National University of Computer and Emerging Sciences

Karachi Campus

Data Structures (CS2001)

Sessional-II Exam

Date: 5th Nov 2025

Course Instructor(s)

Dr. Jawwad, Dr. Anam, Dr. Farrukh, Mr. Basit, Mr. Farooq, Mr. Nouman, Ms. Sania, Ms. Fizza, Ms. Mubashara

Total Time (Hrs):	**1**
Total Marks:	**15**
Total Questions:	**03**

Do not write below this line

Attempt all the questions.

_____ _____ _____

Roll No Section Student Signature

CLO # 1: *Use & explain* concepts related to basic and advanced data structures and describe their usage in terms of common algorithmic operations

Q1: PTCL maintains a directory of customer phone numbers. The newer contacts have 10-digit phone numbers, while older contacts still use 7-digit phone numbers. PTCL classifies a number based on its area code. For a 10-digit phone number, the first three digits represent its area code. For a 7-digit phone number, its area code is considered as "000", meaning that for sorting purposes you must treat them as belonging to the zero area code group. You are required to complete the following function that sorts all phone numbers in ascending order of their area codes using a non-comparison-based algorithm while maintaining stability. The function receives an array of phone numbers and its size, and must return a new array containing the sorted phone numbers (in their original form, without added zeros). ***[5 marks]***

int sort_ptcl_numbers(int* array_in, int size);*

Example Input:	Expected Output:
Total number of Phone Numbers: 7 9234567890 4567890 1002003 7890123456 1234567 9998887776 1000000	4567890 1002003 1234567 1000000 7890123456 9234567890 9998887776

```
#include <bits/stdc++.h>
using namespace std;

// Function to extract area code
int getAreaCode(long long num) {
   int digits = (num == 0) ? 1 : log10(num) + 1;
   if (digits == 10)
      return num / 10000000; // First 3 digits
   return 0; // For 7-digit numbers, area code = 000
}
```

```
void countSort(vector<long long> &arr, int exp) {
    vector<long long> output(arr.size());
    int count[10] = {0};

    for (long long num : arr) {
        int areaCode = getAreaCode(num);
        count[(areaCode / exp) % 10]++;
    }

    for (int i = 1; i < 10; i++)
        count[i] += count[i - 1];

    for (int i = arr.size() - 1; i >= 0; i--) {
        long long num = arr[i];
        int areaCode = getAreaCode(num);
        int index = (areaCode / exp) % 10;
        output[count[index] - 1] = num;
        count[index]--;
    }

    arr = output;
}

void radixSort(vector<long long> &arr) {
    int maxArea = 0;
    for (long long num : arr)
        maxArea = max(maxArea, getAreaCode(num));

    for (int exp = 1; maxArea / exp > 0; exp *= 10)
        countSort(arr, exp);
}

int main() {
    int n;
    cin >> n;

    vector<long long> phone(n);
    for (int i = 0; i < n; i++)
        cin >> phone[i];

    radixSort(phone);

    for (long long num : phone)
        cout << num << endl;

    return 0; }
```

CLO # 3: *Compare* different data structures in terms of their relative efficiency and *design* effective solutions and algorithms that make use of them.

Q2: Suppose you are analyzing a large website's internal structure.Each webpage can have links to child pages, forming a tree-like hierarchy, for example: ***[2+3=5 marks]***

```
Home
├── About
├── Products
│   ├── Phones
│   └── Laptops
└── Contact
```

Ideally, this should form a tree (a structure without cycles). However, due to malicious scripts or poor design, some pages may link back to their ancestors thus creating hidden redirect loops. In such cases a page eventually leads back to one of its parent pages causing infinite redirects.

You are asked to design an algorithm to detect whether such a loop exists in the website's structure. You must keep track of the active chain of pages you are currently visiting, so you can tell if a newly visited page has already appeared earlier in the current exploration path. You may use either a queue or a stack, depending on which best supports this kind of traversal.

A. Which data structure would be more appropriate for detecting these hidden redirect loops, a queue or a stack? Explain your choice, and briefly but clearly.
B. Write an algorithm would use the queue or stack to identify cycles in the website tree.

A stack is more appropriate for detecting hidden redirect loops in the website's structure.

Reason:
A stack follows the Last In, First Out (LIFO) principle, which is ideal for tracking the current path of pages being visited. When we explore links deeply from one page to another (like going from Home → Products → Laptops), each new page is pushed onto the stack.
If we ever find a link that points back to a page already in the stack, it means we've returned to an ancestor — a loop or cycle exists.

How the algorithm works (using DFS with a stack):
Start from the root page (e.g., Home) and push it onto the stack.
Visit one of its linked (child) pages and push it onto the stack.
Continue this process, going deeper into the structure.
Before visiting any new page, check if it is already in the stack:
If yes, a redirect loop is detected (the page links back to an ancestor).
If no, push it and continue exploring.
When a page has no more child links, pop it from the stack (backtrack).
This method ensures we always know the active chain of pages we are currently exploring, which is necessary to detect loops.

If we used a queue (FIFO) instead, we would lose track of which pages are still "active" in the current path, so we couldn't reliably find these redirect loops.

CLO # 3: *Compare* different data structures in terms of their relative efficiency and *design* effective solutions and algorithms that make use of them.

Q3: A courier company keeps package IDs in the order they arrive. Company wants a self-balancing binary search tree (AVL) to keep lookups and insertions fast. ***[2+2+1=5 marks]***

Given sequence of package IDs (in arrival order):

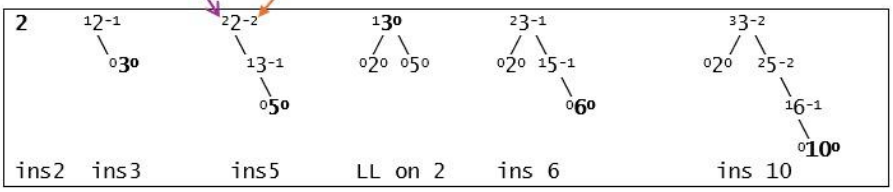
2, 3, 5, 6, 10, 8, 9, 4, 1

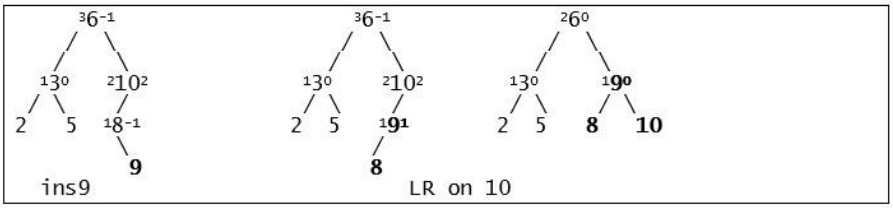
A. Starting with an empty tree, insert the keys one by one into an AVL tree in the given order.
B. After each insertion do the following and show it clearly:

- Draw the tree (show parent/child relationships).
- Write the height of each affected node and the balance factor.
- If the tree becomes unbalanced after an insertion, identify the imbalance case precisely as one of: LL, LR, RR, RL.
- State which rotation(s) you perform (single rotation: left or right; or double rotation: left-right or right-left). Show the tree before and after the rotation.

C. After processing all insertions, draw the final AVL tree and list its in-order traversal (to verify BST property).

Solution: height is mentioned on the left and BF is mentioned on the right of each node in superscript. height balance factor





---------- The End ----------