Data Structures Final Examination

3:30pm-5:20pm (110 minutes), Monday, June 22, 2015

1.

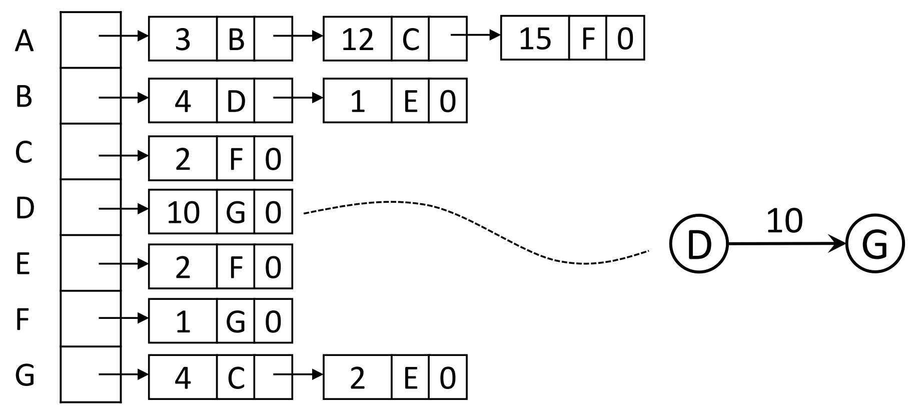
A. Please perform least significant digit (LSD)-first radix sort (radix = 10) over the following numbers in **non-decreasing** order. (5%)

340		340		001		001
135		620		505		135
620		140		620		140
001	→	001	→	324	→	324
140		324		135		340
324		135		340		365
365		365		140		505
505		505		365		620

B. Please perform LSD-first radix sort (**radix = 2**) over the following numbers in **non-decreasing** order. (5%)

15		1111		0110		0100		1001		0011
6		0110		0100		1001		1010		0100
4		0100		1010		1101		0011		0110
9	→	1001	→	1111	→	0110	→	0100	→	0111
7		0111		1001		1010		1101		1001
3		0011		0111		1111		0110		1010
13		1101		0011		0111		1111		1101
10		1010		1101		0011		0111		1111

2. Given a weighted directed graph whose adjacency list representation is as follows



A. Please complete the following table to perform single-source, all-destinations Dijkstra's algorithm starting from vertex A. Please use parenthesis to denote a dist[] value that is not touched and use "—" to denote that a vertex is already selected. (5%)

Iteration	dist[]						Selected Destination	Path Cost
	B	C	D	E	F	G		
1	3	12	∞	∞	15	∞	B	3
2	—	(12)	7	4	(15)	(∞)	E	4
3	—	(12)	(7)	—	6	(∞)	F	6
4	—	(12)	(7)	—	—	7	D	7
5	—	(12)	—	—	—	7	G	7
6	—	11	—	—	—	—	C	11

or

Iteration	dist[]						Selected Destination	Path Cost
	B	C	D	E	F	G		
1	3	12	∞	∞	15	∞	B	3
2	—	(12)	7	4	(15)	(∞)	E	4
3	—	(12)	(7)	—	6	(∞)	F	6
4	—	(12)	(7)	—	—	7	G	7
5	—	11	(7)	—	—	—	D	7
6	—	11	—	—	—	—	C	11

B. Please complete the following graph-traversal sequences of the graph. If there are multiple valid traversals, just list one of them. Mark an "X" in a field where the traversal cannot continue. (5%)

Depth-first traversal: A, C, F, G, E, B, D

Breadth-first traversal: A, B, F, C, (D, E), G (D and E can exchange)

or A, B, F, C, G, (D, E) (D and E can exchange)

Topological traversal: A, B, D, X

3. Please fill in the following tables to perform Quick Sort. Table 1 shows **Basic Quick Sort** that always takes the **left-most** key of a list/sublist as the pivot (5%). Table 2 shows **Ideal Quick Sort** in which the selected pivot always ideally splits a list/sublist into equal halves, (i.e., 3, 1, and 5 are sequentially selected as pivots) (5%). Please note that common practices always use a swap to move the pivot to the left-most position if the pivot is elsewhere (e.g., the first swap in Table

2).

