UNIVERSITY OF VICTORIA
EXAMINATIONS DECEMBER 2001
COMPUTER SCIENCE 225 (F01)

NAME: _____ STUDENT NO. _____

INSTRUCTOR: Frank Ruskey SECTION: F01

DURATION: 3 Hours

TO BE ANSWERED ON THE PAPER

STUDENTS MUST COUNT THE NUMBER OF PAGES IN THIS EXAMINATION PAPER BEFORE BEGINNING TO WRITE, AND REPORT ANY DISCREPANCY IMMEDIATELY TO THE INVIGILATOR.

THIS QUESTION PAPER HAS 9 PAGES.

NOTES: (1) ANSWER ALL QUESTIONS, (2) THERE ARE A TOTAL OF 2^7 MARKS, SCRATCH PAPER IS AVAILABLE FROM THE INVIGILATOR.

Question	Possible marks	Actual marks
1	54	
2	16	
3	6	
4	8	
5	6	
6	5	
7	12	
8	5	
9	5	
10	5	
11	6	
Total	128	

1. Fill in the blanks: [2 marks each, 54 total]

   - What's another name for a LIFO (last-in first-out list)? _____.
   - What exactly is the value of $1 + 2 + \cdots + 2^n$? _____.
   - The minimum number of edges in a connected undirected graph without cycles is (state your answer in terms of n, the number of vertices in the graph).

     ANSWER: _____.
   - Which data structure (circle one) is most closely related to depth-first-search?
     ANSWER: (STACK, QUEUE)
   - Properly implemented, the running time of Huffman's algorithm on an input of n frequencies is (give a big-O answer):
     ANSWER: _____.
   - Using the standard merging algorithm on two sorted lists, one of size n and the other of size m, what (exactly) is the worst case number of key comparisons used by the algorithm?
     ANSWER: _____.
   - What is the exact number of comparisons in the "lower bound on comparison based sorting" for sorting n items (use floor/ceiling).
     ANSWER: _____.
   - Exactly how many comparisons are required to verify that an array of n integers is in sorted order?
     ANSWER: _____.
   - Name a good algorithm for sorting an array in which no item is more than a fixed constant number of positions away from its correct location.
     ANSWER: _____
   - If Algorithm A is $O(n)$ and Algorithm B is $O(n^2)$ does this mean that A is faster than B for all sufficiently large values of n?
     ANSWER: _____.
   - In a binary heap implemented with arrays, the children of the node stored at location i are stored in locations
     ANSWER: _____ and _____.
   - Name two sorting algorithms that are $O(n \log n)$ in the worst case:
     ANSWER: _____ and _____.
   - What is the most commonly used efficient data structure for implementing dictionaries?
     ANSWER: _____
   - What are the two main operations for the priority queue data structure?
     ANSWER: _____ and _____

- In a proof that $3n^2 - 5n + 6$ is $\Omega(n^2)$, what is the smallest value of n_0 that could be used, given that $c = 2$?
  ANSWER: _____

- If the load factor of a hash table is λ and separate chaining is used to resolve collisions, what is the average number of probes used in an unsuccessful search? In a successful search?

  ANSWER: unsuccessful _____ successful _____

- Name a sorting algorithm that is not comparison based.

  ANSWER: _____

- What is the result of running the book's partitioning phase of quicksort on the data 7,3,5,8,1,6,2,4 (pivot is 4)?
  ANSWER: _____

- What is the exact number of key comparisons used in partitioning n distinct elements in an array?
  ANSWER: _____

- What $O(n + m)$ algorithm would you use to find the shortest path from v to w in an *unweighted* directed graph?
  ANSWER: _____

- What is the big-O running time of performing a rotation in a binary search tree?
  ANSWER: _____

- Give a single Java statement for pushing an element `x` onto a stack implemented as an array `st[0..n]` where `st[top]` is the top element on the stack.

  ANSWER: _____.

- With the same assumptions as the previous question, give a single Java statement for popping and returning the top element of a stack.

  ANSWER: _____.

- The running time of a preorder traversal of a binary tree is (use big-O):

  ANSWER: _____.

- The running time of the best implementation we saw in this course of Dijkstra's algorithm on dense graphs is (n vertices, m edges):

  ANSWER: _____.

- The running time of the best implementation we saw in this course of Dijkstra's algorithm on sparse graphs is (n vertices, m edges):

  ANSWER: _____.

- The two techniques used to speed up the UNION-FIND operations using up-trees (parent arrays) are called
  ANSWER: _____ and _____.

2. [16 marks] Do a depth-first search of the following directed graph. Assume that the driver loop processes the vertices in alphabetic order.

A: → D → E
B: → E
C: → F → B → G
D:
E: → H
F: → G
G: → E → I
H: → A
I: → H → F

Draw the graph.

