Esau Hervert
eah170630

Assignment 3

CSE 5343
July 5, 2019

Problem

Implement heap sort.

Write a program that would sort a list of numbers in ascending order using heap sort.

1. Make an array of 15 numbers. The array size should be 20, but the numbers are only 15. The numbers are in random order.
2. Print the numbers from index 0 to index 15. Index 0 stores the actual 15 (numbers in the array).
3. Make a heap of the 15 numbers.
4. Print the numbers in the array from index 0 onwards. It should show the heap.
5. Sort the number using heap sort.
6. Print the numbers in array from index 0.

Submit the code and submit the screenshots.

Code

Code 1: Heap_Functions.h

```
#ifndef Heap_Functions
#define Heap_Functions

#include <iostream>

using namespace std;

class HeapTwenty {
public:
        // This represents the empty heap.
        // Since this heap only takes in positive
        // numbers, we have empty nodes be represented
        // as -1.
        int A[20] = { 0,-1,-1,-1,-1,-1,-1,-1,-1,-1,
                -1,-1,-1,-1,-1,-1,-1,-1,-1,-1 };

        // This function will add nodes to the heap s.t.
        // the structure is a heap. Note that it is not
        // yet a heap, it just has the structure.
        void add(int);
```

```
        // This will print the array that is not yet
        // a heap.
        void printNH();

        // This will print the heap.
        void print();

        // This prints the total array, including -1's.
        void printT();

        // This will be used to print the sorted array in
        // Floyd's algorithm.
        void printA(int, int);

        // This function will use Floyd's algorithm which
        // takes our binary tree with the heap structure
        // and makes it into a tree.
        void Heapify();

        // This function will sort the heap.
        void sort();

        // This function will print the sorted array.
        void printsort();
};

void HeapTwenty::add(int v) {
        int i = 1;
        // We traverse the heap and add the value
        // at the next available node.
        while (A[i] != -1) {
                i++;
        }
        // If the heap is full, we return.
        if (i == 20) {
                cout << "Heap Full!" << endl;
                return;
        }

        A[i] = v; // Add the value at the node.

        //Increment the size of the heap.
        A[0]++;
}

void HeapTwenty::printNH() {
        cout << "The Array is: ";
        int i = 0;
        while (A[i] != -1) {
                cout << A[i] << " ";
```

```
                i++;
        }
        cout << endl;
}

void HeapTwenty::print() {
        cout << "Heap Size: " << A[0] << endl;
        cout << "Heap: ";
        for (int i = 0; i < A[0]; i++) {
                cout << A[i + 1] << " ";
        }
        cout << endl;
}

void HeapTwenty::printT() {
        cout << "The Total Array is: ";
        for (int i = 0; i < 20; i++) {
                cout << A[i] << " ";
        }
        cout << endl;
}
void HeapTwenty::printA(int a, int n) {
        for (int i = a + 1; i <= n; i++) {
                cout << A[i] << " ";
        }
        cout << endl;
}

// This function swaps two values in a array.
void swap(int *A, int i, int j) {
        int temp = A[i];
        A[i] = A[j];
        A[j] = temp;
}

// Given an array and three indexes, this function
// will return the index with the largest value.
int Largest(int *A, int i, int l, int r) {
        int I = A[i];

        // For the case where we have that we don't
        // have a left or right child, we make the value
        // equal to zero so that the max will skip it.
        int L = 0, R = 0;
        if (l != -1) { L = A[l]; }
        if (r != -1) { R = A[r]; }

        if (L > R) {
                if (I >= L) { return i; }
                else { return l; }
```

```
        }
        else {
                if (I >= R) { return i; }
                else { return r; }
        }
}

void MaxHeapify(int *A, int i, int n) {
        // We find the left child. If there is no
        // left child, we set the value equal to
        // -1 to signify no left child.
        int left = -1;
        if (2 * i <= n) { left = 2 * i; }

        // Same logic as left child.
        int right = -1;
        if (2 * i + 1 <= n) { right = 2 * i + 1; }

        // index with the largest value.
        int largest = Largest(A, i, left, right);

        // If it turns out that a child has a larger
        // value, then we swap the values.
        if (largest != i) {
                swap(A, i, largest);
                // We percolate down and swap as needed.
                MaxHeapify(A, largest, n);
        }

        // Once we are done going down a node and have
        // finished with it, we then move on to the
        // preceding node until we reach the top.
        if (i != 1) {
                MaxHeapify(A, i - 1, n);
        }
}

// So we can call the function easily.
void HeapTwenty::Heapify() {
        MaxHeapify(A, A[0] / 2, A[0]);
}

// This algorithm will split the heap into two arrays:
//      - The second array will contain the values going from
//        smallest to largest.
//      - The first array will contain the remaining unsorted
//        heap.
// This algorithm will pick out the largest element in the
// heap and move it to the sorted array, and then fix the
// remaining heap. Then it will continue removing and fixing
```

```
// until there is only one element in the heap left. At this
// point we have sorted the heap so we are done.
void HeapTwenty::sort() {
        int n = A[0];
        cout << "Now we sort the Heap:" << endl;
        for (int i = A[0]; i > 2; i--) {
                swap(A, A[0], 1);
                A[0]--;
                MaxHeapify(A, A[0] / 2, A[0]);
                print(); // Show the remaining heap.
                // Show the sorted array.
                cout << "Sorted Array: ";
                printA(A[0], n);
                cout << endl;
        }
        swap(A, A[0], 1);
        A[0]--;
        print(); // Show the ending heap.
        // Show the sorted array.
        cout << "Sorted Array: ";
        printA(A[0], n);
        cout << endl;
}

void HeapTwenty::printsort() {
        cout << "The Final Sorted Array is: ";
        int i = 1;
        while (A[i] != -1) {
                cout << A[i] << " ";
                i++;
        }
        cout << endl;
}

#endif Heap_Functions
```

