Department of Computer Science

COMP171: Data Structures and Algorithms
Fall 2004

Final Examination

Dr. Qiang Yang

Date: Tuesday, December 14, 2004
Time: 8:30–11:30am
Venue: LG4204

Please follow the lecture notes for all algorithms when they are different from the text book.
Your answers will be graded according to correctness, efficiency, precision and clarity.
This is a closed book examination. Please put aside your cellphones, PDAs, etc.
This exam has 9 questions, for a total of 155 points, and 22 pages, please check it.

Answer the questions in the spaces provided on the question sheets. If you run out of room for an answer, continue on the back of the page.

Name and section: _____

Student ID: _____

Email address: _____

I did not cheat in this examination (Signature): _____

Question	Points	Score
1	15	
2	10	
3	10	
4	20	
5	10	
6	35	
7	10	
8	25	
9	20	
Total:	155	

1. **Complexity** *(15 points)*

   Give a tightest possible bound for the Big-Oh runtime complexity of each of the following C code fragments in terms of the value of the (input) parameter n.
   Variables are all of type `int` unless otherwise stated.
   Results without derivations do not receive any points.

   (a) (5 points) .

```
1   sum = 0;
2   for ( i = 0; i < n; i += 2 ) {
3       for ( j = n * n; j > 4; j-- ) {
4           sum++;
5       }
6   }
```

   Solution:
   $O(1) + O(1)*$ (# of times line 4 is executed)
   $= O(1) + O(1) * ((n^2 - 4) * (n/2))$
   $= O(n^3)$
   OR
   $T(n) \cong \sum_{i=1}^{n/2} n^2 - 4 = n^3/2 - 2n = O(n^3)$

   (b) (5 points) .

```
0       // delta and x are float, not int
1   x = n;
2   delta = 1 / n;
3   while ( x > 0 )
4       x = x - delta;
```

   Solution:
   $O(1) + O(1)*$ (# of times line 4 is executed)
   $= O(1) + O(1) * (n/delta = n/(1/n) = n^2)$
   $= O(n^2)$

(c) *(5 points) (Hint: The complexity of* `toosimple(n)` *is* $T(n)$*.)*

```
1   int toosimple( int i ) {
2       if ( i < 1 )
3           return 1;
4       else
5           return toosimple( i - 2 );
6    }
```

Solution:

$T(0) = c$

$T(n)$

$= T(n-2) + 1$

$= T(n-4) + 1 + 1$

$= T(n-6) + 1 + 1 + 1$

$\vdots$

$= T(n-i) + i/2$

So let $n - i = 0$ then $i = n$

such that $T(n) = T(0) + n/2$

$= O(n)$

2. **Multiple Choice** *(10 points)*

Each question in this section is worth 2 points, it has exactly one correct answer and you are not penalized for incorrect answers. Write clearly. If we cannot understand your choice, you will receive no credit.

(a) (2 points) Let S be a nonempty stack. Suppose an item is first popped, and then pushed back to S, resulting in $S1$. Is **always** $S1 = S$ (true or false)?

A. True B. False

Solution: True

(b) (2 points) Let Q be a nonempty first-in-first-out (FIFO) queue. Suppose an item is first dequeued (removed), and then enqueued (inserted) into the queue Q, resulting in $Q1$. Is **always** $Q1 = Q$ (true or false)?

A. True B. False

Solution: False

(c) (2 points) The best case performance of merge sort is:

A. $O(n^2)$ B. $O(n \log n)$ C. $O(n)$ D. $O(n^2 \log n)$

Solution: $O(n \log n)$

(d) (2 points) Which of the following sorting algorithms has the highest space requirements?

A. Quick Sort B. Bubble Sort C. Merge Sort D. all are the same

Solution: Merge Sort, $2n$

(e) (2 points) True or False? For a graph G and a node v in that graph, the DFS and BFS trees of G rooted at v always contain the same number of edges.

A. True B. False

Solution: True

3. **AVL Trees** *(10 points)*

Given is a complete binary tree with 3 levels. The root contains 'd'. The first level nodes (from left to right) carry 'b' and 'f'. The leaf nodes (from left to right) show the data 'a', 'c', 'e', and 'g'.

The keys in the tree always follow lexicographical (= alphabetical) order.

Draw the sequence of AVL trees that results when the following items are added to the above AVL tree in the order shown:
'p', 'o', 'n', and 'm'.
Draw this sequence including **all** before and after trees. Draw also a figure immediately after an item has been inserted, and no rotation has been done yet. In case of double rotations, you can draw a **single** picture to show the AVL tree after the double rotation.
After each rotation there must be a separate figure with a caption to explain it.