List below the DFS and postorder numberings of the vertices:

Vertices:	A	B	C	D	E	F	G	H	I
DFS numbering:									
postorder numbering									

List the tree edges: _____ back edges: _____

List the forward edges: _____ cross edges: _____

List the strongly connected components: _____

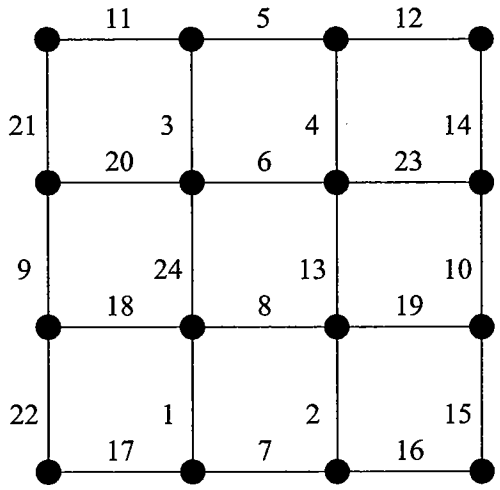
List the topological sort (ignore back edges) derived from the postorder numbering

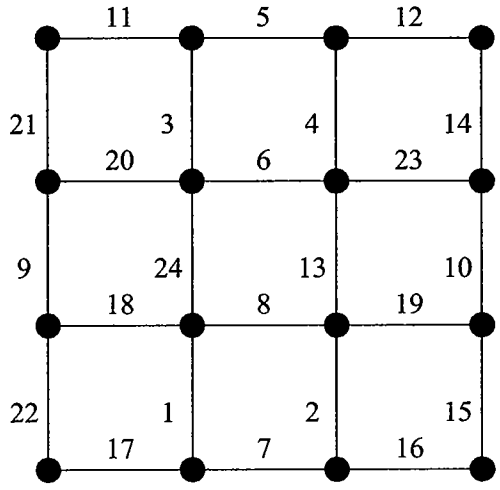
_____

3. (a) Carefully explain why every comparison based array sorting algorithm that only transposes adjacent elements of an array must be $\Omega(n^2)$ in the worst case, when sorting an n element array. (b) Argue the same lower bound, except for the average case; assuming that all permutations of the data in the array are equally likely. [6 marks]

4. Draw an AVL tree of height 4 and with the least possible number of nodes. Give a recurrence relation for $A(h)$, the least number of nodes in an AVL tree of height h. [8 marks]

5. Show, by marking the edges, the spanning tree that results after 11 edges have been added to the tree by both Kruskal's (on the left) and Prim's (on the right) algorithms. For Prim's algorithm start at the upper left corner. [6 marks]





6. Give a small example showing that Dijkstra's algorithm can fail if negative edge weights are allowed. Use no more than 4 vertices in your example. [5 marks].

7. [12 marks] Below is the DFS code for *undirected* graphs that was presented in class and posted on the course web site.

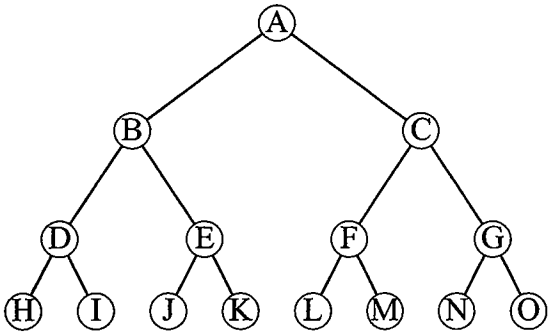
```
void dfs ( int v, int u ) {  // Parameter u is the parent of v.
   num[v] = ++c;
   visited[v] = true; /*** #1 ***/
   System.out.println( "Visiting "+v );
   for (Edge p=adj[v]; p != null; p = p.link ) {
      if (!visited[p.vert] /*** #2 ***/) {
         par[p.vert] = v; /*** #3 ***/
         System.out.println( "Tree edge: ("+v+","+p.vert+")" );
         dfs( p.vert, v );
      } else
      if ((p.vert != u && num[v] >= num[p.vert]) /*** #4 ***/ )
         System.out.println( "Back edge: ("+v+","+p.vert+")" ); /*** \#5 ***/
   }
}

void traverse() {
   int comp = 0;
   for (int i=0; i<n; ++i) {
      if (!visited[i] /*** #6 ***/) {
         System.out.println( "Component #"+(++comp)+":" );
         dfs( i, -1 ); /*** #7 ***/
         System.out.println();
      }
   }
}
```

Assuming that the input to this program is an *undirected* graph g with n vertices, m edges and c connected components, exactly how many times are the following statements(expressions) executed(evaluated) when `g.traverse()` is called? Assume that the graph has no self-loops.

statement or expression	number of times executed
#1	
#2	
#3	
#4	
#5	
#6	
#7	

8. Draw the ordered forest that corresponds to the following binary tree. The forest should look familiar; why? [5 marks]



9. Draw the binary search tree whose postorder traversal is B,C,A,F,H,G,E,D. [5 marks]

10. In the in class implementation of Huffman's algorithm we wrote a class `tNode` for the trees being placed on the heap. This class had to implement `Comparable` so that we could use Weiss' Binary Heap implementation. Fill in the missing code below in the method `compareTo`. [5 marks]

```
class tNode implements Comparable {
  tNode left, right;
  int valu;
  tNode ( int vv, tNode ll, tNode rr ) {
    valu = vv;  left = ll;  right = rr;
  }
  public int compareTo ( Object x ) {
    // You fill in the missing code below:









  }
}
```

11. Assuming the binary tree node declaration `class Node { Node ll, rl }` what is computed by the following functions when `f(r)` is called, where `r` is the root of a binary tree T? [6 marks]

```
int f ( Node t ) { return( t==null ? 0 : 1+f(t.ll)+f(t.rl) ); }
```

```
int f ( Node t ) { return( t==null ? -1 : 1 + max( f(t.ll), f(t.rl)); }
```

```
int f ( Node t ) { return( t==null ? 1 : f(t.ll)+f(t.rl) ); }
```

END