COMP 3711 Design and Analysis of Algorithms
Fall 2014 Final Exam

1. **Time Complexity** (8 pts)

   We have two algorithms, A and B. Let $T_A(n)$ and $T_B(n)$ denote the time complexities of algorithm A and B respectively, with respect to the input size n. Below there are 8 different cases of time complexities for each algorithm. Complete the last column of the following table with "A", "B", or "U", where:

   - "A" means that algorithm A is faster;
   - "B" means that algorithm B is faster;
   - "U" means that we do not know which algorithm is faster.

Case	$T_A(n)$	$T_B(n)$	Faster
1	$\Theta(n^2)$	$\Theta(n^2/\log n)$	
2	$O(n^3)$	$\Omega(2^{\sqrt{n}})$	
3	$\Theta(\log n)$	$O(\log\log n)$	
4	$\Theta(\log^3 n)$	$\Theta(\sqrt[3]{n})$	
5	$O(n^2)$	$O(n^{2.31})$	
6	$\Omega(n^2)$	$O(n^{2.5})$	
7	$\Omega(n^3)$	$O(n^{2.81})$	
8	$\Theta(\sqrt{\log n})$	$\Theta(\log n/\log\log n)$	

2. **Divide and Conquer** (12 pts)

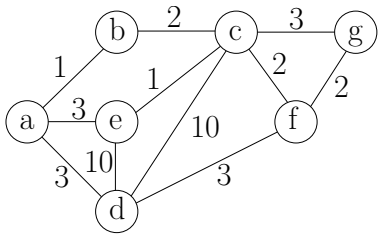
   You are given an array $A[1..n]$ that contains a sequence of 0 followed by a sequence of 1 (e.g., 0001111111). A contains at least one 0 and one 1.

   (a) (5 pts) Design an $O(\log n)$-time algorithm that finds the position k of the last 0, i.e., $A[k] = 0$ and $A[k+1] = 1$.

   (b) (7 pts) Suppose you have been told that k is much smaller than n. Design an $O(\log k)$-time algorithm that finds the position k of the last 0. [Hint: you can re-use the solution of part (a).]

3. **Prim vs. Dijkstra** (16 pts)

   Prim's minimum spanning tree algorithm and Dijkstra's shortest path algorithm are very similar, but with crucial differences. Run both algorithms on the following graph, and show the partial MST / shortest path tree after every new edge has been added. The starting vertex for both algorithms is "a".



4. **Minimum Spanning Tree** (20 pts)

   All MST algorithm studied in class use the notion of "safe" edges, and the MST lemma shows that any safe edge can be added to a partial MST. There is an alternate formulation. Let G be a weighted undirected graph, where the edge weights are distinct. We say that an edge e is *dangerous* if it is the longest edge in ***some*** cycle in G.

   (a) (10 pts) Prove that no MST of G can contain a dangerous edge.

   (b) (10 pts) Given the graph G and an edge e, design and analyze an algorithm to check whether e is dangerous. For full credits, your algorithm should run in $O(V + E)$ time.

5. **Greedy Algorithm** (15 pts)

   In the old days, files were stored on tapes rather than disks. Reading a file from tape isn't like reading a file from disk; first we have to fast-forward past all the other files, and that takes a significant amount of time. Suppose we have a set of n files that we want to store on a tape, where file i has length $L[i]$. Given the array $L[1..n]$, your job is to design an algorithm to find the optimal order to store these files on a tape to minimize the cost. Note that the cost of reading file i is total length of all files stored before it, including file i itself. Your algorithm should run in $O(n \log n)$ time.

   (a) (5 pts) Suppose each file is accessed with equal probability, and you want to minimize the expected cost. For example, if $L[1] = 3, L[2] = 6, L[3] = 2$, you would want to use the order $(3, 1, 2)$. This way, the expected cost is $2/3 + (2+3)/3 + (2+3+6)/3 = 6$, which is optimal. You need to prove the optimality of your algorithm.

   (b) (10 pts) Suppose the files are not accessed uniformly; file i will have probability $p[i]$ to be accessed. Given the array $L[1..n]$ and $p[1..n]$, how would you find an ordering that minimizes the expected cost? For example, if $L[1] = 3, L[2] = 6, L[3] = 2$ and $p[1] = 1/6, p[2] = 1/2, p[3] = 1/3$, then the optimal ordering would be $(3, 2, 1)$, with an expected cost of $2/3 + (2 + 6)/2 + (2 + 6 + 3)/6 = 6.5$. Remember to prove the optimality of your algorithm.

6. **Recurrences and Dynamic Programming** (14 pts)

   Consider the following recursive algorithm RA, which takes as input a positive integer n.

```
RA(n):
    if n = 1 then return 10
    s ← 5
    for i ← 1 to n − 1 do
        s ← s + 3 · RA(i)
    return s
```

(a) (7 pts) Describe the recurrence for the running time $T(n)$ of RA, and use the substitution method (i.e., induction) to prove that $T(n) = O(2^n)$.

(b) (7 pts) Provide the pseudocode of a dynamic programming algorithm that achieves the same result as RA and analyze its running time.

7. **Maximum Independent Set** (15 pts)

In class we learned about the maximum independent set problem on an array, where we want to pick a subset of non-adjacent numbers from an array to maximum their sum. Here we generalize this problem to a binary tree. More precisely, given a binary tree where each node has a number, your job is to design an algorithm to pick a subset of nodes to maximize their sum. Still, adjacent nodes cannot be picked. The figure below shows an example where the grayed nodes form the optimal solution. You will get full credits if your algorithm runs in $O(n)$ time, where n is the number of nodes in the binary tree. Your algorithm just needs to output the sum, not the actual nodes picked.