Test 2: CPS 100

Owen Astrachan

April 3, 1996

Name: ______________________________

Honor code acknowledgement (signature) ______________________________

<table>
<thead>
<tr>
<th>Problem</th>
<th>Value</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem 1</td>
<td>13 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 2</td>
<td>14 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 3</td>
<td>13 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 4</td>
<td>12 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 5</td>
<td>15 pts.</td>
<td></td>
</tr>
<tr>
<td>EXTRA</td>
<td>8 pts.</td>
<td></td>
</tr>
<tr>
<td>TOTAL:</td>
<td>67 pts.</td>
<td></td>
</tr>
</tbody>
</table>

This test has 8 pages, be sure your test has them all. Do NOT spend too much time on one question — remember that this class lasts 50 minutes.

**Declarations** For all problems on this test you can assume the following declaration for a templated tree node is made.

```cpp
template <class Kind>
struct TNode // "standard binary tree declaration"
{
    Kind info;
    TNode * left;
    TNode * right;
    TNode (const Kind & val, TNode * lchild = 0, TNode * rchild = 0)
        : info(val),
          left(lchild),
          right(rchild)
    {};
};
```
PROBLEM 1:  (Short Answer: 13 points)

Three points each, except for last two which are two points

1. Both bubble-sort and selection sort are $O(n^2)$ sorts. Briefly, why is the running time of bubble sort so much slower than selection sort, and why is this difference more pronounced when strings are sorted rather than integers?

2. Linear-probing is one method for resolving collisions when hashing. Explain in one or two sentences one problem with using linear probing to resolve collisions.

3. A database of names will be stored in memory for a computer program. Names will be added occasionally, and deleted occasionally, but the names will need to be printed in alphabetical order very frequently. Why is a balanced binary search tree a better choice for storing the data than a hash table?

4. If $n$ items are stored in an array, beginning at position 1 (the zero-th item isn’t used) what is the complexity, using big-Oh, of turning the $n$ items into a binary-heap with the smallest element at index one ________

5. If you do NOT use a makefile, but run the g++ compiler without specifying a name for the resulting executable (e.g. g++ foo.cc), what will be the name of the executable file ________?
PROBLEM 2:  (Trees 14 points)

Part A 4 points
Give the postorder traversal of the binary search tree below.

```
  p
 /   \
q   u
 / \
 g  r
 / \   / \
b  f  q  s
 / \   / \
  i  w  v  y
 /   /   /   
x  s  m  j
```

Postorder: __________________________________________

Part B 4 points
Add nodes containing x, s, m and j, in that order, to the search tree shown above.

Part C 6 points
If a node with two children in a binary search tree is deleted, it is replaced by its inorder successor, i.e., the node that comes next in an inorder traversal. For example, in the original tree above (before any values are added as in part B), node p (the root) would be replaced by q, and node u would be replaced by node v. Deleting a node with no children is straightforward, the node is simply removed. Deleting a node with one child results in “pulling” up the child, e.g., to delete node g above just replace it by d.

Write the function ReplacePtr below that returns a pointer to the node that will replace tree. ReplacePtr should return 0/NULL if tree has no children, a pointer to tree's only child if there is only one child, and a pointer to the inorder successor of tree if tree has two children.

```c++
template <class Kind>
TNode<Kind> * ReplacePtr(TNode<Kind> * tree)
// postcondition: returns pointer to node that will replace node tree
// if node tree is deleted
{
}
```
PROBLEM 3: (gnitroS Two 13 points)

Part A 3 points
Is the following statement true or false, why (briefly)?
There are absolutely no sorts that have running times better than $O(n \log n)$

Part B 4 points
Give one reason to prefer mergesort to quicksort and also one reason to prefer quicksort to mergesort (be brief).
Part C  6 points

The function DoMerge from sortall.cc is modified (as shown below) to divide a vector into three parts, recursively sort each of the three parts, then merge the parts back together. One line is missing from the function DoMerge. Add the one line to DoMerge.

Write a recurrence relation for the version of mergesort described above and implemented in DoMerge. What is the solution of this recurrence relation, i.e., the big-Oh complexity of the new version of mergesort? You do NOT need to solve the recurrence, a one sentence justification is sufficient (although you may solve the recurrence).

template <class Type>
void Merge(Vector<Type> & a, int left, int mid, int right)
// precondition: a sorted from a[left] ... a[mid] and from
// a[mid+1] to a[right]
// postcondition: a sorted from a[left] ... a[right]

template <class Type>
void DoMerge(Vector<Type> & a, int left, int right)
// postcondition: a[left] <= ... <= a[right]
{
    int onethird = (right - left)/3;     // [left...first] is first interval
    int first = left + onethird;        // [first...second] is second interval
    int second = right - onethird;      // [second...right] is third interval

    if (left < right)
    {
        DoMerge(a, left, first);
        DoMerge(a, first+1, second);
        DoMerge(a, second+1, right);
        Merge(a, left, first, second);
        // one call of Merge missing below
    }
}
PROBLEM 4:  (*Heap o' Trouble* 12 points)

The values stored in the array below form a min-heap.

```
  0  1  2  3  4  5  6  7  8  9  10  11
  4 12  7 20 37 38 27 26 32   
```

Part A  4 points
Draw the complete binary tree that corresponds to the heap in the array above.

Part B  4 points
If 39 is added to the heap, it will be stored at index 10, with no movement of items in the heap. However, if 8 is added to the heap, items will be moved. What two items will be moved if 8 is added to the heap?

Part C  4 points
If a *DeleteMin* operation is performed, the item at the root/index 1 will be removed and data will be moved. Draw either the tree or the array that results from a *DeleteMin* operation in the original heap from part A.
PROBLEM 5:  (HR HuffnStuff 15 points)

A file contains seven different characters (shown below) will be compressed using Huffman coding.

```
trees are a treasure
```

One possible Huffman tree for the characters above is shown below.

```
    (space)
   /       \
  a       e
 /  \    /  \  
 u   t   r   e
```

**Part A  6 points**
Label each non-leaf node in the tree above with the priority/weight given to it as a result of building the Huffman tree.

**Part B  5 points**
What bits are written to the compressed file as a result of compressing the word “trees”?

**Part C  4 points**
Using the same tree, what message is coded by the bit sequence below?

```
1 0 0 1 0 1 1 1 0 1 0 0 1 1 0
```
This problem is Extra Credit, it is NOT required

The tree below is a binary search tree.

```
lemon  
  |   |  
cherry apple  guava  
papaya  orange  tangerine  
  |   |            
  watermelon
```

Write the function *DoPrintPath* that prints, for each leaf in *tree*, one line of output consisting of the values stored on the root-to-leaf path. For example, for the tree above the output is shown below.

```
lemon  cherry  apple  
lemon  cherry  guava  
lemon  papaya  orange  
lemon  papaya  tangerine  watermelon
```

The function *DoPrintPath* is called by *PrintPath* as shown below. You may assume a boolean-valued function *IsLeaf* exists. Assume that there at most 200 strings stored in any tree.

```c
void DoPrintPath(TNode<string> * tree, Vector<string> & path, int depth)  
  // precondition: depth = depth of tree (# edges from root, root has depth 0)  
  // postcondition: for each leaf of tree, print values on root-to-leaf path  
  // one line of output for each leaf
  {
  
  }
```

```c
void PrintPath(TNode<string> * tree)  
{
  if (tree != 0)
  {
    Vector<string> path(200);  
    path[0] = tree->info;  
    DoPrintPath(tree,path,0);  
  }
}
```