Test 2: CPS 100

Owen Astrachan

November 15, 2000

Name: __________________________

Login: ____________

Honor code acknowledgment (signature) __________________________

<table>
<thead>
<tr>
<th>Problem</th>
<th>value</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem 1</td>
<td>12 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 2</td>
<td>30 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 3</td>
<td>8 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 4</td>
<td>20 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 5</td>
<td>16 pts.</td>
<td></td>
</tr>
<tr>
<td>TOTAL:</td>
<td>86 pts.</td>
<td></td>
</tr>
</tbody>
</table>

This test has 9 pages, be sure your test has them all. Do NOT spend too much time on one question — remember that this class lasts 50 minutes.

In writing code you do not need to worry about specifying the proper \texttt{include} header files. Assume that all the header files we’ve discussed are included in any code you write.

Some common recurrences and their solutions.

\[
\begin{align*}
T(n) &= T(n/2) + O(1) \quad O(\log n) \\
T(n) &= T(n/2) + O(n) \quad O(n) \\
T(n) &= 2T(n/2) + O(1) \quad O(n) \\
T(n) &= 2T(n/2) + O(n) \quad O(n \log n) \\
T(n) &= T(n-1) + O(1) \quad O(n) \\
T(n) &= T(n-1) + O(n) \quad O(n^2)
\end{align*}
\]
PROBLEM 1 :  \((acgtacgtacgt\ (12\ points))\)

You must develop a program to sort long strings representing strands of DNA. A strand is composed of the letters 'c', 'a', 'g', and 't'; for example some strands are shown below on the left.

\[
\begin{array}{ll}
tgaaaatcctttctatatttaatgc & \text{strands time (sec.)} \\
atggcattagggcgtttaa & 4000 \ 16 \\
gtacgccccgtgcccctaaactact & 8000 \ 64 \\
taggggaatgtccgaaaaacgcaggctccgcg & 10000 \ 100 \\
\end{array}
\]

The people who hire you require you to use insertion sort. You’ve developed a program and timed it on three input files. The results are shown above on the right in a table with the number of strands and the time to sort the strands.

**Part A**

Given this data, how long will it take the program to sort 20,000 strands of DNA? Show your reasoning.

**Part B**

Given this data, how long will it take the program to sort 1,000 strands of DNA? Show your reasoning.

**Part C**

You determine that the first letter in a strand is equally likely to be 'a', 'g', 't', or 'c'. You then make the following suggestion to sort the strands of DNA stored in a file.

1. As the strands are read in, store them in one of four vectors: one for strands beginning with 'a', one for strands beginning with 'g', one for 't', and one for 'c'.
2. Sort each of the four vectors.
3. Combine the four vectors into one vector taking first the strands from the 'a' vector, then the 'c' strands, then the 'g' strands, followed by the 't' strands.

This method will correctly sort the strands. Assuming there are roughly the same number of strands starting with 'a', 'c', 'g', or 't' in the files you’ll be sorting, roughly how long will it take with this new method to sort a file of 8,000 strands. Briefly justify.
PROBLEM 2:  (eerst (8 points))

Part A
The function `sort` below uses two helper functions and a binary search tree to sort a vector. The code is correct. What is the big-Oh complexity of `sort` assuming the vector `list` is in random order? Briefly justify your answer.

Part B
What is the complexity if the vector `list` is already sorted? Briefly justify your answer.

```cpp
tree * insert(tree * t, const string& s)
// pre: t is a search tree
// post: return t after adding s, t is a search tree
{
    if (t == 0) return new tree(s,0,0);
    else if (s <= t->info) t->left = insert(t->left,s);
    else t->right = insert(t->right,s);
    return t;
}

void traverse(tree * t, tvector<string>& tv)
// pre: t is a search tree
// post: values of t copied into tv in order
{
    if (t != 0) {
        traverse(t->left, tv);
        tv.push_back(t->info);
        traverse(t->right,tv);
    }
}

void sort(tvector<string>& list)
// post: list is sorted
{
    tree * t = 0;
    int k;
    for(k=0; k < list.size(); k++) {
        t = insert(t, list[k]);
    }
    list.clear();
    traverse(t,list);
}
```
**Problem 3: Stacking the Deck (30 points)**

**Part A (9 points)**
Evaluate the following prefix and postfix expressions. If the statements aren’t legal, write “error”. Show some work for partial credit.

1. \( 7 \ 3 \ 6 \ * \ + \ 4 \ * \)
2. \( 8 \ 6 \ + \ 5 \ + \ 2 \ 7 \ * \ + \)
3. \( 6 \ 2 \ - \ 5 \ * \)

**Part B (12 points)**
The code on the next page in `preparse.cpp` correctly parses an arithmetic expression written in prefix form, returns a pointer to the root of such an expression, and then evaluates the expression. For example, if the user enters \( * \ + 5 \ 2 \ 3 \) the code returns a pointer to the tree diagrammed below. This tree is then evaluated to 21, and the value 21 is printed.

