ECS 60 **Programming Assignment #3 (55 points)** **Spring 2014**

Due: Wednesday, May 7th, Write-up by 4:00 P.M., Program at 11:59 pm to p3 in the cs60 account.

#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 nine new ADTs 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. Among other things, you will lose one point for each time you use passive voice. 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 (15 points, 70 minutes) Name your files huffman.cpp, and BinaryTree.h. Your program will read a file and then write to the screen the Huffman encoding for each of the characters in the file. Huffman codes permit a simple method to compress a message. A description of the encoding method is described on pages 413-419 in your text. Note that for this assignment, unlike Weiss' inconsistent example, if |T1| < |T2| and T1 and T2 are merged, then T1 will be the left child (using a '0') of the new tree. Many files tested will have occurrences where |T1| == |T2|, e.g. AVLTree.cpp. In this case, there will be more than one proper answer. ssdavisHuffman.txt has no such indeterminate cases.

The filename will be given as the command line parameter. The output should be the result of an inorder traversal of the final tree, with internal nodes not shown. Your BinaryTree.h file should contain both the definition and implementation of a BinaryTree template. You need only define those operations needed for this assignment. You must implement a public printTree() method for your BinaryTree template class that will be called from main(). You may create other class(es) as you see fit. huffman.cpp should only contain main(), and your additional class(es). You may not use Weiss or STL files for this program, except BinaryHeap.

As usual, your format must match mine.

Hints: If you look at the definition of a tree, you'll see that you do not need a BinaryNode class. A private printTree can take parameters. I used a BinaryTreePtr class in my BinaryHeap to preserve the BinaryTree pointers.

```
[@lect1 p3]$ cat weissHuffman.txt
aaaaaaaaaaeeeeeeeeeeeeeeeiiiiiiiiiiii          ssstttt
[@lect1 p3]$
[@lect1 p3]$ huffman.out weissHuffman.txt
i   12 00
    13 01
e   15 10
t    4 1100

    1 11010
s    3 11011
a   10 111
[@lect1 p3]$
[@lect1 p3]$ cat ssdavisHuffman.txt
A
BB
CCCC
DDDDD
EEEEEE
FFFFFFFF
GGGGGGGGG
HHHHHHHHHH
JJJJJJJJJJJJJ
KKKKKKKKKKKKKK
LLLLLLLLLLLLLLLL
MMMMMMMMMMMMMMMMM
[@lect1 p3]$
[@lect1 p3]$ huffman.out ssdavisHuffman.txt
D    5 0000
E    6 0001

   12 001
J   13 010
K   14 011
A    1 100000
B    2 100001
C    4 10001
F    8 1001
L   16 101
M   17 110
G    9 1110
H   10 1111
[@lect1 p3]$
[@lect1 p3]$ huffman.out AvlTree.cpp
  4687 0
{   40 10000000
x   44 10000001
-   93 1000001
R   22 100001000
w   23 100001001
k   49 10000101
/   96 1000011
o  375 10001
l  382 10010
r  385 10011
e  782 1010

  413 10110
<   97 1011100
&   24 101110100
"    6 10111010100
+    6 10111010101
D   14 1011101011
T   57 10111011
g  108 1011110
C  109 1011111
m  229 110000
c  113 1100010
y   28 110001100
1   30 110001101
.   59 11000111
s  231 110010
b  120 1100110
=   60 11001110
:   61 11001111
f  121 1101000
L   63 11010010
?    3 110100110000
            1 11010011000100
4    1 11010011000101
P    1 11010011000110
~    1 11010011000111
!    8 11010011001
O   16 1101001101
_   16 1101001110
F   18 1101001111
n  259 110101
(  131 1101100
)  131 1101101
h  272 110111
A   68 11100000
N   69 11100001
,   36 111000100
W   18 1110001010
I   18 1110001011
u   73 11100011
E   18 1110010000
M   18 1110010001
U   37 111001001
7    1 11100101000000
S    1 11100101000001
@    1 11100101000010
j    1 11100101000011
3    5 111001010001
#    2 1110010100100
0    3 1110010100101
V    5 111001010011
2   20 1110010101
}   40 111001011
d  152 1110011
*  161 1110100
p  162 1110101
i  329 111011
;   81 11110000
v   82 11110001
>  170 1111001
a  354 111101
t  705 11111
[@lect1 p3]$
```