Pivot	Keys						
1	1	5	4	2	3	0	6
	1	0	4	2	3	5	6
4	0	1	4	2	3	5	6
3	0	1	3	2	4	5	6
5	0	1	2	3	4	5	6

Pivot	Keys						
3	1	5	4	2	3	0	6
	3	5	4	2	1	0	6
	3	0	4	2	1	5	6
	3	0	1	2	4	5	6
1	2	0	1	3	4	5	6
	1	0	2	3	4	5	6
5	0	1	2	3	4	5	6
	0	1	2	3	5	4	6
	0	1	2	3	4	5	6

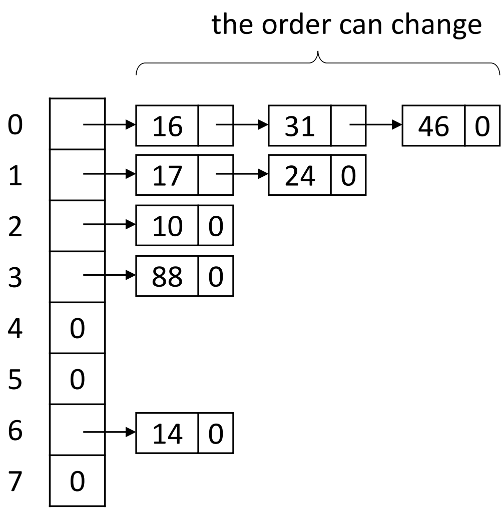
4. Please consider inserting the keys 46, 24, 31, 10, 14, 16, 17, 88 into a hash table. The hash function h(k) = (k mod 23).

	46	24	31	10	14	16	17	88
h(k) = k % 23	0	1	8	10	14	16	17	19
h(k) % 8	0	1	0	2	6	0	1	3
h(k) % 16	0	1	8	10	14	0	1	3

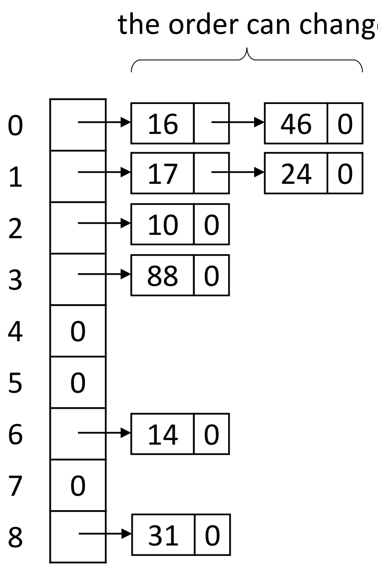
A. Please show the result if we use an **eight-bucket table**, **single-slot buckets**, **three least-significant bits of h(k)**, (i.e., h(k) mod 8), and **linear probing**. (5%)

0	46
1	24
2	31
3	10
4	16
5	17
6	14
7	88

B. Please show the result if we use an **eight-bucket table**, **three least-significant bits of h(k)**, and **chaining**. (5%)



C. Please show the result if **chaining** and **directory-less dynamic hashing** is used and the number of buckets increases from eight to nine after 88 is inserted. (5%)



5. We want to perform heap sort to sort an array “5, 19, 25, 15, 20, 16, 10, 30” into non-decreasing order. The first phase of heap sort is to in-place heapify the array as a **max heap**.

A. Please list the array contents after heapification completes if we use a **binary heap**, in which every parent have two children. (5%)

30	20	25	19	5	16	10	15

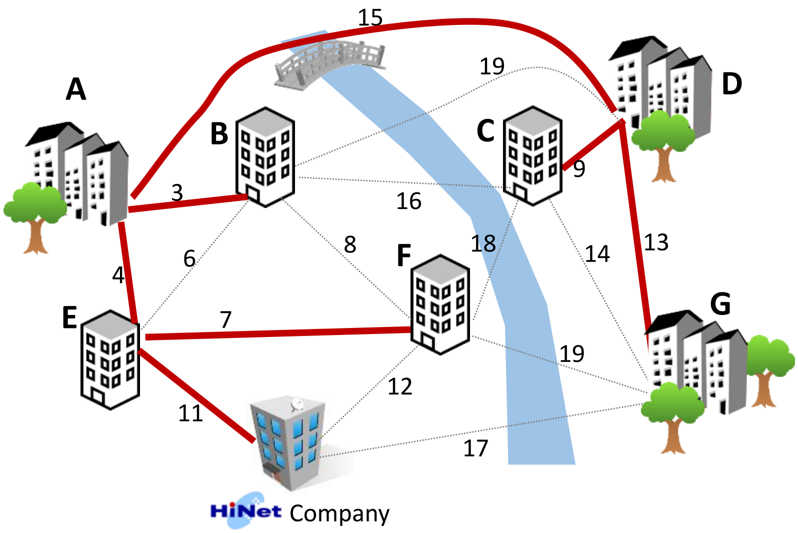
B. Please list the array contents after heapification completes if we use a **ternary heap**, in which every parent have three children. (5%)

