CSC 225 Final Exam
December 1997

1. [20] Circle true or false for each question and justify your answer. No marks will be given unless there is a correct justification.

(a) A heap can be constructed in O(n) time.
True **False**

(b) When designing a divide and conquer algorithm for items stored in a linked list of size n, an approach which divides the problem in half will always be faster asymptotically in the worst case than one which divides the list into one subproblem of size one and one of size $n - 1$.
True **False**

(c) Let f, and g be functions from the natural numbers to the positive real numbers. Then if $f \in O(g)$ and $g \in \Omega(f)$, then $f \in \Theta(g)$.
True **False**

(d) Radix sort which is $O(n)$ is faster on all inputs than any of the comparison model algorithms.
True **False**

2. Consider the following C code fragment:

```
answer = 0;
x = 2;
for (i= 0; i < n-1; i++)
{
   answer = answer + x;
   x = x + 2;
}
```

Your goal is to determine a closed formula for *answer* when the loop terminates.

(a) [3] Set up an appropriate loop invariant.

(b) [5] Prove by induction that your loop invariant is correct.

(c) [2] Use your loop invariant to determine a closed formula for the value of answer when the loop terminates.

3. For this question, use weighted union (weight is the number of nodes in a component), with **collapsing** find. The nodes should be numbered starting with zero.

(a) [4] Give a sequence of eight union instructions which results in the *parent* array equal to:

-8	0	1	2	1	6	0	0	-1

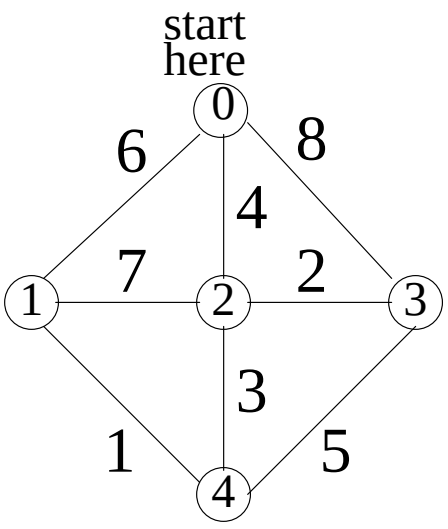
1.
2.
3.
4.
5.
6.
7.
8.

(b) [3] What are the new contents of the *parent* array from part (a) after a call to W-UNION(8, 3, *parent*)?

(c) [8] Prove that the weighted union which uses the number of nodes in a component as the weight takes $O(\log n)$ for FIND in the worst case on a graph which has n nodes by proving by induction:

Statement S(h): A tree of height h, formed by the policy of merging the tree with fewer nodes into the tree with a larger number of nodes has at least 2^h nodes.

4. [15] Show the values of the data structures *tree*, *min_wt*, and *closest* (as described in class) after each phase of the Dijkstra/Prim minimum spanning tree algorithm. The *phase* equals the number of vertices in the tree so far. The columns are numbered by the phase and the rows by the vertex numbers. Mark the edges in the MST.



tree:	0	1	2	3	4	5
0						
1						
2						
3						
4						

min_wt:	0	1	2	3	4	5
0						
1						
2						
3						
4						

closest:	0	1	2	3	4	5
0						
1						
2						
3						
4						

5.(a) [6] Fill in the following chart with the worst case time complexities using Θ notation:

Operation	Sorted array	Min-heap	Binary Search Tree
INSERT key x			
FIND key x			

(b) [6] Consider the following three sequences of operations each starting out initially on an empty data structure:

P_1: n INSERT's, n^2 FIND's P_2: n INSERT's, $\log(n)$ FIND's P_3: n INSERT's, n FIND's

Which data structures do you recommend from part (a) for each sequence if you want to optimize the behaviour in the worst case? Justify your answer.

(c) [3] What approach from part (a) do you think would work best in practice on average on these and similar applications? Justify your answer.

6. [10] Consider the following sorting algorithm which uses the **find_middle** function defined below:

final_sort(*start*)
Initially, *start* points to the beginning of an unsorted linked list. The links in the list are rearranged so that on termination, the values are sorted.

0. If *start* is NULL return.
1. Call find_middle(*start*, *middle*).
2. Traverse the list which begins at *start*. Place cells which have key value less than or equal to *middle* $\rightarrow$ *data* in a list with start *start_small* and the others into a list which has start *start_big*.
3. Call **final_sort**(*start_small*).
4. Call **final_sort**(*start_big*).
5. Return with *start* pointing to the front of a list consisting of the concatenation of the lists which start at *start_small*, *middle*, and *start_big*.

find_middle(*start*, *middle*)
Initally, *start* points to the beginning of a linked list with $n \geq 1$ data items. Let x_0, x_1, ... , x_{n-1} be the data values as listed in sorted order. This routine removes a cell from the list with data value x_k where $k = \left\lfloor \frac{n}{2} \right\rfloor$. On return, *start* points to this new list of $n - 1$ items and *middle* points to the cell that was removed. The *next* pointer of the cell that was removed is set to NULL.

The **find_middle** function takes time $\Theta(n)$ when called with a list of size n.

Analyze the worst case time complexity of this algorithm in detail. This should include an appropriate recurrence relation and its solution. For full marks, you must fully justify the recurrence you are using.

Important Hint: What happens when all the data values are equal?

7.(a) [5] Explain how to change the ‘‘divide’’ step of the algorithm from Question #6 to create an algorithm which is faster in the worst case. Hint: The correct solution has linear time complexity when the data values are all equal.

(b) [10] Give a detailed worst case time complexity analysis of your algorithm from part (a). This should include an appropriate recurrence relation and its solution. For full marks, you must fully justify the recurrence you are using.