Algorithm Final Exam (Spring 2013)

總分：125 %

Name: _____

Student ID #: _____

Question	Score
1 (25%)	
2 (5%)	
3 (15%)	
4 (10%)	
5 (10%)	
6 (35%)	
7 (15%)	
7 (10%)	
Total	

1. **[Simple questions: 25%]** Only simple answers are necessary for the following questions.
   (a) (Yes/No) A sequence of strictly decreasing order is always a max-heap. (2%)

   (b) (Yes/No) Find the top-ten elements in an un-sorted array can be done in less than $\Theta(n \lg n)$ time. (2%)

   (c) (Yes/No) In a directed graph, we need a *queue* for its breadth-first search. (2%)

   (d) Rearrange the following functions from the smallest complexity one to the largest complexity one: (i) $n / \log n$, (ii) n^2, (iii) 2^n, (iv) $n!$, (v) $\log (n!)$ (4%)

   (e) What is the space that quicksort uses in each sorting of n elements? (4%)

(f) Give an example of graph when its BFS and DFS can be the same. (4%)

(g) List any thing we need to consider when we have the deletion operation on an open-address hash table. (5%)

(h) How do you think of this course, too easy, too difficult, you learned anything interesting, comments? (2%)

2. **[Recurrence equation: 5%]** Solve the following recurrence equation.

   $T(n) = 16T(n/4) + n^3 \lg n$

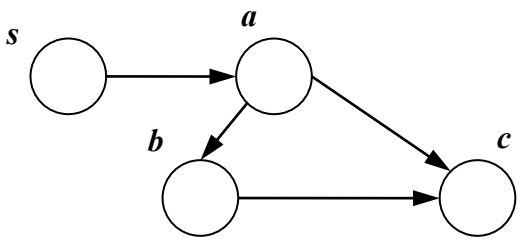
   Your answer should be as tight as possible. You can use master theorem when there is a need.

3. **[Dynamic Programming: 15%]** Given six matrices A_1, … , A_6 with their dimensions equal to 1×200, 200×3, 3×100, 100×2, 2×500, 500×4, find the most efficient way to compute the product of $A_1 A_2 A_3 A_4 A_5 A_6$. You should write down your answer by putting full parentheses to the product.

4. **[Dynamic Programming 10%]** Give an $O(n^2)$-time algorithm to find the longest monotonically increasing subsequence of a sequence 32457451. Discuss two cases separately: (1) simple increasing subsequences (2) strictly increasing subsequences. (hint: sorting)

5. **[Graph: 10%]** Given an adjacency-list representation of a directed graph, how long does it take to compute the out-degree of a vertex? How long does it take to compute the in-degrees of a vertex? How about the case of adjacency matrix?

6. **[DFS & BFS 35%]** Given the following graph, answer the questions related to DFS and BFS. You need to explain your answer for all the questions.



(a) List all possible results of BFS. (6%)

(b) How many times DFS will execute line 7 in the DFS(G) code given below? Give all the possible answers. (5%)

(c) What edge can be non-tree edge in some execution of DFS? (4%)

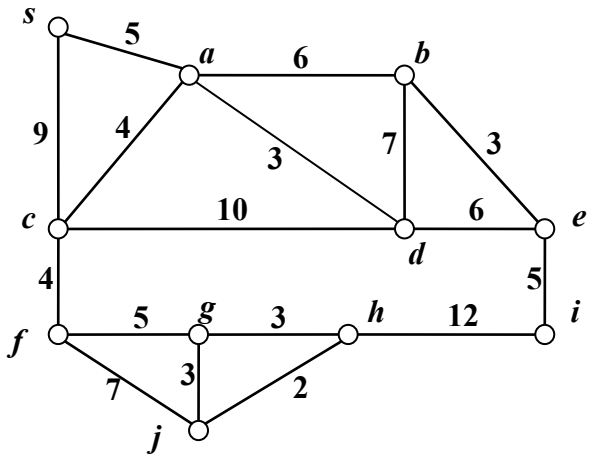
(d) List all possible results of *topological sort*. Give the finish time f [] of all vertices for each of your result. (6%)

(e) Given a graph G, what is the time complexity to build G^T, the transpose graph of G when the graph structure is an adjacency list? (5%)

(f) Following part (e), but for adjacency matrix? (5%)

(g) Add as few edges as possible to make the graph to contain only one *strongly connected component*? What is/are the edge(s)? (4%)

7. **[Minimum Spanning Tree: 15%]** Given the following graph, answer the questions related to minimum spanning tree.



(a) Find the MST via the Kruskal's and Prim's algorithms (starting from the vertex *s*). Other than giving the MST, you should also list the edges in order in discovering your MST. (10%)

(b) If we focus on vertex *d*, we have four edges linked it to four other vertices. Can you conclude that edge *ad* must be included in a MST? Explain your answer. (5%)

8. **[Stable Marriage Problem: 10%]** Find a stable match for the following bipartite graph with four boys (*A*, *B*, *C*, *D*) and four girls (1, 2, 3, 4) and their preference.
   A: 3, 1, 2, 4
   B: 4, 2, 1, 3
   C: 1, 4, 2, 3
   D: 4, 1, 2, 3

   1: *A*, *B*, *C*, *D*
   2: *B*, *C*, *A*, *D*
   3: *C*, *A*, *D*, *B*
   4: *C*, *D*, *B*, *A*

Below are some questions for graphs, BFS, DFS and minimum spanning tree. We list several codes for your reference.

```
BFS(G, s)
1   for each vertex u ∈ V(G) − {s}
2         do color[u] ← WHITE
3            d[u] ← ∞
4            π[u] ← NIL
5   color[s] ← GRAY
6   d[s] ← 0
7   π[u] ← NIL
8   Q ← {s}
9   while Q ≠ ∅
10        do u ← DEQUEUE(Q)
11              for each v ← Adj[u]
12                    do if color[v] = WHITE
13                          then color[v] ← GRAY
14                               d[v] ← d[u] + 1
15                               π[v] ← u
16                               ENQUEUE(Q, v)
17        color[u] ← BLACK
```

```
DFS(G)
1   for each vertex u∈V[G]
2         do color[u] ← WHITE
3            π[v] ← NIL
4   time ← 0
5   for each vertex u∈V[G]
6         do if color[u] = WHITE
7               then DFS-VISIT(u)

DFS-VISIT(u)
1   color[u] = GRAY
2   d[u] ← time ← time +1
3   for each v∈Adj[u]
4         do if color[v] = WHITE
5               then π[v] ← u
6                    DFS-VISIT(v)
7   color[u] = BLACK
8   f[u] ← time ← time + 1
```

```
MST-KRUSKAL(G, w)
1  A ← ∅
2  for each vertex v∈V[G]
3       do MAKE-SET(v)
4  sort the edge of E into nondecreasing order by weight w
5  for each edge (u, v) ∈ E, taken in nondecreasing order by w
6       do if FIND-SET(u) ≠ FIND-SET(v)
7            then A ← A ∪ {(u, v)}
8                 UNION(u, v)
9  return A
```

```
MST-PRIM(G, w, r)
1   for each u ∈ V[G]
2         do key[u] ← ∞
3            π[u] ← NIL
4   key[r] ← 0
5   Q ← V[G]
6   while Q ≠ ∅
7         do u ← EXTRACT-MIN(Q)
8               for each v ∈ Adj[u]
9                     do if v ∈ Q and w(u, v) < key[v]
10                          then π[v] ← u
11                               key[v] ← w(u, v)
```
