Name: ______________________________

ID Number: ___________________________

UNIVERSITY OF VICTORIA
EXAMINATIONS- DECEMBER 2009
CSC 225 - Algorithms and Data Structures: I
Section A01 (CRN 10839)
Instructor: Wendy Myrvold
Duration: 3 hours

TO BE ANSWERED ON THE PAPER.

Instructions:

Students **MUST** count the number of pages in this examination paper before beginning to write, and report any discrepancy immediately to the invigilator.

This exam has eleven pages (the last page is blank in case you need extra space) plus the header page.

Use only space provided on exam for answering questions. Closed book. No aids permitted.

Question	**Value**	**Mark**
1	20	
2	20	
3	10	
4	10	
5	15	
6	25	
Total	**100**	

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

(a) [5] Suppose singly linked lists are used to implement a queue. Then, adding to the queue should be done at the front of the list, and deletions from the queue should be taken from the end of the list.

True **False**

(b) [5] It is possible to sort an array of n numbers in $O(n \log n)$ time in the worst case using only $O(1)$ extra space.

True **False**

(c) [5] 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 1 and one of size $n-1$.

True **False**

(d) [5] The lower bound for sorting is $\Omega(n \log n)$ so it is impossible to find a sorting algorithm which is $O(n)$ in the worst case.

True **False**

2.(a) [8] Give detailed pseudocode for a recursive middleMin algorithm. The algorithm should take as input a linked list which has both a start and a rear pointer and also a value n which is the number of items in the list. It should return a pointer, *prevPtr*, which is null if the minimum value is at the start of the list, and otherwise it points to the cell just before the one which has the minimum value. The algorithm should solve the problem by splitting the list in half, solving each half recursively, and then marrying the solutions.

middleMin(n, start, rear) : returns a pointer *prevPtr* to a list node.

Question 2 continued.

(b) [2] Set up a recurrence for the time complexity of your middleMin algorithm from (a).

(c) [5] Solve your recurrence from (b). You may assume that $n = 2^k$ for some integer k.

Question 2 continued.

(d) [5] Prove by induction that your solution from part (c) to the recurrence in part (b) is correct.

Your recurrence:

Your solution:

3. [10] You wish to determine the height expressed in floors of a building from which a Little Tykes tricycle will break if tossed out of a window on that floor. The building has n floors, and 2 tricycles are available. You are allowed only one operation to determine the breaking height- you may toss a tricycle out the window. If it does not break it can be reused in another experiment. One possible outcome is that tricycles will not break even when dropped from the top floor. Otherwise there is some floor f so that any tricycle dropped from a floor below f does not break and a tricycle dropped from floor f or higher does break. The fact that a tricycle has been used in a previous experiment (and it did not break) does not affect the breaking height of the tricycle.

   Draw the decision tree for 15 floors and 2 tricycles which is optimal with respect to minimizing the number of experiments done in the worst case.

4. [10] What is the time complexity of Kruskal's algorithm for finding a minimum weight spanning tree if the data structures and algorithms chosen for choosing the minimum weight edge at each step and the union/find data structure are as follows. Express your times in terms of both *n* and *m* where *n* is the number of vertices of the graph and *m* is the number of edges.

(a) The adjacency matrix is scanned at each step to find a minimum weight edge. The flat union/find data structure is applied.

(b) The edges are stored in a min-heap and deleteMin is used to remove an edge from the heap at each step. The union/find data structure is a weighted union with a non-collapsing find.

(c) Radix sort is used to first sort the edges by weight. Weighted union is used with a collapsing find.

Fill in this table:

The column *E* represents the total amount of work that the algorithm does in order to pick a minimum weight edge summed over all the iterations.

The column *U* represents the total amount of work that is done for all the union/find operations.

The column *K* represents the time complexity of Kruskal's algorithm.

Algorithm	*E*	*U*	*K*
(a)			
(b)			
(c)			

5. 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.

(a) [2] *tree:*

Phase:	0	1	2	3	4	5	6
a							
b							
c							
d							
e							
f							

(b) [5] *min_wt:*

Phase:	0	1	2	3	4	5	6
a							
b							
c							
d							
e							
f							

(c) [5] *closest:*

Phase:	0	1	2	3	4	5	6
a							
b							
c							
d							
e							
f							

(d) [3] Mark the edges in the minimum weight spanning tree.

6. Suppose that Universal Studios has a large collection of old movies that they want to sort. Each movie has an integer identification number which is the key for the sort. The movies take up a lot of space (several Megabytes each) and so swapping takes an enormous amount of time; the time taken for other operations is negligible.

(a) [7] Assuming it takes at most one hour per swap operation, derive a tight upper bound on the time that it would take to sort *n* movies using a MaxSort, and then prove that your answer is correct using induction.

(b) [3] Assuming you use a Heapsort, give a function $f(n)$ such that the number of swaps in the worst case on a problem of size n is in $\Theta(f(n))$ counting all the swaps done during all of the deleteMax operations, but ignoring swaps done building the initial heap.

(c) [7] Prove that the Heapsort does $\Omega(f(n))$ swaps (ignoring those done building the initial heap) for your $f(n)$ from (b). You should include in your argument a description of one example of a heap of order n such that this worst case time is realized.

(d) [5] Prove that the Heapsort does $O(f(n))$ swaps in the worst case (ignoring those done building the initial heap).

Your $f(n)$ from part (b):

(e) [3] If Universal Studios asks you which sorting routine they should use for this problem, MaxSort or Heapsort, what would your advice be and why? Justify your answer.

Use this page if you need more space.
Clearly indicate the question you are answering.