Lecture 03: Elementary Sorting (Selection, Insertion, Shellsort)

C++ Code Samples — Sedgwick Algorithms Course — lecture-03-samples.cpp

```cpp
// ============================================================================
// Lecture 03: Elementary Sorting (Selection, Insertion, Shellsort)
// Sedgwick Algorithms Course
//
// Topics covered:
//   - Selection sort with step-by-step output
//   - Insertion sort with step-by-step output
//   - Shellsort with Knuth's increment sequence (3x+1)
//   - Helper function to print array state
//   - Comparison of swap/compare counts between all three
// ============================================================================

#include <iostream>
#include <vector>
#include <string>
#include <iomanip>

using namespace std;

// === SECTION: Helper -- Print Array State ===
// Prints the array with an optional marker showing the current position.
void printArray(const vector<int>& arr, const string& label = "",
                int marker = -1) {
    if (!label.empty()) cout << "    " << label << ": ";
    else cout << "    ";
    cout << "[";
    for (int i = 0; i < (int)arr.size(); i++) {
        if (i > 0) cout << ", ";
        if (i == marker) cout << "(" << arr[i] << ")";
        else             cout << " " << arr[i] << " ";
    }
    cout << "]\n";
}

// === SECTION: Selection Sort ===
// Find the minimum of the unsorted portion, swap it into place.
// Always O(n^2) comparisons, O(n) swaps.
void selectionSort(vector<int>& arr, bool verbose = false) {
    int n = (int)arr.size();
    for (int i = 0; i < n - 1; i++) {
        int minIdx = i;
        for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[minIdx]) minIdx = j;
        }
        swap(arr[i], arr[minIdx]);
        if (verbose) {
            cout << "    Pass " << i + 1 << ": swapped arr[" << i
                 << "]=" << arr[i] << " with min at [" << minIdx << "] -> ";
            printArray(arr);
        }
    }
}

// === SECTION: Insertion Sort ===
// Slide each element left into its correct position among sorted prefix.
// Best case O(n) for nearly sorted data, worst case O(n^2).
void insertionSort(vector<int>& arr, bool verbose = false) {
    int n = (int)arr.size();
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
        if (verbose) {
            cout << "    Insert " << key << " at position " << j + 1 << ": ";
            printArray(arr);
        }
    }
}

// === SECTION: Shellsort with Knuth's Increment Sequence ===
// Uses h-sorting with gaps 1, 4, 13, 40, 121, ... (3h+1).
// Moves elements many positions at once, then refines.
// Empirically sub-quadratic, roughly O(n^(3/2)) for Knuth sequence.
void shellSort(vector<int>& arr, bool verbose = false) {
    int n = (int)arr.size();

    // Compute the largest h in Knuth's sequence that fits
    int h = 1;
    while (h < n / 3) h = 3 * h + 1;  // 1, 4, 13, 40, 121, ...

    if (verbose) {
        cout << "    Knuth gaps: ";
        int temp = h;
        while (temp >= 1) { cout << temp << " "; temp /= 3; }
        cout << "\n";
    }

    while (h >= 1) {
        // h-sort the array (insertion sort with stride h)
        for (int i = h; i < n; i++) {
            int key = arr[i];
            int j = i;
            while (j >= h && arr[j - h] > key) {
                arr[j] = arr[j - h];
                j -= h;
            }
            arr[j] = key;
        }
        if (verbose) {
            cout << "    After h=" << h << " sort: ";
            printArray(arr);
        }
        h /= 3;
    }
}

// === SECTION: Instrumented Versions for Counting ===
// These versions count comparisons and swaps for performance comparison.

