue and its operations.** Given a number N, write a function that generates and prints all binary numbers with decimal values from 1 to N. **Example 1:** Input: n = 2 Output: 1, 10 **Example 2:** Input: n = 5 Output: 1, 10, 11, 100, 101	06 (HR)	

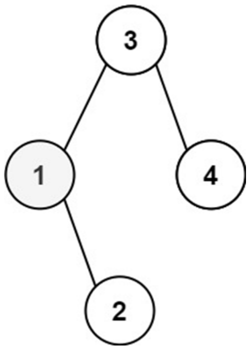
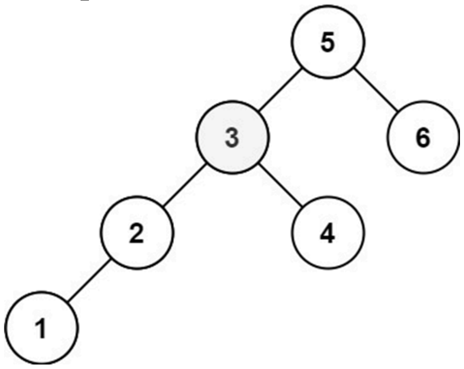


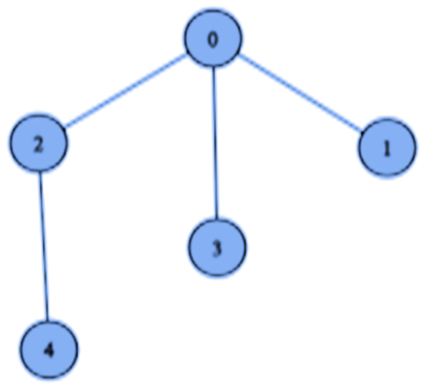
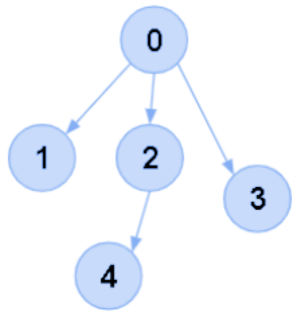
Decimal Number	Binary Number
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
12	1100

6	**Binary Tree (Doubly Linked List)**			
	6.1	Implement below operations on Binary Tree (a) Insert a node at Left (b) Delete a node at Right (c) Binary Tree Inorder Traversal (d) Binary Tree Preorder Traversal (e) Binary Tree Postorder Traversal (f) Binary Tree Level Order Traversal **Note: for (a) and (b) , First node will be the root node. Display the content of Binary Tree as per Inorder, Preorder, Postorder and Level Order Traversal.**	02 (VS)	
	6.2	**Maximum Depth of Binary Tree** Given the root of a binary tree, return its maximum depth. A binary tree's maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node. Example 1: Input: root = [3,9,20, null, null,15,7] Output: 3	02 (HR)	
	6.3	**Custom traversals** **(a) Binary Tree Right View** Given the root of a binary tree, imagine yourself standing on the right side of it, return the values of the nodes which you can see ordered from top to bottom.	04 (HR)	

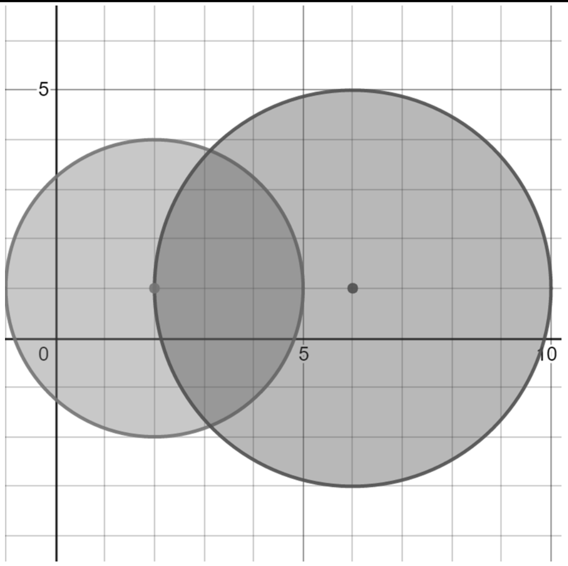
		 Example 1: Input: root = [1,2,3,4,5,null,null] Output: [1,3,5] Example 2: Input: root = [1,null,3] Output: [1,3] Example 3: Input: root = [] Output: [] **(b) Binary Tree boundary** Given a binary tree, print boundary nodes of the binary tree Anti-Clockwise starting from the root. The boundary includes: left boundary (nodes on left excluding leaf nodes) leaves (consist of only the leaf nodes) right boundary (nodes on right excluding leaf nodes)		

		 **Example:** root : 20 left- boundary nodes: 8 leaf nodes: 4 10 14 25 right – boundary nodes: 22		
7	Binary search tree			
	7.1	**Perform following operations on BST.** **(a)Insert a node in BST** **(b)Search a node in BST** You are given the root of a binary search tree (BST) and an integer val. Find the node in the BST that the node's value equals val and return the subtree rooted with that node. If such a node does not exist, return null  **Example 1:** Input: root = [4,2,7,1,3], val = 2 Output: [2,1,3]	02 (VS)	
	7.2	**Kth smallest element in BST**	02	

