Lecture 01: Introduction, Algorithm Analysis, Big-O Notation

C++ Code Samples — Sedgwick Algorithms Course — lecture-01-samples.cpp

```cpp
// ============================================================================
// Lecture 01: Introduction, Algorithm Analysis, Big-O Notation
// Sedgwick Algorithms Course
//
// Topics covered:
//   - Binary search (O(log n)) vs linear search (O(n))
//   - Counting operations in nested loops
//   - Timing two approaches to sum 1..N (loop vs formula)
//   - Three-sum problem (brute force O(n^3))
// ============================================================================

#include <iostream>
#include <vector>
#include <algorithm>
#include <chrono>

using namespace std;

// === SECTION: Linear Search O(n) ===
// Scans every element left to right. Worst case examines all n elements.
int linearSearch const vector<int>& a, int target, int& comparisons) {
    comparisons = 0
    for (int i = 0; i < a.size(); i++) {
        comparisons++
        if (a[i] == target) return i;
    }
    return -1
}

// === SECTION: Binary Search O(log n) ===
// Requires a sorted array. Halves the search space each step.
int binarySearch const vector<int>& a, int target, int& comparisons) {
    comparisons = 0
    int lo = 0, hi = a.size() - 1
    while (lo <= hi) {
        comparisons++
        int mid = lo + (hi - lo) / 2
        if (a[mid] == target) return mid;
        else if (a[mid] < target) lo = mid + 1
        else hi = mid - 1
    }
    return -1
}

// === SECTION: Counting Operations in Nested Loops ===
// Demonstrates how nested loops lead to O(n^2) and O(n^3) growth.
void countOperations
    << "\n--- Counting Operations in Nested Loops ---\n"

    for (int n : {10, 100, 1000}) {
        long long countSingle = 0, countDouble = 0, countTriple = 0

        // Single loop: O(n)
        for (int i = 0; i < n; i++) countSingle++

        // Double nested loop: O(n^2)
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                countDouble++

        // Triple nested loop: O(n^3) -- only run for small n
```

```
        if (n <= 100) {
            for (int i = 0; i < n; ++i)
                for (int j = 0; j < n; ++j)
                    for (int k = 0; k < n; ++k)
                        countTriple++;
        }

        cout << "  n=" << n
             << "  O(n)=" << countSingle
             << "  O(n^2)=" << countDouble;
        if (n <= 100)
            cout << "  O(n^3)=" << countTriple;
        else
            cout << "  O(n^3)=skipped (too slow)";
        cout << "\n";
    }
}

// === SECTION: Sum 1..N -- Loop vs Formula ===
// Loop approach is O(n). Gauss's formula is O(1).
void compareSumApproaches() {
    cout << "\n--- Timing Sum 1..N: Loop O(n) vs Formula O(1) ---\n";

    long long n = 100000000LL;   // 100 million

    // Approach 1: Loop O(n)
    auto start = chrono::high_resolution_clock::now();
    long long sumLoop = 0;
    for (long long i = 1; i <= n; ++i) sumLoop += i;
    auto end = chrono::high_resolution_clock::now();
    double ms = chrono::duration<double, milli>(end - start).count();

    // Approach 2: Gauss formula O(1)
    auto start2 = chrono::high_resolution_clock::now();
    long long sumFormula = n * (n + 1) / 2;
    auto end2 = chrono::high_resolution_clock::now();
    double ms2 = chrono::duration<double, milli>(end2 - start2).count();

    cout << "  N = " << n << "\n";
    cout << "  Loop sum    = " << sumLoop    << "  (" << ms  << " ms)\n";
    cout << "  Formula sum = " << sumFormula << "  (" << ms2 << " ms)\n";
    cout << "  Results match: " << (sumLoop == sumFormula ? "YES" : "NO") << "\n";
}

// === SECTION: Three-Sum Problem (Brute Force O(n^3)) ===
// Count the number of triples (i, j, k) where a[i]+a[j]+a[k] == 0.
int threeSumCount(const vector<int>& a) {
    int n = (int)a.size();
    int count = 0;
    for (int i = 0; i < n; ++i)
        for (int j = i + 1; j < n; ++j)
            for (int k = j + 1; k < n; ++k)
                if (a[i] + a[j] + a[k] == 0)
                    count++;
    return count;
}

void threeSumDemo() {
    cout << "\n--- Three-Sum Problem (Brute Force O(n^3)) ---\n";

    vector<int> a = {-40, -20, -10, 0, 5, 10, 30, 40};
    cout << "  Array: ";
    for (int x : a) cout << x << " ";
    cout << "\n";
```

```
    auto start = chrono::high_resolution_clock::now();
    int count = threeSumCount(data);
    auto end = chrono::high_resolution_clock::now();
    double ms = chrono::duration<double, milli>(end - start).count();

    cout << "  Triples summing to zero: " << count << "\n";
    cout << "  Time: " << ms << " ms\n";

    // Show the actual triples
    cout << "  Triples found:\n";
    int n = data.size();
    for (int i = 0; i < n; ++i)
        for (int j = i + 1; j < n; ++j)
            for (int k = j + 1; k < n; ++k)
                if (data[i] + data[j] + data[k] == 0)
                    cout << "    (" << data[i] << ", "
                         << data[j] << ", " << data[k] << ")\n";
}

// === MAIN ===
int main() {
    cout << "========================================\n";
    cout << " Lecture 01: Algorithm Analysis & Big-O\n";
    cout << "========================================\n";

    // --- Binary Search vs Linear Search ---
    cout << "\n--- Binary Search O(log n) vs Linear Search O(n) ---\n";
    vector<int> sorted(10000);
    for (int i = 0; i < 10000; ++i) sorted[i] = i * 2;  // even numbers 0..19998

    int target = 9998;  // exists in array
    int linComp = 0, binComp = 0;

    int linIdx = linearSearch(sorted, target, linComp);
    int binIdx = binarySearch(sorted, target, binComp);

    cout << "  Array size: " << sorted.size() << "\n";
    cout << "  Searching for: " << target << "\n";
    cout << "  Linear search: found at index " << linIdx
         << ", comparisons = " << linComp << "\n";
    cout << "  Binary search: found at index " << binIdx
         << ", comparisons = " << binComp << "\n";

    // --- Nested Loop Operation Counts ---
    countOperations();

    // --- Sum 1..N Timing ---
    compareSumApproaches();

    // --- Three-Sum ---
    threeSumDemo();

    return 0;
}
```