![Tree Diagram]

**Part B.1**
What is the purpose of the \( = 0 \) in the member function `Expression::evaluate`?

**Part B.2**
Draw the tree that results from reading the expression \( + \ * \ + 7 \ 5 \ 1 \ 3 \) whose value is 15.

**Part B.3**
On the code page indicate where code for parsing an expression that begins with a minus sign belongs. Assume a class `MinusExp` has been implemented, but just indicate where the code belongs for processing a minus sign you do not need to write the code.

**Part B.4**
Suppose you copy/paste the class `MultExp` In implementing a new class `MinusExp` to represent prefix expressions that begin with a minus sign `-`. What code in the copied class `MultExp` would change in implementing `MinusExp` (other than the name of the class).
this is a code page
Part C (9 points)
You are implementing a queue using stacks (stacks use the standard class tstack). Adding an element to the queue causes the element to be pushed onto a stack holding all the queue elements (see code below). The dequeue operation requires popping all the elements onto a temporary stack, removing the top element from the temporary stack, and then popping the elements back to the original stack. This code is shown below for a class named staque that represents a queue using stacks.

Part C.1
What is the big-Oh complexity of one staque::dequeue operation when the staque contains $N$ elements. Briefly justify your answer.

Part C.2
What is the complexity of inserting $n$ elements and then deleting all $n$ elements one-at-a-time (using big-Oh). Briefly justify your answer.

Part C.3
What is the complexity of inserting an element, then deleting the element, and repeating this pair of operations $n$ times. Briefly justify your answer (use big-Oh).

```cpp
void staque::enqueue(int val)
{
    myStack.push(val); // val on stack
}

void staque::dequeue()
// post: first element inserted is removed
{
    tstack<int> temp;
    copyFromTo(myStack,temp);
    temp.pop();
    copyFromTo(temp,myStack);
}

void staque::copyFromTo(tstack<int>& from, tstack<int>& to) const
// post: all values copied from->to,
//        from.size() == 0
{
    while (from.size() > 0) {
        to.push(from.top());
        from.pop();
    }
}
```
PROBLEM 4: (Heap o’ Trouble (15 points))

The figure on the right below shows a (min) heap after inserting the values 22, 15, 18, 20, and 27, in that order.

The diagrams on the left show how the heap is stored after the first two values are inserted: the top figure/vector shows the heap after 22 is inserted, the second figure/vector shows the heap after 15 is inserted. Fill in the remaining figure/vectors by adding values in the order described above (first add 18, then 20, then 27) diagramming the heap after each addition.

In the last/bottom figure/vector, draw the heap after one deletemin operation executes.

Part B
It possible to create a priority queue using the class tpqueue so that the maximum value is stored at the root (instead of the minimum value). Briefly describe how this is done, don’t write code, just describe briefly what to do.
PROBLEM 5:  (Hashing (16 points))

We reviewed maps implemented using binary search trees and maps implemented using hash-tables.

Part A
Describe an application/situation where search trees are preferable to hash-tables and why they’re preferable.

Part B
Describe an application/situation where hash tables are preferable to search trees and why they’re preferable.

Part C
The code on the next page (we saw this code in class) determines if two vectors contain the same values. The code shown uses maps implemented with hash tables. If \( a \) and \( b \) each contain \( n \) values, what is the complexity of \( \text{isSame2}(a,b) \) in the average case? Why?

Part D
Why are pointers to map objects used instead of map objects.
void update(tmap<string,int> * map, const string& s)
// post: if map-&gt;contains(s) then count of s is incremented by 1
//        else map contains (s,1), so s occurs once
{
    if (map-&gt;contains(s)) map-&gt;get(s)++;
    else map-&gt;insert(s,1);
}

bool isSame2(tvector<string>& a, tvector<string>& b)
// post: return true if and only if a and b contain exactly the same values
{
    tmap<string,int> * map = new HMap<int>(10001);
    tmap<string,int> * map2 = new HMap<int>(10001);
    int k;
    for(k=0; k < a.size(); k++) {
        update(map,a[k]);
        update(map2,b[k]);
    }
    Iterator<pair<string,int> >* it = map-&gt;makeIterator();
    Iterator&lt;pair<string,int>&gt;* it2 = map2-&gt;makeIterator();
    it-&gt;Init(); it2-&gt;Init();
    while (it-&gt;HasMore() &amp;&amp; it2-&gt;HasMore()) {
        if (it-&gt;Current().first != it2-&gt;Current().first ||
            it-&gt;Current().second != it2-&gt;Current().second) {
            return false;
        }
        it-&gt;Next(); it2-&gt;Next();
    }
    return ! (it-&gt;HasMore() || it2-&gt;HasMore());
}