Algorithms
Comprehensive Exam
(Fall 2018)

SHORT QUESTIONS (Answer any five questions, each carrying 8 points.)

1. The divide and conquer strategy (D&C) has been used to solve problem effectively to reduce the overall computational cost to certain types of problems.
   (a) Which conditions have to be satisfied for D&C to solve such problems successfully? (Clearly state)
   (b) Suppose the size of a problem involved in D&C is $n = 2^k$. Let the cost in dividing the problem into an equal size is constant and the time to combine solutions to sub-problems is linear. Write the recurrence relations and then find the tight bound in solving such problems using D&C.

2. (a) What is the lower bound for comparisons based sorting algorithm? (Outline the justification of your answer.)
   (b) What is the strategy behind greedy algorithm?

3. Find the tight bounds (by deriving their upper and lower bounds) of following expressions.
   (a) $T(n) = 2 \cdot T(n/8) + n^{1/3}$.
   (b) $T(n) = \log(n!)$.

4. Briefly describe what is dynamic programming. Can one apply it to solve any optimization problem? If yes explain why; if not, what particular conditions must be met.

5. The utilization efficiency of a hash table depends heavily on its hashing function(s) employed. Describe with a diagram to illustrate how a multiplication method of hashing functions works on a machine with the word size of w bits for a hash table with 2^p entries, $p < w$.

6. Show your construction of an optimal Huffman code for the set of 7 frequencies: **a**: 3, **b**: 12, **c**: 5, **d**: 20, **e**: 16, **f**: 34, **g**: 18.

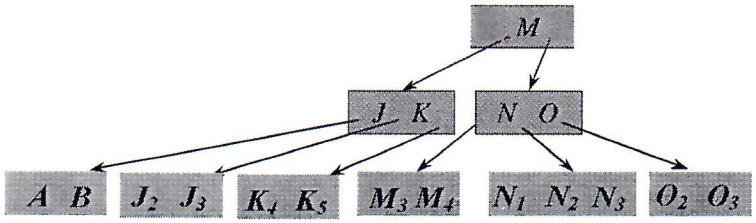
LONG QUESTIONS (Answer any four questions, each carrying 15 points.)

1. Describe how dynamic programming is used to construct an optimal binary search tree.

   Using the following probability of p_i and q_i, obtain the expected cost of searching an optimal binary search tree constructed by dynamic programming.

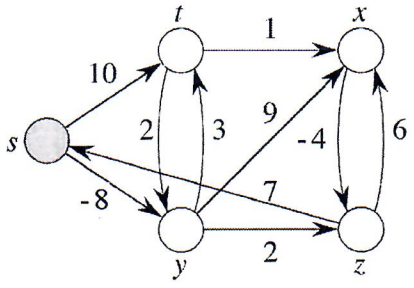
i	0	1	2	3	4	5	6	7
p_i		0.04	0.06	0.08	0.06	0.1	0.12	0.1
q_i	0.06	0.06	0.06	0.06	0.05	0.05	0.05	0.05

2. Given the initial B-tree with the minimum node degree of $t = 3$ below, show the results (a) after deleting the key of K , (b) followed by inserting the key of L, (c) then by deleting the key of J_2, (d) then by inserting the key of N_4, with $N_3 < N_4 < O$, and (e) then by deleting K_4. (Show the result after each deletion and after each insertion.)



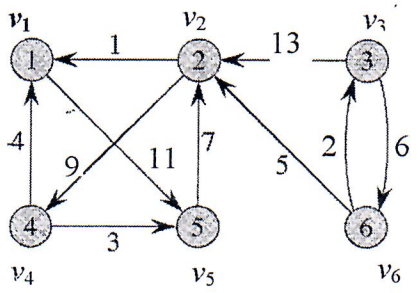
3. Follow depth-first search (*DFS*), starting from Node s, to traverse the graph shown below, with its edge weights all ignored and the start time equal to 1. Mark (1) the type of every edge and (2) the discovery and the finish times of each node.

   Follow breadth-first search (*BFS*), starting from Node s, to traverse the graph shown below, with its edge weights all ignored. Show the predecessor tree rooted at Node s after *BFS*, with the number of links (i.e., distance) from Node s to every other node indicated.



4. The Dijkstra's algorithm (*DS*) solves the single-source shortest-path problem in a weighted directed graph $G = (V, E)$ without negative weighted edges or cycles, by edge relaxation at one vertex at a time until all vertices are examined. Given the graph G below, follow *DS* to find shortest paths from vertex v_1 to all other vertices, with all predecessor edges shaded and estimated distance values from v_1 to all vertexes provided at the end. Also list the sequence of vertices at which relaxation takes place.

   What is the time complexity of *DS* for a general graph $G = (V, E)$, when candidate vertices are kept in an array?



5. Given the matrix-chain multiplication problem for four matrices sized 30×24, 24×15, 15×20, 20×50, follow the tabular, bottom-up method in the procedure of MATRIX-CHAIN-ORDER below to construct two tables of $m[i, j]$, for all $1 \le i, j \le 4$, and $s[i, j]$, for all $1 \le i \le 3$ and $2 \le j \le 4$. Construct the two tables, with their entry values shown.

MATRIX-CHAIN-ORDER(p)

```
1  n = p.length − 1
2  let m[1..n, 1..n] and s[1..n − 1, 2..n] be new tables
3  for i = 1 to n
4      m[i, i] = 0
5  for l = 2 to n                    // l is the chain length
6      for i = 1 to n − l + 1
7          j = i + l − 1
8          m[i, j] = ∞
9          for k = i to j − 1
10             q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
11             if q < m[i, j]
12                 m[i, j] = q
13                 s[i, j] = k
14 return m and s
```