ECS 60 **Programming Assignment #3** **Fall 2017**

Due: Monday, November 6th, Write-up by 4:00 P.M., Program at 11:59 pm to p3 in the cs60 account.
Filenames: authors.csv, BSTX.cpp, and BST2.h.
Format of authors.csv: author1_emai,author1_last_name,author1_first_name
author2_email,author2_last_name,author2_first_name

#1 (40 points) I have written an extended version of the timetest.cpp from programming assignment #1, called timetest3.cpp. Both the source code, and PC executable are available in ~ssdavis/60/p3. I have also made the data files five times larger than in P1, i.e., 2.5 million operations instead of 500,000. For this question, you are to write an extensive paper (4-7 pages typed double spaced and no more, not including the tables) that compares the performance of eight new ADTs and SkipList for the four files, File1.dat, File2.dat, File3.dat, and File4.dat. You need to run each new ADT only once on each file. If an ADT does not finish within five minutes, then note that and kill the program. For BTree try M = 3 with L = 1; M = 3 with L = 200; M = 1000 with L = 2; and M = 1000 with L = 200. For the Quadratic Probing Hash try load factors of 2, 1, 0.5, 0.25, and 0.1. For the Separate Chaining Hash try load factors of 0.5, 1, 10, 100, and 1000. To set the load factor for hash tables in timetest3, you supply the size of the original table. For example, for File1.dat to have a load factor of 5 you would enter 2,500,000 / 5 = 500000 for the table size.

Each person must write and TYPE their own paper. There is to be NO group work on this paper. There are two ways to organize your paper. One way is by dealing with the results from each file separately: 1) File1.dat, 2) File2.dat, 3) File3.dat, 4) File4.dat, and 5) File2.dat vs. File3.dat. If there are differences between how a specific ADT performs on File2.dat and File3.dat explain the differences in the last section. The other way is to deal with each ADT separately.

In any case, make sure you compare the trees (including the BTree using M = 3 with L = 1) to each other and to skip list. For BTrees, explain the performance in terms of M and L. You should also compare the hash tables to each other somewhere in your paper. For the hashing ADTs, you should also discuss the affects of different load factors on their performance with each file. Compare the performance of the Quadratic Probing hash with QuadraticProbingPtr in a separate paragraph. You should determine the big-O's for each ADT for each file; this should include five big-O values: 1) individual insertion; 2) individual deletion; 3) entire series of insertions; 4) entire series of deletions; and 5) entire file. Use table(s) to provide the run times and the big-O's.

Do not waste space saying what happened. The tables show that. Spend your time explaining what caused the times that were they were relative to each other. Always try to explain any anomalies you come upon. For example, for most ADTs, you should clearly explain why there are different times for the three deleting files. While a quick sentence explaining the source of a big-O is enough for ADT-File combinations that perform as expected, you should devote more space to the unexpected values.

Five points of your grade will be based on grammar and style. If you use Microsoft Word, go to the menu Tools:Options:Spelling &Grammar:Writing Style and set it to Formal. This setting will catch almost all errors, including the use of passive voice.

2. (1.5 hours, 10 points) Given the recursive nature of a binary tree, a good strategy for writing a C++ function that operates on a binary tree is often first to write a recursive definition of the task. For example, suppose that the task is to count the number of nodes in a binary tree. An appropriate recursive definition for the number of nodes C(T) of a binary tree T is:

$$C(T) = \begin{cases} 0 & \text{if T is the empty tree} \\ C(\text{left subtree of T}) + C(\text{right subtree of T}) + 1 & \text{otherwise.} \end{cases}$$

Given such a recursive definition, a C++ implementation is often straightforward.

Each team should implement the definitions in C++. For simplicity, assume that a tree data item is a single integer and that there are no duplicates. I've included two files in my p3 subdirectory that will provide you a starting point: BSTX.cpp, and BSTX.h. You should not need to change BSTX.h. You will need to add the appropriate functions definitions to the bottom of BSTX.cpp. You need to handin only BSTX.cpp.

I've included BSTXtest.cpp in the p3 subdirectory that will test your implementation.

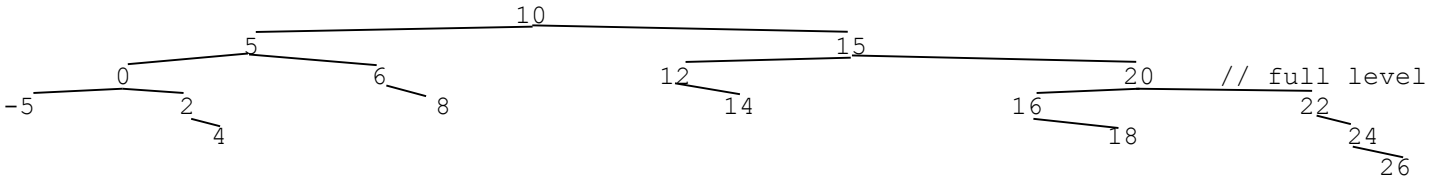
a) Computer the height of a tree. int height() const
b) Find the sum of the elements. int sum() const
c) Determine whether one item is an ancestor of another (that is, whether one item is in the subtree rooted at the other item). bool isAncestor(const Object x, const Object possibleAncestor) const

d) Determine the height of the deepest level that is full or, equivalently, has the maximum number of nodes for that level. int highestFull() const.
e) Determine the depth of the deepest level that is full or, equivalently, has the maximum number of nodes for that level. int deepestFull() const.

```
[ssdavis@lect1 private]$ g++ -o BSTX.out BSTXtest.cpp
[ssdavis@lect1 private]$ cat BSTXtest.cpp

#include <iostream>
#include "BSTX.h"
using namespace std;

/*  The tree
```




```
 */

int main()
{
  int i;
  BinarySearchTreeX<int> tree(-1);

  for(i = 10; i <= 20; i += 5)
  {
    tree.insert(i);
    tree.insert(15 - i);
  }
  for(i = 2; i < 28; i += 2)
    if(i % 5 != 0)
      tree.insert(i);
  cout << "Height: " << tree.height() << endl;
  cout << "Sum: " << tree.sum() << endl;

  if(tree.isAncestor(4,5))
    cout << "5 is an ancestor of 4\n";
  else
    cout << "5 is not an ancestor of 4\n";

  if(tree.isAncestor(5,4))
    cout << "4 is an ancestor of 5\n";
  else
    cout << "4 is not an ancestor of 5\n";

  cout << "Highest Full: " << tree.highestFull() << endl;
  cout << "Deepest Full: " << tree.deepestFull() << endl;
  return 0;
} // main())

[ssdavis@lect1 private]$ BSTX.out
Height: 5
Sum: 197
5 is an ancestor of 4
4 is not an ancestor of 5
Highest Full: 3
Deepest Full: 2
[ssdavis@lect1 private]$
```