(a) (2 points) Insert(p):

Solution: Weiss, page 149. Numbers represent letters (a=1, b=2, ...).
Insert(16)

(b) (3 points) Insert(o):

Solution: Weiss, page 149.
Insert(15)

(c) (3 points) Insert(n):

Solution: Weiss, page 150.
Insert(14)

(d) (2 points) Insert(m):

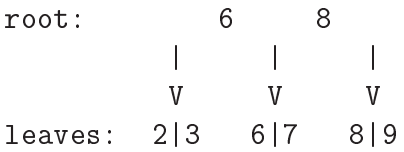
Solution: Weiss, page 150.
Insert(13)

4. **B+ Trees** *(20 points)*

Construct a B+ tree with branching factor $M = 3$. Each leaf can contain up to L data items, where $L = M$.

(a) (5 points) Draw the B+ tree after insertion of 2, 9, 3, 8, 6, and 7 (in this given order).

Solution:



You may use a calculator to doublecheck your calculations. But you must show **all** derivations to get full marks.

(b) (5 points) We have 1,000,000,000 records that we wish to store in a B+-tree where the branching factor M and leaf node size L are equal. We wish to limit each lookup to no more than 3 disk accesses. We also wish to leave maximum room for future insertion, in order to avoid node splits.

What is M?

Solution: $(\frac{M}{2})^2 = \frac{100{,}000{,}000}{M/2}$
$\Rightarrow (\frac{M}{2})^3 = 100,000,000$
$\Rightarrow \frac{M}{2} = 1000 \Rightarrow M = 2000$
Unforutnately there was a typo in the exam:
100,000,000 instead of 1,000,000,000.
Anyway, most of the students corrected it unconsciously, and others used the calculator ..
Both versions are accepted.

Now assume a branching factor $M = 10$. And assume that we need 10,000 nodes in the tree at the leaf level. We drop the requirement that we want to leave maximum room for future insertion.

(c) (5 points) How many levels does the tree have?

Solution: Let n the number of leaf nodes. Number of levels is

$$\begin{aligned} \lceil \log_M n \rceil + 1 &= \lceil \log_{10} 10,000 \rceil + 1 \\ &= 4 + 1 = 5 \end{aligned}$$

(d) (5 points) For each level of the tree, how many nodes are at the level?

Solution:

level	Number of nodes
5 (leaf)	$10,000$
4	$\lceil 10,000/10 \rceil = 1,000$
3	$\lceil 1,000/10 \rceil = 100$
2	$\lceil 100/10 \rceil = 10$
1 (root)	$\lceil 10/10 \rceil = 1$

5. **Hashing** *(10 points)*

Consider an insertion of they key $x = 222$ into the hash table shown in the figure. For both of the following probing methods, calculate and indicate the sequence of table entries that would be probed, and the final location of insertion for the key. The hash function is $h(y) = y \bmod 10$ so for the key $x = 222$ we have $h(x) = 2 \bmod 10$. (In both cases start with the same initial table, an example calculation is given at index 2.)

0	
1	
2	152
3	53
4	
5	75
6	136
7	27
8	
9	999

(a) (5 points) Linear probing.

0		
1		
2	152	$h(222) = 2 \pmod{10}$. (example calculation)
3	53	
4		
5	75	
6	136	
7	27	
8		
9	999	

Solution:

0		
1		
2	152	$h(222) = 2 \pmod{10}$. (example calculation)
3	53	$h_1(222) = h(222) + f(1)(\text{mod}10) = 2 + 1(\text{mod}10) = 3(\text{mod}10)$.
4	**222**	$h_2(222) = h(222) + f(2)(\text{mod}10) = 2 + 2(\text{mod}10) = 4(\text{mod}10)$.
5	75	
6	136	
7	27	
8		
9	999	

(b) (5 points) Quadratic probing

0		
1		
2	152	$h(222) = 2 \pmod{10}$. (example calculation)
3	53	
4		
5	75	
6	136	
7	27	
8		
9	999	

Solution:

0		
1	**222**	$h_3(222) = h(222) + f(3)(\text{mod}10) = 2 + 9(\text{mod}10) = 1(\text{mod}10)$.
2	152	$h(222) = 2 \pmod{10}$. (example calculation)
3	53	$h_1(222) = h(222) + f(1)(\text{mod}10) = 2 + 1(\text{mod}10) = 3(\text{mod}10)$.
4		
5	75	
6	136	$h_2(222) = h(222) + f(2)(\text{mod}10) = 2 + 4(\text{mod}10) = 6(\text{mod}10)$.
7	27	
8		
9	999	

6. **Sorting** *(35 points)*

(a) (10 points) Give a tightest possible bound for the Big-Oh worst case runtime complexity of the following sorting algorithms in terms of the value of the input parameter n.

