Lecture 05: Priority Queues and Heapsort

C++ Code Samples — Sedgwick Algorithms Course — lecture-05-samples.cpp

```cpp
// ============================================================================
// Lecture 05: Priority Queues and Heapsort
// Sedgwick Algorithms Course
//
// Topics covered:
//   1. Min-Heap (insert, extractMin, peek)
//   2. Max-Heap using array
//   3. Heapsort algorithm
//   4. Merging K sorted arrays with a priority queue
// ============================================================================

#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>
#include <climits>

using namespace std;

// === SECTION: Min-Heap ===
// A min-heap is a complete binary tree where every parent is <= its children.
// We store it in an array: for node at index i, left child = 2i+1, right = 2i+2, parent = (i-1)/
2.

class MinHeap {
    vector<int> heap;

    void swimUp(int i) {
        // Move element up until heap property is restored
        while (i > 0) {
            int parent = (i - 1) / 2;
            if (heap[i] < heap[parent]) {
                swap(heap[i], heap[parent]);
                i = parent;
            } else {
              break;
            }
        }
    }

    void sinkDown(int i) {
        // Move element down until heap property is restored
        int n = heap.size();
        while (2 * i + 1 < n) {
            int left = 2 * i + 1;
            int right = 2 * i + 2;
            int smallest = i;
            if (left < n && heap[left] < heap[smallest]) smallest = left;
            if (right < n && heap[right] < heap[smallest]) smallest = right;
            if (smallest != i) {
                swap(heap[i], heap[smallest]);
                i = smallest;
            } else {
              break;
            }
        }
    }

public:
    void insert(int val) {
        heap.push_back(val);
        swimUp(heap.size() - 1);
    }

    int peek() const {
        return heap.front();
    }

    int extractMin() {
        int minVal = heap.front();
        heap[0] = heap.back();
        heap.pop_back();
        if (!heap.empty()) sinkDown(0);
        return minVal;
    }

    bool empty() const { return heap.empty(); }
    int size() const { return heap.size(); }

    void printHeap(const string& label) const {
        cout << label << ": [";
        for (int i = 0; i < (int)heap.size(); i++) {
            if (i > 0) cout << ", ";
            cout << heap[i];
        }
        cout << "]" << endl;
    }
};

// === SECTION: Max-Heap ===
// Same idea but every parent is >= its children.

class MaxHeap {
    vector<int> heap;

    void swimUp(int i) {
        while (i > 0) {
            int parent = (i - 1) / 2;
            if (heap[i] > heap[parent]) {
                swap(heap[i], heap[parent]);
                i = parent;
            } else break;
        }
    }

    void sinkDown(int i) {
        int n = heap.size();
        while (2 * i + 1 < n) {
            int left = 2 * i + 1, right = 2 * i + 2, largest = i;
            if (left < n && heap[left] > heap[largest]) largest = left;
            if (right < n && heap[right] > heap[largest]) largest = right;
            if (largest != i) { swap(heap[i], heap[largest]); i = largest; }
            else break;
        }
    }

public:
    void insert(int val) {
        heap.push_back(val);
        swimUp(heap.size() - 1);
    }

    int extractMax() {
        int maxVal = heap.front();
        heap[0] = heap.back();
        heap.pop_back();
        if (!heap.empty()) sinkDown(0);
        return maxVal;
    }

    bool empty() const { return heap.empty(); }

    void printHeap(const string& label) const {
        cout << label << ": [";
        for (int i = 0; i < (int)heap.size(); i++) {
            if (i > 0) cout << ", ";
            cout << heap[i];
        }
        cout << "]" << endl;
    }
};

// === SECTION: Heapsort ===
// Heapsort works in two phases:
//   1. Build a max-heap in-place (bottom-up heap construction)
//   2. Repeatedly extract the max and place it at the end

void heapSinkDown(vector<int>& arr, int i, int n) {
    while (2 * i + 1 < n) {
        int left = 2 * i + 1, right = 2 * i + 2, largest = i;
        if (left < n && arr[left] > arr[largest]) largest = left;
        if (right < n && arr[right] > arr[largest]) largest = right;
        if (largest != i) { swap(arr[i], arr[largest]); i = largest; }
        else break;
    }
}

void heapsort(vector<int>& arr) {
    int n = arr.size();

    // Phase 1: Build max-heap (start from last non-leaf, go up)
    for (int i = n / 2 - 1; i >= 0; i--) {
        heapSinkDown(arr, i, n);
    }

    // Phase 2: Extract max repeatedly
    for (int i = n - 1; i > 0; i--) {
        swap(arr[0], arr[i]);         // Move current max to sorted position
        heapSinkDown(arr, 0, i);      // Restore heap on the reduced array
    }
}

void printArray(const vector<int>& arr, const string& label) {
    cout << label << ": [";
    for (int i = 0; i < (int)arr.size(); i++) {
        if (i > 0) cout << ", ";
        cout << arr[i];
    }
    cout << "]" << endl;
}

// === SECTION: Merge K Sorted Arrays ===
// Use a min-heap to efficiently merge K sorted arrays.
