CSE 330 Homework Assignment 4 , Spring 2018

Instructor: Kerstin Voigt

(Total number of points: 15)

In this assignment, you will be comparing the algorithm performances of TreeSort and HeapSort. Both algorithm have the data values stored in a binary tree structure; one uses a binary search tree, the other uses a binary heap. Both algorithms share an average computational complexity of O(N log N) in the “best” and “average” cases. Yet they are different … as you will demonstrate in this assignment.

Start by obtaining copies of the following files (from Blackboard):

- Random.h, BST_HW4.h, BinaryHeap_HW4.h – use exactly these files, exactly as given; do not modify.
- **TreeSortHeapSort.h – with code to be completed**

In file TreesortHeapSort.h, you are given the complete implementation of TreeSort:

```
template <typename C>
void TreeSort(vector<C>& data, int& comps)
{
  CLUMSY_COUNT = 0;
  BinarySearchTree<C> bst;

  // enter all N many many data items into the bst;
  // loops iterates  N times; each iteration has cost O(log N);
  // thus: cost of this loop is O(N log N);

  for (int i = 0; i < data.size(); i++)
    {
      bst.insert(data[i]); // average cost of insertion O(log N)
    }

  // iterate through the bst, access each data item in turn and
  // copy it back into the ith field in the vector;
  // cost: O(N)

  int i = 0;
  typename BinarySearchTree<C>::iterator itr = bst.begin();
  for (; itr != bst.end(); ++itr)
    {
      data[i] = *itr;
      i++;
    }                      // total cost: O(N log N) + O(N) = O(N log N)
  comps = CLUMSY_COUNT;
}
```

Examine the TreeSort function in detail! Notice its two reference arguments: the vector, to be modified (sorted) by the function body and the output parameter ‘int & comps’ which is to pass outward to total number of data item comparisons (a measure cost) that were made while sorting the vector. The number of comparisons is accumulated in a global external variable CLUMSY_COUNT. There are better, less risky ways to maintain a cumulative measure of computational effort, yet, the global variable solutions was easiest to insert with our existing code (in the _HW4.h files). Let’s just day, it works well enough …

The TreeSort algorithm, along with explanations of its computational complexity, is given in the inserted comments. Please read these and make sure you understand them.

Homework Task 1: (5 pts) Complete the implementation of the HeapSort function, following the example of TreeSort. The binary heap data structure does not have an iterator, but based on your knowledge of the BinaryHeap interface, you should find a suitable alternative for what needs to happen.

Homework Task 2: (5 pts) Test your completed code with the below int main() in a file THSort_HW4.cpp. This function will sort vectors of integers of varying sizes (10 to 50 items) first with TreeSort, then with HeapSort. For each vector size, 25 random vectors are thus sorted. The average number of comparisons for TreeSort and HeapSort for each batch of 25 vectors is recorded.

```
// THSort_HW4.cpp
// comparing the performance of TreeSort vs. HeapSort

#include <iostream>
#include <vector>
#include <cmath>
#include <cassert>
#include "Random.h"
#include "TreeSortHeapSort.h"
using namespace std;

int main()
{
  vector<int> tosort1;
  vector<int> comps1;
  vector<int> comps2;
  int sum1;
  int sum2;
  int numdata = 25;

  rand_seed();

  for (numdata = 10; numdata <= 50; numdata+=5)
    {
      sum1 = 0;
      sum2 = 0;
      for (int i = 1; i <= 25; i++)
      {
        tosort1 = rand_int_vector(numdata,1,1000);
        vector<int> tosort2(tosort1);
        int k1,k2;
```