Algorithm	Running Time
BubbleSort	
InsertionSort	
MergeSort	
HeapSort	
QuickSort	

Solution:

Algorithm	Running Time
BubbleSort	$O(n^2)$
InsertionSort	$O(n^2)$
MergeSort	$O(nlogn)$
HeapSort	$O(n \log n)$
QuickSort	$O(n^2)$

(b) (5 points) Assume that in a Quicksort implementation, the pivot is always selected as the left most element of the array.
Under which conditions does the worst case for Quicksort appear?
Show the complete mathematical worst case analysis.

Solution: Weiss, page 277.

Assume we have an unsorted list of n numbers. We must store the numbers in an array of size n.

(c) (5 points) Propose an $O(\log n)$ search algorithm to access any key k.
Hint: You can use an $O(n \log n)$ algorithm once to rearrange the numbers.

Solution: Binary Search, after sorting the numbers.

(d) (5 points) Analyze the time complexity of your search algorithm.
Show that its worse case running time is bounded by $O(\log n)$.

Solution: Analysis of Binary Search (lecture notes).

(e) (5 points) Given two sets A and B of n integers, determine if A is equal to B.

Can you do this in $O(n \log n)$ time?

Solution: Heapsort A and B and compare the sorted lists sequentially.

(f) *(5 points) BONUS question*
Let $A = a_1, a_2, ..., a_n$ and $B = b_1, b_2, ..., b_n$ be two lists of n integers, x another integer.
Write an efficient algorithm to decide if $x = a_i + b_j$ for some i and j.
Sure you can do it in $O(n^2)$ time. But can you do it in $O(n \log n)$ time?

Solution: Define another list $B' = x - b_1, x - b_2, ..., x - b_n$. Sort both A and B', and then compare sequentially.

7. **Heaps** *(10 points)*

Suppose that we run heap sort on $A = [10; 9; \ldots; 2; 1]$.
In the beginning the array looks like:

keys		10	9	8	7	6	5	4	3	2	1
index	0	1	2	3	4	5	6	7	8	9	10

(We start from position 1 in the array).

(a) What is the instance of A after build-**MIN**-heap applied to A?
Fill in the array that represents the heap.
(You may additionally draw the min heap to get a better picture for yourself. But this is not sufficient to get marks for this question.)

i. (2 points) percolateDown(index 5):

keys											
index	0	1	2	3	4	5	6	7	8	9	10

Solution:

keys		10	9	8	7	1	5	4	3	2	6
index	0	1	2	3	4	5	6	7	8	9	10

ii. (3 points) after percolateDown(index 4), percolateDown(index 3), percolateDown(index 2), and percolateDown(index 1):

keys											
index	0	1	2	3	4	5	6	7	8	9	10

Solution:

keys		1	2	4	3	6	5	8	10	7	9
index	0	1	2	3	4	5	6	7	8	9	10

(b) (4 points) Now we want to sort the numbers in decreasing order.
What is the instance of A after the first iteration of the loop of the heap sort algorithm?

keys											
index	0	1	2	3	4	5	6	7	8	9	10

Solution:

keys		2	3	4	7	6	5	8	10	9	1
index	0	1	2	3	4	5	6	7	8	9	10

(c) (1 point) What is the instance of A after the 9th iteration of the loop of the heap sort algorithm?

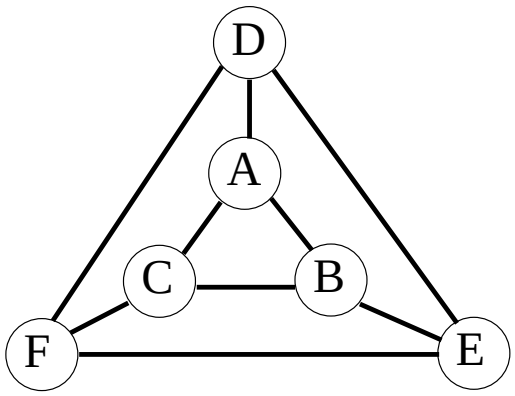
keys											
index	0	1	2	3	4	5	6	7	8	9	10

Solution: Sorted in decreasing order.

keys		10	9	8	7	6	5	4	3	2	1
index	0	1	2	3	4	5	6	7	8	9	10

8. **Graph Algorithms** *(25 points)*

Consider the following undirected graph.