30	20	25	15	19	16	10	5

C. Please list the array contents after the sorting algorithm pops the top element from the max heap and places the element at the end of the array (considering a **binary heap**) (5%)

25	20	16	19	5	15	10	30

6. HiNet company wants to use a fiber network to connect eight buildings, A to G, and the HiNet building, together. The candidate fiber routes and the corresponding cost (in the unit of 100,000 NT$) are shown as follows.



A. Please mark the fiber network that has the minimum total cost. (5%)

B. Please describe an algorithm that can find the network with **the second minimum** total cost (5%).

One algorithm is as follows

1) Find an MST, T, in the graph. Using any algorithm such as Kruskal's algorithm is good. We refer to edges that are not part of the MST as non-tree edges.
2) For each non-tree edge, add the non-tree edge into T to form T'. T' must contain exactly one cycle.
3) Remove an edge that is of the second largest cost in the cycle to form T''
4) By separately adding each non-tree edge to T, we get several different T''. The T'' with the minimum cost is a second-minimum cost spanning tree.

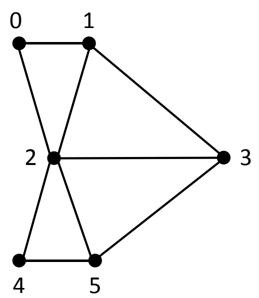
7. omitted

8.

A. Please design an algorithm (using pseudo code) that takes an undirected graph in adjacency matrix representation as input and determines whether an Eulerian path exists. An Eulerian path is a path in a graph which visits each edge exactly once. A graph has an Eulerian path if and only if the number of vertices that have odd degree is either zero or two. (5%)

```
int nOdd = 0;
int degree;
for (int u = 0; u<n; u++){
   degree = 0;
   for (int v=0; v<n; v++){
      if (input[u][v] == 1)    degree ++;
   }
   if (degree%2 == 1) // odd
      nOdd ++;
   if (nOdd > 2) return false; // no Eulerian paths
}
if (nOdd == 1) return false; // no Eulerian paths
else return true;    // nOdd is zero or two, Eulerian paths exist
```

B. Please use an adjacency matrix to represent the following undirected graph. (5%)



→

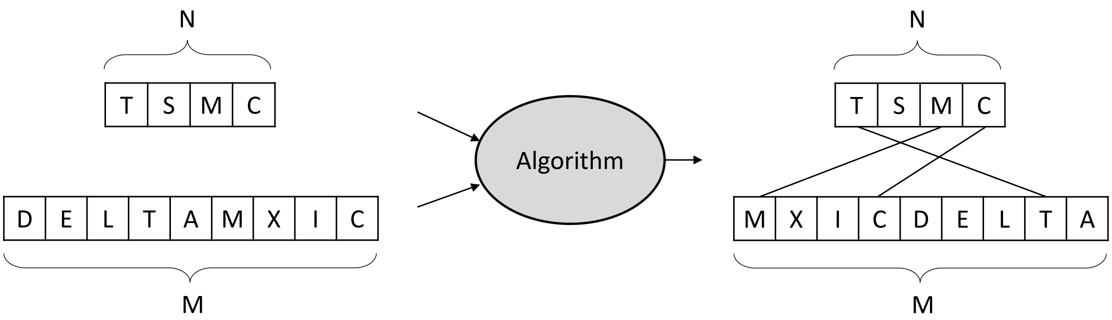
	0	1	2	3	4	5
0	0	1	1	0	0	0
1	1	0	1	1	0	0
2	1	1	0	1	1	1
3	0	1	1	0	0	1
4	0	0	1	0	0	1
5	0	0	1	1	1	0

9. Please perform decision tree based algorithm analyses.

A. Please prove that any comparison-based algorithm requires log(N!) comparisons in the worst case to sort an N-element list. (5%)

Given N elements to sort, there are a total of N! possible outcomes. Therefore, the decision tree representation of any sorting algorithm must have N! outcomes, too. A decision tree of height k corresponds to at most 2^k outcomes. Therefore, the decision tree representation of sorting N elements must be at least of height log(N!). Any algorithm must perform log(N!) comparisons to sort N elements in the worst case.