struct SortStats {
    long long compares;
    long long swaps;
};

SortStats selectionSortCounted(vector<int>& arr) {
    SortStats st = {0, 0};
    int n = (int)arr.size();
    for (int i = 0; i < n - 1; i++) {
        int minIdx = i;
        for (int j = i + 1; j < n; j++) {
            st.compares++;
            if (arr[j] < arr[minIdx]) minIdx = j;
        }
        if (minIdx != i) { swap(arr[i], arr[minIdx]); st.swaps++; }
    }
    return st;
}

SortStats insertionSortCounted(vector<int>& arr) {
    SortStats st = {0, 0};
    int n = (int)arr.size();
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;
        while (j >= 0) {
            st.compares++;
            if (arr[j] > key) { arr[j + 1] = arr[j]; st.swaps++; j--; }
            else break;
        }
        arr[j + 1] = key;
    }
    return st;
}

SortStats shellSortCounted(vector<int>& arr) {
    SortStats st = {0, 0};
    int n = (int)arr.size();
    int h = 1;
    while (h < n / 3) h = 3 * h + 1;
    while (h >= 1) {
        for (int i = h; i < n; i++) {
            int key = arr[i];
            int j = i;
            while (j >= h) {
                st.compares++;
                if (arr[j - h] > key) { arr[j] = arr[j - h]; st.swaps++; j -= h; }
                else break;
            }
            arr[j] = key;
        }
        h /= 3;
    }
    return st;
}

// === MAIN ===
int main() {
    cout << "=============================================\n";
    cout << " Lecture 03: Elementary Sorting Algorithms\n";
    cout << "=============================================\n";

    // --- Selection Sort (verbose) ---
    cout << "\n--- Selection Sort (step-by-step) ---\n";
    vector<int> a1 = {64, 25, 12, 22, 11};
    cout << "  Input: "; printArray(a1);
    selectionSort(a1, true);
    cout << "  Sorted: "; printArray(a1);

    // --- Insertion Sort (verbose) ---
    cout << "\n--- Insertion Sort (step-by-step) ---\n";
    vector<int> a2 = {64, 25, 12, 22, 11};
    cout << "  Input: "; printArray(a2);
    insertionSort(a2, true);
    cout << "  Sorted: "; printArray(a2);
    // --- Shellsort (verbose) ---
    cout << "\n--- Shellsort with Knuth's Sequence (step-by-step) ---\n";
    vector<int> a3 = {82, 31, 56, 12, 95, 44, 18, 67, 23, 73, 39, 50};
    cout << "  Input: "; printArray(a3);
    shellSort(a3, true);
    cout << "  Sorted: "; printArray(a3);

    // --- Comparison of Swap/Compare Counts ---
    cout << "\n--- Compare & Swap Counts (n=1000, random data) ---\n";

    // Generate a random-looking array using a simple LCG
    const int N = 1000;
    vector<int> base(N);
    int seed = 42;
    for (int i = 0; i < N; i++) {
        seed = (seed * 1103515245 + 12345) & 0x7fffffff;
        base[i] = seed % 10000;
    }

    // Copy for each sort so they all sort the same input
    vector<int> c1 = base, c2 = base, c3 = base;

    SortStats s1 = selectionSortCounted(c1);
    SortStats s2 = insertionSortCounted(c2);
    SortStats s3 = shellSortCounted(c3);

    cout << "  " << left << setw(18) << "Algorithm"
         << right << setw(12) << "Compares"
         << setw(12) << "Swaps" << "\n";
    cout << "  " << string(42, '-') << "\n";
    cout << "  " << left << setw(18) << "Selection Sort"
         << right << setw(12) << s1.compares
         << setw(12) << s1.swaps << "\n";
    cout << "  " << left << setw(18) << "Insertion Sort"
         << right << setw(12) << s2.compares
         << setw(12) << s2.swaps << "\n";
    cout << "  " << left << setw(18) << "Shellsort"
         << right << setw(12) << s3.compares
         << setw(12) << s3.swaps << "\n";

    cout << "\n  Key observations:\n";
    cout << "  - Selection sort always does ~n^2/2 compares\n";
    cout << "  - Insertion sort does fewer compares on nearly-sorted data\n";
    cout << "  - Shellsort is dramatically faster on random data\n";

    return 0;
}
```