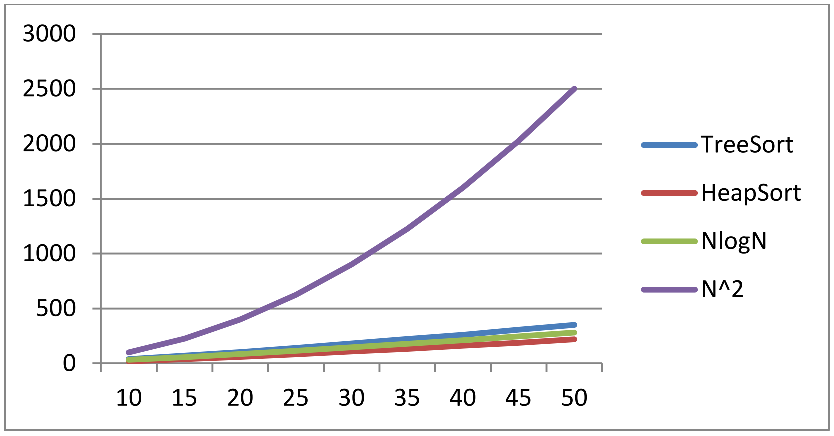
```
          TreeSort(tosort1,k1);
          assert(tosort1.size() == numdata);

          HeapSort(tosort2,k2);
          assert(tosort2.size() == numdata);

          comps1.push_back(k1);
          comps2.push_back(k2);
          sum1 += k1;
          sum2 += k2;
        }

        cout << endl;
        double nlogn = numdata* log10(numdata)/log10(2);
        int nn = numdata * numdata;
        double avg1 = (double)sum1 / 25;
        double avg2 = (double)sum2 / 25;

        cout << numdata << ", " << (int)avg1 << ", "
           << (int)avg2 << ", " << (int)nlogn
           << ", " << nn << endl;
      }

  return  0;
}
```

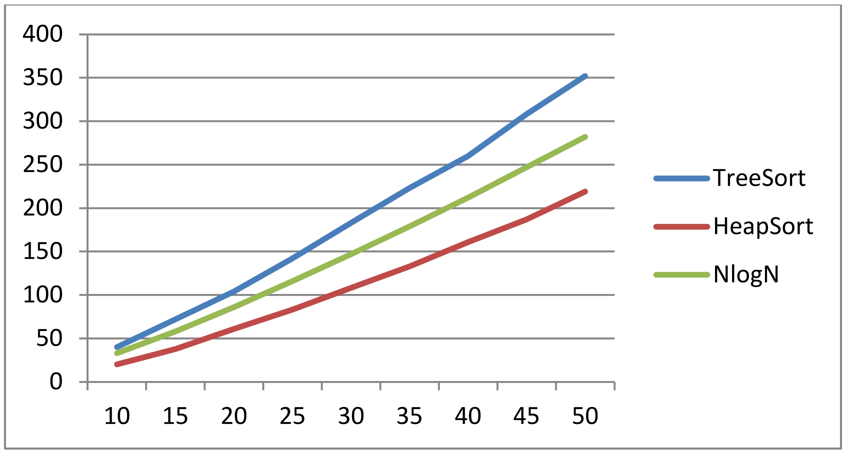
When your code runs correctly, it will provide output that looks like shown below (your numbers will be different):

```
10, 41, 19, 33, 100
15, 72, 37, 58, 225
20, 104, 61, 86, 400
25, 142, 83, 116, 625
30, 186, 106, 147, 900
35, 221, 134, 179, 1225
40, 266, 161, 212, 1600
45, 305, 190, 247, 2025
50, 348, 219, 282, 2500
```

The first number is the vector size (e.g., 10), the second number is the average number of comparisons for 25 runs for TreeSort (e.g., 41), the third number is the average number of comparisons for HeapSort (e.g., 19) the fourth number is the theoretical number for N log(N), cast as an integer value (e.g., 33), and the fifth number is the theoretical number for N^2 (e.g., 100).

Homework Task 3: (5 pts) As we have done in previous labs and assignments, plot the data set that you obtain for your own runs of random data. For example, the above data yield





Submit on Tuesday, June 12, 2018, at the beginning of the Final Exam: (1) A hardcopy of your completed file TreeSortHeapSort.h, (2) a typescript or screenshot that demonstrates the successful compilation and running of your int main(); it should contain the set of performance data to be plotted, (3) 1-2 data plots of the performances of TreeSort and HeapSort, relative to each other and the theoretical values of N log(N) and N^2.

If you do not succeed at producing code that compiles and runs, submit your best effort at (1), and for (2) a typescript with your compiler or run-time errors.