B. Please derive the lower bound of the worst-case number of comparisons any comparison-based algorithm requires to pair two lists, one with N distinct keys and the other with M distinct keys (N<M). The following graph shows exampling inputs and outputs of such a pairing algorithm. (5%)



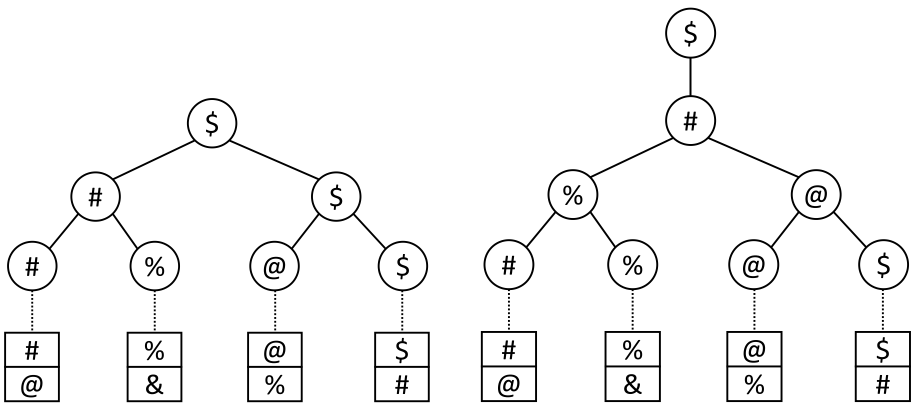
The number of possible outcomes of pairing two lists, respectively having N and M distinct keys, N<M, is a product of N terms $(M+1) \times (M) \times (M-1) \ldots \times (M-N+2)$. The first term is (M+1) because the first key in the first N-key list can be paired to either one on the M possible keys in the second list or nothing.

Please note that 2NM and NM overestimate and underestimate the number of possible outcomes, respectively.

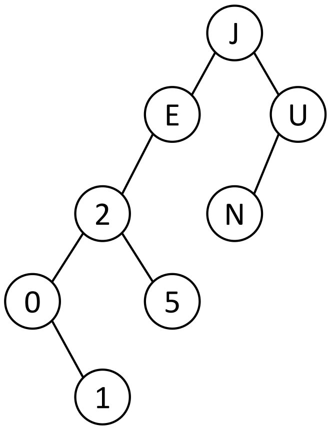
Therefore, similar to the analysis in question A, the worst-case number of comparisons required by any paring algorithm is $\log((M+1) \times (M) \times (M-1) \ldots \times (M-N+2)) \cong N \times \log(M)$

10.

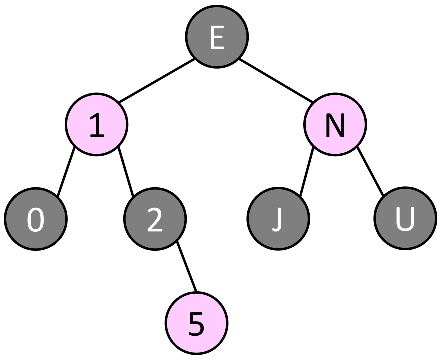
A. Please complete the following winner and loser trees (5%).



B. Please plot the result of sequentially inserting eight letters, “J U N E 2 0 1 5”, into an empty, **standard binary search tree**. (0 < 1 < 2 < 5 < E < J < N < U) (5%)



C. Please plot the result of sequentially inserting eight letters, “J U N E 2 0 1 5”, into an empty, **red-black tree**. (0 < 1 < 2 < 5 < E < J < N < U) (Hint: Check whether two consecutive red nodes appear after insertion; if so, check whether the uncle node is red or black; perform rotation or color changes accordingly; always color the root node black.) (5%)



Thank you for your participation during the class.

Best wishes in your future studies!