// We push one element from each array, then repeatedly extract the min
// and push the next element from that same array.
// Time: O(N log K) where N is total elements.

struct HeapEntry {
    int val;
```

```cpp
    int arrayIdx;    // which array this came from
    int elemIdx;     // index within that array
};

vector<int> mergeKSortedArrays(const vector<vector<int>>& arrays) {
    // Custom min-heap using a vector of HeapEntry
    auto cmp = [](const HeapEntry& a, const HeapEntry& b) {
        return a.val > b.val; // min-heap via greater comparator
    };
    vector<HeapEntry> heap;

    auto push = [&](HeapEntry e) {
        heap.push_back(e);
        // swim up
        int i = heap.size() - 1;
        while (i > 0) {
            int p = (i - 1) / 2;
            if (cmp(heap[p], heap[i])) { swap(heap[p], heap[i]); i = p; }
            else break;
        }
    };

    auto pop = [&]() -> HeapEntry {
        HeapEntry top = heap[0];
        heap[0] = heap.back();
        heap.pop_back();
        // sink down
        int i = 0, n = heap.size();
        while (2 * i + 1 < n) {
            int l = 2 * i + 1, r = 2 * i + 2, sm = i;
            if (l < n && cmp(heap[sm], heap[l])) sm = l;
            if (r < n && cmp(heap[sm], heap[r])) sm = r;
            if (sm != i) { swap(heap[i], heap[sm]); i = sm; }
            else break;
        }
        return top;
    };

    // Initialize: push first element from each array
    for (int i = 0; i < (int)arrays.size(); i++) {
        if (!arrays[i].empty()) {
            push({arrays[i][0], i, 0});
        }
    }

    vector<int> result;
    while (!heap.empty()) {
        HeapEntry top = pop();
        result.push_back(top.val);
        // Push next element from the same array
        if (top.elemIdx + 1 < (int)arrays[top.arrayIdx].size()) {
            push({arrays[top.arrayIdx][top.elemIdx + 1], top.arrayIdx, top.elemIdx + 1});
        }
    }
    return result;
}

// === MAIN ===

int main() {
    cout << "========================================" << endl;
    cout << " Lecture 05: Priority Queues & Heapsort" << endl;
    cout << "========================================" << endl;
    // --- Demo 1: Min-Heap ---
    cout << "\n--- Min-Heap Demo ---" << endl;
    MinHeap minH;
    int vals[] = {15, 10, 20, 5, 8, 25, 3};
    for (int v : vals) {
        minH.insert(v);
        minH.printHeap("  After insert " + to_string(v));
    }
    cout << "  Peek (min): " << minH.peek() << endl;
    cout << "  Extract sequence: ";
    while (!minH.empty()) cout << minH.extractMin() << " ";
    cout << endl;

    // --- Demo 2: Max-Heap ---
    cout << "\n--- Max-Heap Demo ---" << endl;
    MaxHeap maxH;
    for (int v : vals) {
        maxH.insert(v);
        maxH.printHeap("  After insert " + to_string(v));
    }
    cout << "  Extract sequence: ";
    while (!maxH.empty()) cout << maxH.extractMax() << " ";
    cout << endl;

    // --- Demo 3: Heapsort ---
    cout << "\n--- Heapsort Demo ---" << endl;
    vector<int> arr = {38, 27, 43, 3, 9, 82, 10};
    printArray(arr, "  Before");
    heapsort(arr);
    printArray(arr, "  After ");

    // --- Demo 4: Merge K Sorted Arrays ---
    cout << "\n--- Merge K Sorted Arrays Demo ---" << endl;
    vector<vector<int>> sortedArrays = {
        {1, 5, 9, 21},
        {2, 3, 7, 12},
        {4, 8, 14, 17}
    };
    for (int i = 0; i < (int)sortedArrays.size(); i++) {
        printArray(sortedArrays[i], "  Array " + to_string(i));
    }
    vector<int> merged = mergeKSortedArrays(sortedArrays);
    printArray(merged, "  Merged");

    return 0;
}
```