(a) (5 points) Draw a DFS tree for the graph starting at node A. (If several nodes can be chosen at some step of the DFS, pick the vertex that is alphabetically first.)

Solution: A - B - C - F - D - E

(b) (5 points) Draw a BFS tree for the graph starting at node A. (If several nodes can be chosen at some step of the BFS, pick the vertex that is alphabetically first.)

Solution:

Use only the graph algorithms learned in class to design new algorithms (here you do not have to explain how the algorithms learned in class work and you can just refer to them as subroutines).

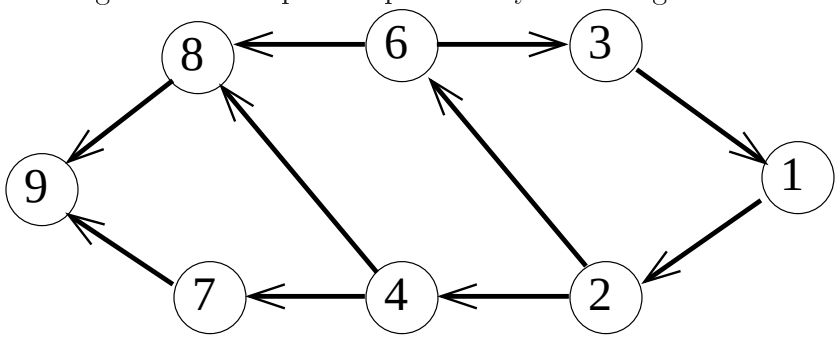
(c) (5 points) Using BFS as a subroutine, calculate the number of maximally connected components in an undirected graph.

Solution:

```
counter=0;
For v=1..n, do {if (flag(v)=F) BFS(v); counter++}
```

(d) (5 points) Modify the BFS algorithm to check if a cycle exists in a directed graph that goes through a given node s.

Figure 1: Example Graph with cycle through $s = 6$.



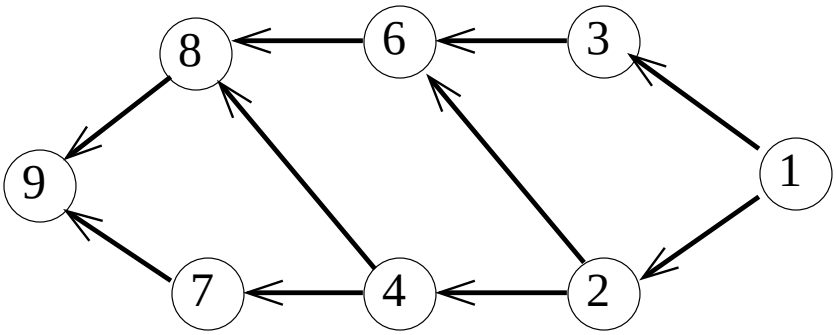
Solution: we can check, in a breadth first manner, whether a path exists back to s.

(e) (5 points) Modify the DFS algorithm so that, when applied to binary trees, it gives a post order traversal of the tree.

Solution: Usual implementation.

9. **Topological Sort** *(20 points)*

Consider the following graph.



(a) (6 points) What is the topological sort of the above graph if the depth-first search is started at node 1 and neighbors are visited in numerical order?

Solution: 1 - 3 - 2 - 6 - 4 - 8 - 7 - 9

(b) (2 points) Give an example of a DAG G with at least four vertices that has a unique topological sort.

Solution:

```
A -> B -> C -> D
```

(c) (2 points) Give an example of a DAG G with at least four vertices in which every permutation of the vertices is a topological sort.

Solution: Four single nodes, no edges: A B C D.

A complete graph with bidirected arcs is NOT acyclic.

(d) (10 points) Let `S` be an empty stack initially. Modify the DFS algorithm to return the stack such that when all elements are popped, it gives a topological sort of the vertices in a DAG.
Call this procedure `topsort(v: vertex, var S: stack)`.
You can use an array `flag`, where `flag[v] = T` means that vertex `v` has been visited.
`S` has two operations: `S.push(v)` and `S.pop()`.

Solution: (see lecture notes.)

```
procedure topsort (v: vertex, var S: stack) {
    // S passed by reference
    var w: vertex
    flag[v]=T;
    for each neighbor w  of v, do
        if (! flag[w]) then
            topsort(w, S);
    S.push(v);
}
```