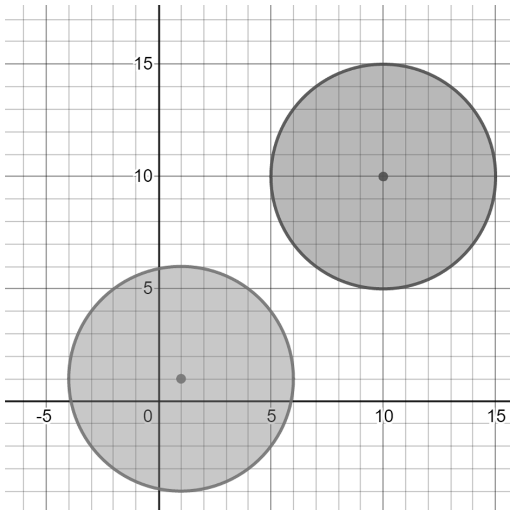
		Given the root of a binary search tree, and an integer k, return the kth smallest value (1-indexed) of all the values of the nodes in the tree.  Example 1: Input: root = [3,1,4,null,2], k = 1 Output: 1 **Example 2:**  Input: root = [5,3,6,2,4,null,null,1], k = 3 Output: 3	(HR)	
8	Graph			
	8.1	**(a) DFS of Graph** You are given a connected undirected graph. Perform a Depth First Traversal of the graph. Note: Use a recursive approach to find the DFS traversal of the graph starting from the 0th vertex from left to right according to the graph. Example 1: Input: V = 5 , adj = [[2,3,1] , [0], [0,4], [0], [2]]	04 (VS)	

		 Output: 0 2 4 3 1 Explanation: 0 is connected to 2, 3, 1. 1 is connected to 0. 2 is connected to 0 and 4. 3 is connected to 0. 4 is connected to 2. so starting from 0, it will go to 2 then 4, and then 3 and 1. Thus dfs will be 0 2 4 3 1. **(b) BFS of graph** Given a directed graph. The task is to do Breadth First Traversal of this graph starting from 0. **Note:** One can move from node u to node v only if there's an edge from u to v and find the BFS traversal of the graph starting from the 0th vertex, from left to right according to the graph. Also, you should only take nodes directly or indirectly connected from Node 0 in consideration. **Example 1:** **Input:**  Output: 0 1 2 3 4 Explanation: 0 is connected to 1 , 2 , 3.		
	8.2	**Course Schedule (Cycle detection in graph)** There are a total of “numCourses” courses you have to take, labeled from 0 to numCourses - 1. You are given an array named	02 (HR)	

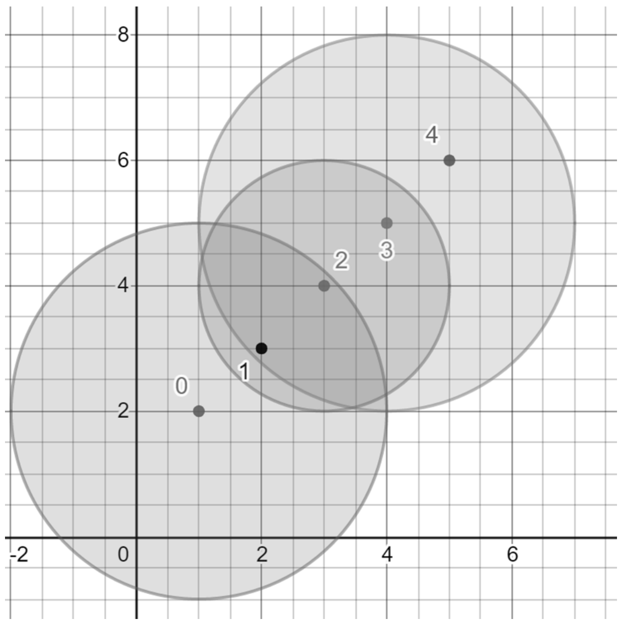
		prerequisites where prerequisites[i] = [ai, bi] indicates that you must take course bi first if you want to take course ai. For example, the pair [0, 1], indicates that to take course 0 you have to first take course 1. Return true if you can finish all courses. Otherwise, return false. **Example 1:** Input: numCourses = 2, prerequisites = [[1,0]] Output: true Explanation: There are a total of 2 courses to take. To take course 1 you should have finished course 0. So it is possible. **Example 2:** Input: numCourses = 2, prerequisites = [[1,0],[0,1]] Output: false Explanation: There are a total of 2 courses to take. To take course 1 you should have finished course 0, and to take course 0 you should also have finished course 1. So it is impossible.		
	8.3	**Maximum Bomb detonation** **(Graph edge creation and search)** You are given a list of bombs. The range of a bomb is defined as the area where its effect can be felt. This area is in the shape of a circle with the center as the location of the bomb. The bombs are represented by a 0-indexed 2D integer array “bombs” where bombs[i] = [xi, yi, ri]. xi and yi denote the X-coordinate and Y-coordinate of the location of the ith bomb, whereas ri denotes the radius of its range. You may choose to detonate a single bomb. When a bomb is detonated, it will detonate all bombs that lie in its range. These bombs will further detonate the bombs that lie in their ranges. Given the list of bombs, return the maximum number of bombs that can be detonated if you are allowed to detonate only one bomb. **Example 1:** Input: bombs = [[2,1,3],[6,1,4]] Output: 2 Explanation: The above figure shows the positions and ranges of the 2 bombs. If we detonate the left bomb, the right bomb will not be affected. But if we detonate the right bomb, both bombs will be detonated. So the maximum bombs that can be detonated is max(1, 2) = 2.	02 (HR)	



