COMP 3711 Design and Analysis of Algorithms
Fall 2011 Final Exam

1. **Quick-Answer Questions** ($5 \times 4 = 20$ pts)

   1.1 Please arrange the following functions in asymptotic ascending order (e.g., n, n^2, n^3): (a) n; (b) $n \log n$; (c) $n \log^* n$; (d) $\frac{n}{\log n}$; (e) $n^{1.0001}$.

   1.2 What is the minimum number of nodes in an AVL tree of height 4? Note that the AVL tree of height 1 has 1 node; the AVL tree of height 2 has 2 nodes at the minimum.

   1.3 Radix sort may take $O(n)$ time to sort n integers. This breaks the $\Omega(n \log n)$ sorting lower bound we have proved. Is this a contradiction? Why?

   1.4 Let G be a weighted, connected, undirected graph, where all the weights are distinct. Dijkstra's algorithm can be used to find all the shortest paths from a given source vertex. These paths form a tree, called the *shortest path tree*. Is it always the same as the minimum spanning tree of G? If yes, give a proof; otherwise give a counter example.

2. **Majority** (10 pts)

   Assume you have an array $A[1..n]$ of n elements. A *majority element* of A is any element occurring more than $n/2$ times (e.g., if $n = 8$, then a majority element should occur at least 5 times). Your task is to design an algorithm that finds a majority element, or reports that so such element exists.

   (a) This problem can be solved by simply sorting the array in $O(n \log n)$ time. But to have more fun (or torture?) we have decided that you are not allowed to order the elements, but can only test them for equality. More precisely, the only way you can access the elements is to check whether two elements are equal or not, so you cannot sort the elements. Can you still design an $O(n \log n)$-time algorithm for this problem? [Hint: Divide-and-conquer.]

   (b) You will get 10 bonus points if you can design an $O(n)$-time algorithm. Of course, you are still only allowed to use equality tests on the elements. (Warning: This is very difficult — do not spend too much time on it unless you have finished all other questions.)

3. **Topological Sort** (8 pts)

   We have shown in class how to do topological sort by modifying the BFS algorithm. We can also design a topological sort algorithm based on DFS. Please complete the following DFS-based topological sort algorithm.

Algorithm 1: Main algorithm

```
foreach u ∈ V do
    color[u]= WHITE;
end
foreach u ∈ V do
    if color[u]=WHITE and in-degree(u) = 0 then
        DFS-visit(u);
    end
end
```

Algorithm 2: DFS-visit(u)

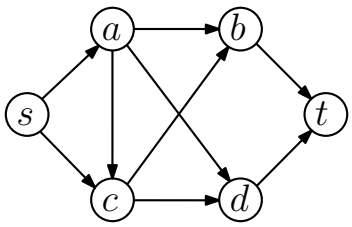
4. **Minimum Spanning Tree** (10 pts)

   Let $G = (V, E)$ be a weighted, undirected, connected graph, whose weights are integers from $\{1, \ldots, k\}$. Design an algorithm to compute the minimum spanning tree of G in $O(|E| \log k)$ time. [Hint: Modify Prim's algorithm.]

5. **Counting Paths** (12 pts)

   Let $G = (V, E)$ be a connected, directed, acyclic graph (DAG). Let s, t be two vertices from G. Please design an algorithm that counts the number of different paths from s to t in G. For full credits, your algorithm should run in $O(|E|)$.

   For example, the graph below has 6 different paths from s to t, as follows. (Your algorithm just needs to output the number 6, not the actual paths.)



1. $s \to a \to b \to t$;
2. $s \to a \to c \to b \to t$;
3. $s \to a \to c \to d \to t$;
4. $s \to a \to d \to t$;
5. $s \to c \to b \to t$;
6. $s \to c \to d \to t$;

6. **Consulting Firm** (20 pts)

   Suppose you're running a lightweight consulting firm with a few friends. Each month, you can either run your business from an office in Hong Kong or from an office in Beijing. Since your clients travel a lot, the demand varies from month to month, and the profits are different if your office is in different cities. In month i, let the profit in Hong Kong be H_i, and the profit in Beijing be B_i. But every time you switch cities, you incur a fixed moving cost M. Given the H_i and B_i values for a sequence of n months, we would like to find the optimal plan for the locations where you run your business for these n months that maximizes your net profit (i.e., total profit minus the moving costs). Suppose you start your business in Hong Kong in month 1.

   Example: Suppose $n = 4, M = 10$, and the profits are given by the following table:

	Month 1	Month 2	Month 3	Month 4
HK	80	90	30	20
BJ	20	30	80	70

   Then the optimal plan would be the sequence [HK, HK, BJ, BJ], with a net profit of $80 + 90 - 10 + 80 + 70 = 310$.

   (a) (4 pts) The following greedy strategy looks rather reasonable: If in the next month, the profit in the other city is M more than the profit of the current city, then we switch. More precisely, if we are currently in HK in month i, and $B_{i+1} - H_{i+1} > M$, we move to Beijing for month $i + 1$; if we are currently in BJ in month i, and

$H_{i+1} - B_{i+1} > M$, we move to Hong Kong for month $i+1$. This greedy algorithm, however, is not always correct. Please give an example in which this algorithm returns a non-optimal plan.

(b) (10 pts) Design and analyze an algorithm that always finds the optimal plan. For full credits, your algorithm should run in $O(n)$ time. Your algorithm should return both the maximum net profit and the optimal plan that achieves this profit.

(c) (6 pts) Now consider the case that your business may incur a loss (i.e., negative profit) for certain months due to bad economy. Further suppose that you have the freedom of choosing a starting month and a finishing month. Please design an algorithm that finds the best time to start and finish your business given the profit table and M. Your algorithm just needs to output the optimal net profit, not the actual plan or the starting/finishing month. Your starting city is still Hong Kong. You may assume that the profit is positive for at least one month in HK (otherwise you may not want to start the business at all).

Example: Suppose $n = 5, M = 10$, and the profits are given by the following table:

	Month 1	Month 2	Month 3	Month 4	Month 5
HK	-10	30	-30	40	-5
BJ	20	30	-5	10	5

Then the optimal plan is to start in month 2 and finish in month 4, with the sequence [HK, BJ, HK]. The net profit is $30 - 10 - 5 - 10 + 40 = 45$.

For full credits, your algorithm should run in time $O(n)$. Note that if $M = \infty$, then you wouldn't want to move at all, and this problem degenerates to the *maximum contiguous subarray problem*. In this case, your algorithm will actually beat the $O(n \log n)$-time algorithm given in class!

7. **Greedy Algorithm** (10 pts)

Given two sequences $X = (x_1, \ldots, x_m)$ and $Y = (y_1, \ldots, y_n)$, design an algorithm that determines if X is a subsequence of Y. You can assume $m \leq n$. For example, if $X =$ `BCBA` and $Y =$ `ABCBDAB`, then the answer is "yes". But if $X =$ `BDBA`, then the answer is "no". For full credits your algorithm should run in $O(n)$ time. Remember to argue for the correctness of your algorithm unless it is obvious.

8. **Huffman Coding** (10 pts)

Please design a Huffman code for the following alphabet of 7 letters:

Character	Number of occurrences
a	34
b	22
c	20
d	34
e	100
f	8
g	12