Code 2: Assignment_3_Heaps.cpp

```
#include "Heap_Functions.h"

#include <iostream>

using namespace std;

int main() {
        // Here is the heap.
        HeapTwenty H;

        // We add numbers 1-15 in random order.
        H.add(1);
        H.add(13);
```

```
        H.add(10);
        H.add(3);
        H.add(6);
        H.add(14);
        H.add(7);
        H.add(15);
        H.add(12);
        H.add(4);
        H.add(9);
        H.add(5);
        H.add(2);
        H.add(11);
        H.add(8);

        H.printNH(); // Print the unheaped array.
        H.printT();
        cout << endl;

        // Make the heap and print it.
        H.Heapify();
        H.print();
        H.printT();
        cout << endl;

        // Now, sort the heap and print.
        H.sort();
        H.printsort();

        // Print the resulting array.
        cout << endl;
        H.printNH();
        H.printT();
}
```

Results

I did all the steps in one run. Here is the results.

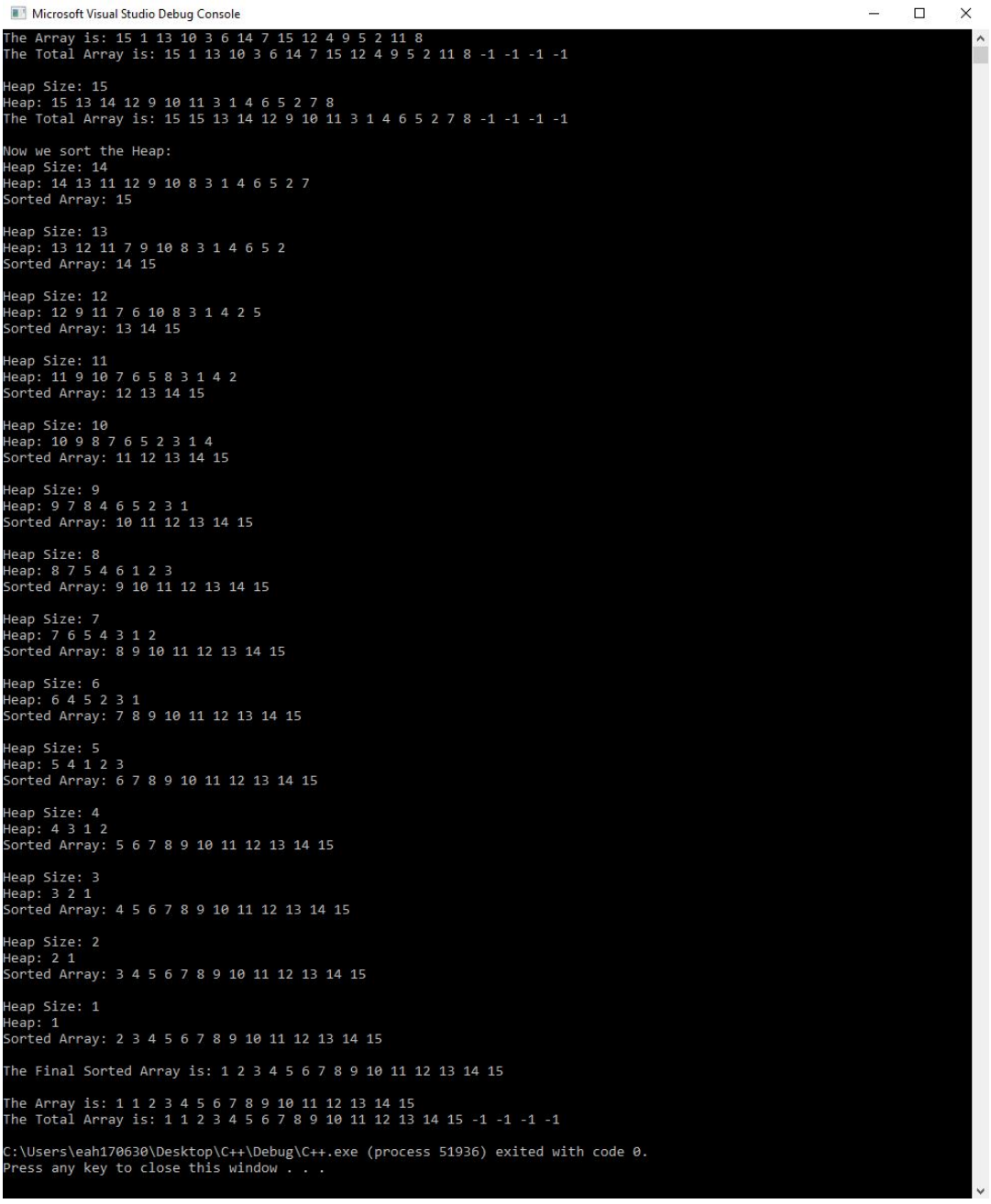

Figure 1: Results