Example 2:
Input: bombs = [[1,1,5],[10,10,5]]
Output: 1
Explanation:
Detonating either bomb will not detonate the other bomb, so the maximum number of bombs that can be detonated is 1.



Example 3:
Input: bombs = [[1,2,3],[2,3,1],[3,4,2],[4,5,3],[5,6,4]]
Output: 5
Explanation:
The best bomb to detonate is bomb 0 because:
- Bomb 0 detonates bombs 1 and 2. The red circle denotes the range of bomb 0.
- Bomb 2 detonates bomb 3. The blue circle denotes the range of bomb 2.
- Bomb 3 detonates bomb 4. The green circle denotes the range of bomb 3.

		Thus all 5 bombs are detonated. 		
9	Hashing			
	9.1	**Implementing a Hash Table for Student Records Management** **Background**: You are tasked with implementing a hybrid hash table to manage student records efficiently. Each student record consists of a unique student ID (key) and the corresponding student score (data). The hash table will support two methods of collision handling: **separate chaining and linear probing**. This flexibility ensures efficient handling of collisions, enabling you to choose the most suitable method based on different scenarios. **Objectives**: **Hash Table Initialization:** Create a hash table with an appropriate size that is a prime number greater than the initially specified size to reduce collisions. Implement a hash function that uses the modulo operation to map keys to indices. **Insertion of Records:** Implement functionality to insert student records into the hash table.	04 (VS)	

Implement both chaining and linear probing for collision handling separately.

Deletion of Records:
Implement functionality to delete a record by its key.
Ensure the integrity of the hash table after deletion for both collision handling methods.
Display of Records:
Implement functionality to display the contents of the hash table.
Show all student records stored at each index, clearly indicating which collision handling method is being used.

Requirements:
Hash Table Initialization:
The hash table should be initialized with a size that is the next prime number greater than the specified initial size.
The hash function should use the modulo operation to map keys to indices.
Insertion:
The insertItem method should allow adding a student ID and score to the hash table.
For separate chaining, use linked lists to handle collisions.
For linear probing, find the next available slot in case of a collision.
Deletion:
The deleteItem method should allow removing a student record based on the student ID.
Ensure that the hash table remains functional after a record is deleted.
Display:
The displayHash method should output the entire hash table, showing all student records stored at each index.
Clearly distinguish between the records stored using separate chaining and linear probing.

Constraints:
Student IDs are integers.
Student scores are also integers.
The hash table should handle multiple student records having the same hash index using linked lists (separate chaining) or linear probing.

Example Usage:
Initialize a hash table with an initial size of 6.

Insert the following student records into the hash table using separate chaining:

Student ID: 231, Score: 123
Student ID: 326, Score: 432
Student ID: 212, Score: 523
Student ID: 321, Score: 43
Student ID: 433, Score: 423
Student ID: 262, Score: 111
Display the hash table.

Delete the record with student ID 212.

Display the hash table again to show the updated records.

Repeat the insertion and deletion steps using linear probing instead of separate chaining, and display the hash table after each operation.

Sample Output:

```
Using Separate Chaining:
table[0] --> (231, 123)
table[1]
table[2] --> (212, 523)
table[3] --> (262, 111)
table[4] --> (326, 432)
table[5]
table[6] --> (321, 43) --> (433, 423)

After deleting record with student ID 212:
table[0] --> (231, 123)
table[1]
table[2]
table[3] --> (262, 111)
table[4] --> (326, 432)
table[5]
table[6] --> (321, 43) --> (433, 423)

Using Linear Probing:
table[0] --> (231, 123)
table[1] --> (321, 43)
table[2]
table[3] --> (433, 423)
table[4] --> (326, 432)
table[5] --> (212, 523)
table[6] --> (262, 111)
```

		After deleting record with student ID 212: table[0] --> (231, 123) table[1] --> (321, 43) table[2] table[3] --> (433, 423) table[4] --> (326, 432) table[5] table[6] --> (262, 111) **Implementation Notes:** Ensure that the hash table size is a prime number for better distribution of keys. Use separate chaining with linked lists and linear probing to handle collisions effectively. Implement the insertItem, deleteItem, and displayHash methods to manage and display student records as required.		