Mid lab assessment topics

Excellent, Abdullah ✅
Let’s go topic-by-topic and write **proper C++ codes with full explanations** so you can practice for your **Mid Lab Assessment** confidently.

🧮 Part i: Vector Operations

✅ 1. Finding Vector Size, Insert, Update, Delete

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

int main() {
    vector<int> v = {10, 20, 30};

    // Finding size
    cout << "Size: " << v.size() << endl;

    // Inserting elements
    v.push_back(40); // insert at end
    v.insert(v.begin() + 1, 15); // insert 15 at index 1

    // Updating element
    v[2] = 25; // change value at index 2

    // Deleting element
    v.erase(v.begin() + 3); // delete element at index 3
    v.pop_back(); // remove last element

    // Displaying vector
    cout << "Final vector: ";
    for (int x : v) cout << x << " ";
}
```

🧠 Explanation:

- `v.push_back(x)` → adds element at the end

- `v.insert(v.begin()+i, x)` → insert at index `i`
- `v[i]` → updates element
- `v.erase(v.begin()+i)` → removes element at index `i`
- `v.pop_back()` → removes last element

✅ 2. 2D Vector (Insert and Display)

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

int main() {
    vector<vector<int>> v2d = {
        {1, 2, 3},
        {4, 5},
        {6}
    };

    // Insert new element in 2nd row
    v2d[1].push_back(9);

    // Insert a new row
    v2d.push_back({7, 8, 9});

    // Display 2D vector
    for (int i = 0; i < v2d.size(); i++) {
        for (int j = 0; j < v2d[i].size(); j++) {
            cout << v2d[i][j] << " ";
        }
        cout << endl;
    }
}
```

🧠 Explanation:

A 2D vector is a vector of vectors.

You can access elements using `v2d[row][col]`.

Rows and columns can have different sizes.

✅ 3. Sorting (Ascending & Descending)

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

int main() {
    vector<int> v = {5, 1, 4, 2, 3};

    // Ascending sort
    sort(v.begin(), v.end());
    cout << "Ascending: ";
    for (int x : v) cout << x << " ";
    cout << endl;

    // Descending sort
    sort(v.begin(), v.end(), greater<int>());
    cout << "Descending: ";
    for (int x : v) cout << x << " ";
}
```

🧠 Explanation:

- `sort(v.begin(), v.end())` → ascending
- `sort(v.begin(), v.end(), greater<int>())` → descending

💡 Part ii: Problem Solving

✅ 1. Trailing Zeroes in Factorial

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

int main() {
    int n;
    cout << "Enter n: ";
    cin >> n;

    int count = 0;
    for (int i = 5; n / i >= 1; i *= 5)
        count += n / i;

    cout << "Trailing zeroes in " << n << "! = " << count;
}
```

🧠 Explanation:

Every `5` in the factorial contributes to a trailing zero (with a matching 2).

We count multiples of 5, 25, 125, etc.

✅ 2. Count Digits in Factorial

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

int main() {
    int n;
    cout << "Enter n: ";
    cin >> n;

    if (n == 0 || n == 1) {
        cout << "Digits in " << n << "! = 1";
        return 0;
    }

    double digits = 0;
    for (int i = 2; i <= n; i++)
        digits += log10(i);

    cout << "Digits in " << n << "! = " << floor(digits) + 1;
}
```

🧠 Explanation:

Instead of calculating large factorials, we use **logarithms** to count the number of digits:

$$\text{digits} = \lfloor \log_{10}(n!) \rfloor + 1$$

✅ 3. Only Pluses

Problem idea:

Given an array, find how many additions are needed to make all elements equal.

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

int main() {
    vector<int> v = {1, 3, 5};
    int mx = *max_element(v.begin(), v.end());
    int moves = 0;

    for (int x : v)
        moves += mx - x;

    cout << "Minimum + operations needed = " << moves;
}
```

🧠 Explanation:

Make all numbers equal to the **maximum** by adding the difference.

✅ 4. Make It Beautiful (Example Type)

Given an array, rearrange it so that no prefix sum equals the total sum.

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

bool makeBeautiful(vector<int> &a) {
    sort(a.begin(), a.end());
    int total = accumulate(a.begin(), a.end(), 0);
    int prefix = 0;
    for (int i = 0; i < a.size(); i++) {
        prefix += a[i];
        if (prefix == total) return false;
    }
    return true;
}

int main() {
    vector<int> a = {3, 1, 2};
    if (makeBeautiful(a)) cout << "YES";
    else cout << "NO";
}
```

🧠 Explanation:

Check if rearranging (e.g., sorting) avoids a prefix sum equal to total sum.

⚙️ Part iii: Elementary Algorithms

✅ 1. Linear Search

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

int main() {
    vector<int> a = {5, 3, 8, 2};
    int key = 8;
    bool found = false;

    for (int i = 0; i < a.size(); i++) {
        if (a[i] == key) {
            cout << "Found at index " << i;
            found = true;
            break;
        }
    }
    if (!found) cout << "Not found";
}
```

✅ 2. Binary Search

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

int main() {
    vector<int> a = {2, 3, 4, 10, 40};
    int key = 10;

    int l = 0, r = a.size() - 1;
    bool found = false;
    while (l <= r) {
        int mid = (l + r) / 2;
        if (a[mid] == key) {
            cout << "Found at index " << mid;
            found = true;
            break;
        } else if (a[mid] < key)
            l = mid + 1;
        else
            r = mid - 1;
    }
    if (!found) cout << "Not found";
}
```

✅ 3. Bubble Sort

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

int main() {
    vector<int> a = {5, 1, 4, 2, 8};
    int n = a.size();

    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (a[j] > a[j + 1])
                swap(a[j], a[j + 1]);
        }
    }

    cout << "Sorted: ";
    for (int x : a) cout << x << " ";
}
```

✅ 4. Selection Sort

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

int main() {
    vector<int> a = {29, 10, 14, 37, 13};
    int n = a.size();

    for (int i = 0; i < n - 1; i++) {
        int minIdx = i;
        for (int j = i + 1; j < n; j++)
            if (a[j] < a[minIdx])
                minIdx = j;
        swap(a[i], a[minIdx]);
    }

    cout << "Sorted: ";
    for (int x : a) cout << x << " ";
}
```

✅ 5. Insertion Sort

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

int main() {
    vector<int> a = {12, 11, 13, 5, 6};
    int n = a.size();

    for (int i = 1; i < n; i++) {
        int key = a[i];
        int j = i - 1;
        while (j >= 0 && a[j] > key) {
            a[j + 1] = a[j];
            j--;
        }
        a[j + 1] = key;
    }

    cout << "Sorted: ";
    for (int x : a) cout << x << " ";
}
```

✅ 6. Maximum Subarray Sum (Kadane’s Algorithm)

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

int main() {
    vector<int> arr = {-2, 1, -3, 4, -1, 2, 1, -5, 4};

    int maxSum = INT_MIN, curr = 0;
    for (int x : arr) {
        curr = max(x, curr + x);
        maxSum = max(maxSum, curr);
    }

    cout << "Maximum Subarray Sum = " << maxSum;
}
```

🧠 Explanation:

Kadane’s algorithm keeps track of the **maximum sum ending here** and the **global maximum**.

Would you like me to make a **PDF “Mid Lab Practice Booklet”** with all these codes, outputs, and diagrams (so you can print or revise offline)?