ADT-SEM

Filter and Rebuild Min-Heap Based on Threshold:

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

int main() {

    int n;
    cin >> n;

    vector<int> heap(n);
    for (int i = 0; i < n; i++) {
        cin >> heap[i];
    }

    make_heap(heap.begin(), heap.end(), greater<int>());

    int threshold = 2 * heap.front();

    vector<int> remaining;
    for (int x : heap) {
        if (x >= threshold) {
            remaining.push_back(x);
        }
    }

    make_heap(remaining.begin(), remaining.end(), greater<int>());

    for (int x : remaining) {
        cout << x << " ";
    }

    return 0;
}
```

Program Name: Min-Heap Parcel Weight Manager:

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

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

    int sum = 0, count = 0;
    vector<int> heap;

    for (int i = 0; i < n; i++) {
        int w;
        cin >> w;

        if (w > 0) {
            heap.push_back(w);
            push_heap(heap.begin(), heap.end(), greater<int>());
            sum += w;
            count++;
        }
    }

    if (count == 0) {
        cout << "No Valid Weight";
        return 0;
    }

    for (int x : heap) {
        cout << x << " ";
    }
    cout << endl;

    double avg = double(sum) / count;
    cout << fixed << setprecision(2) << avg;

    return 0;
}
```

Max Heap Insertion using Fibonacci Gems:

```
#include<iostream>
#include<vector>
using namespace std;

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

    vector<long long> gems;
    if(n>=1) gems.push_back(2);
    if(n>=2) gems.push_back(3);

    for(int i=2; i<n; i++){
        gems.push_back(gems[i-2] + gems[i-1]);
    }

    vector<long long> heap;
    for(int j=0; j<n; j++){
        long long val = gems[j];
        heap.push_back(val);

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

        cout<<"Insert "<<val<<": ";
        for(long long m : heap){
            cout<<m<<" ";
        }
        cout<<endl;
    }

    return 0;
}
```

Max Heap Employee ID Manager:

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

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

    vector<int> heap;
    int total = 0;

    for(int x = 1; x <= n; x++){
        heap.push_back(x);
        total += x;

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

    for(int x : heap) cout << x << " ";
    cout << endl << total;

    return 0;
}
```

Kth Best-Selling Product Finder (Using Heap Sort Logic):

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
```

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

    vector<int> heap;

    for (int i = 0; i < n; i++) {
        int a;
        cin >> a;
        heap.push_back(a);
    }

    // Sort in descending order (Heap Sort logic idea)
    sort(heap.begin(), heap.end(), greater<int>());

    int k;
    cin >> k;

    cout << heap[k - 1];

    return 0;
}
```

Lexicographic Word Sorter (Using Heap Sort Logic):

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

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

    string text;
    vector<string> heap;

    for (int i = 0; i < n; i++) {
        cin >> text;
        heap.push_back(text);
    }

    sort(heap.begin(), heap.end());
```

```
    for (int i = 0; i < n; i++) {
        cout << heap[i] << " ";
    }

    return 0;
}
```

Pattern Found or Not Found:

```
#include <iostream>
#include <string>
using namespace std;

int main() {
    string text;
    string pattern;

    getline(cin, text);
    getline(cin, pattern);

    bool foundMatch = false;
    size_t pos = text.find(pattern, 0);

    while (pos != string::npos) {
        cout << "Pattern found at index " << pos << endl;
        foundMatch = true;
        pos = text.find(pattern, pos + 1);
    }

    if (!foundMatch) {
        cout << "Pattern not found in the text." << endl;
    }

    return 0;
}
```

Activity Selection Problem:

```
n = int(input())
start = list(map(int, input().split()))
finish = list(map(int, input().split()))

activities = [(finish[i], start[i], i) for i in range(n)]
activities.sort()  # Sort by finish time

selected = []
last_finish = -1

for f, s, idx in activities:
    if s >= last_finish:
        selected.append(idx)
        last_finish = f

selected.sort()
print(*selected)
```

Knapsack:

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <iomanip>
using namespace std;

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

    vector<int> value(n), weight(n);
    cout << "Enter value and weight of each item:\n";
    for (int i = 0; i < n; i++)
        cin >> value[i] >> weight[i];

    int capacity;
    cout << "Enter total capacity: ";
    cin >> capacity;

    // Step 1: Calculate ratio (value/weight)
```

```
    vector<pair<double, int>> ratio(n);
    for (int i = 0; i < n; i++)
        ratio[i] = { (double)value[i] / weight[i], i };

    // Step 2: Sort items in descending order of ratio
    sort(ratio.rbegin(), ratio.rend());

    double totalValue = 0.0;

    // Step 3: Select items
    for (int i = 0; i < n; i++) {
        int idx = ratio[i].second; // actual item index

        if (weight[idx] <= capacity) {
            // Take the full item
            totalValue += value[idx];
            capacity -= weight[idx];
        } else {
            // Take fractional part
            totalValue += value[idx] * ((double)capacity / weight[idx]);
            break;
        }
    }

    cout << fixed << setprecision(2);
    cout << "The maximum value of the current list is:\n" << totalValue <<
endl;

    return 0;
}
```