ECS 60 Programming Assignment #3 Winter 2015

Due: Wednesday, February 11th, Write-up by 4:00 P.M., Program at 11:59 pm to p3 in the cs60 account.
Filenames: authors.txt, QuadraticProbingPtr.cpp, QuadraticProbingPtr.h, BST2.cpp, and BST2.h.
Format of authors.txt: author1_email 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. 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 250000 / 5 = 50000 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 (10 points, 15 minutes) File names: QuadraticProbingPtr.cpp and QuadraticProbingPtr.h.

Weiss' QuadraticProbing class stores actual objects in the vector. Because at least half the vector must be empty, if the objects are more than eight bytes in size this wastes space, and may even make the table too large to fit in RAM. Based on Weiss' implementation, create a new template class, QuadraticPtrHashTable that stores pointers to objects in the vector. The new class will now have "vector<const HashedObj*> array;". You will need to add a destructor to the class to delete the valid pointers. Rather than use Weiss' HashEntry struct you can use NULL and a pointer to the ITEM_NOT_FOUND object to indicate EMPTY and DELETED respectively. You should add a paragraph in your timetest write-up that compares the performance of this new hash table with that of the original QuadraticProbing hash table.

I suggest that you follow these steps:

1. Copy QuadraticProbing files to QuadraticProbingPtr files.
2. Use a global search and replace to convert the QuadraticHashTable code to QuadraticPtrHashTable code. At this point, only the names have been changed.
3. Alter timetest3.cpp to use QuadraticProbingPtr.h and QuadtraticPtrHashTable, and see if it compiles before you make any other changes.
4. Change the vector to vector<const HashedObj*> array; and eliminate the enum and struct from the .h file.
5. Work through the .cpp file line by line making changes wherever necessary. I found that I used the isActive() method many places.
6. Compile the class, and correct errors.
7. Add the destructor to the .h and .cpp files and re-compile.

#3 (10 points, 15 minutes) File names: BST2.cpp, and BST2.h. You are to implement the printRange(w,y) function for a BST that prints all of the values in the tree between x and y, inclusive. You will find in ~ssdavis/60/p3 the BST2.cpp, BST2.h, PR1.dat, PR2.dat, the driver program rangetest.cpp, my rangetest.out, the dat file creator source code rangecreate.cpp, and rangecreate.out. Note that BST2 allows duplicates in the tree. You may add any functions you need necessary to BST2.h and BST.cpp. Your function(s) should be as efficient as possible. A simple traversal of the whole tree testing each value is unacceptable.

```
void RunBST(char *filename)
{
  BinarySearchTree <int> tree(-1);
  ifstream inf(filename);
  char command, s[512];
  int i, j;

  inf.getline(s,512);

  while(inf >> command >> i)
  {
    if (command == 'i')
      tree.insert(i);
    else if (command == 'd')
      tree.remove(i);
    else
    {
      inf >> j;
      tree.printRange(i,j);
    }
  } //while
} // RunBST()
```

```
[ssdavis@lect1 p3]$ cat PR1.dat
Has a series of 20 insertions in order from 0 to 19 followed by 5 deletions 15 to 20,
followed by 5 range requests
i0 i1 i2 i3 i4 i5 i6 i7 i8 i9 i10 i11 i12 i13 i14 i15 i16 i17 i18 i19 d19 d18 d17 d16 d15
r7 8 r6 9 r5 10 r4 11 r3 12

[ssdavis@lect1 p3]$ rangetest.out
Filename >> PR1.dat
7 8
6 7 8 9
5 6 7 8 9 10
4 5 6 7 8 9 10 11
3 4 5 6 7 8 9 10 11 12

[ssdavis@lect1 p3]$ cat PR2.dat
Has a series of 20 random insertions from 0 to 99 followed by 10 random range requests
i83 i86 i77 i15 i93 i35 i86 i92 i49 i21 i62 i27 i90 i59 i63 i26 i40 i26 i72 i36 r11 68
r67 29 r82 30 r62 23 r67 35 r29 2 r22 58 r69 67 r93 56 r11 42

[ssdavis@lect1 p3]$ rangetest.out
Filename >> PR2.dat
15 21 26 26 27 35 36 40 49 59 62 63
35 36 40 49 59 62 63
35 36 40 49 59 62 63 72 77
26 26 27 35 36 40 49 59 62
35 36 40 49 59 62 63
15 21 26 26 27
26 26 27 35 36 40 49

59 62 63 72 77 83 86 86 90 92 93
15 21 26 26 27 35 36 40
[ssdavis@lect1 p3]$
```