Data Structure 2023

U2287

National Taipei University

Final Exam

1. Please traverse the tree shown in Fig. 1a using **preorder** and **inorder** (8%)

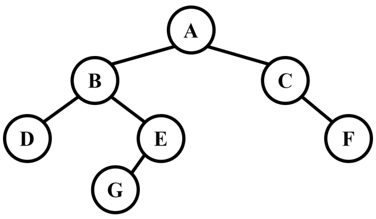


Fig.1a: tree traversal (Q1)

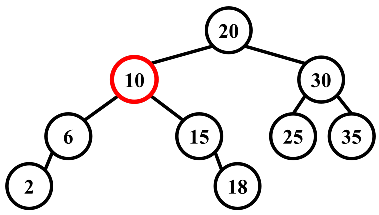
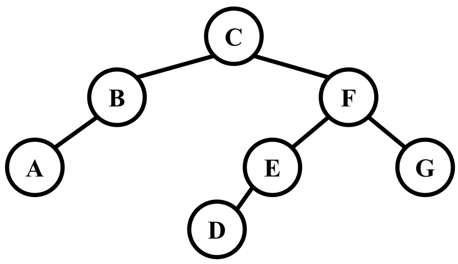


Fig.1b: binary search tree deletion (Q2)

Preorder: A → B → D → E → G → C → F
Inorder: D → B → G → E → A → C → F

2. Please describe the algorithm for **deleting a non-leaf node with two children** in a **binary search tree** using the **node 10** in Fig. 1b as an example (6%)
We can swap the node 10 with its successor: the minimum node of the right subtree (or the maximum node of the left subtree) (3%). After that, we can simply cut and reconnect the rest of the tree (2%) because the successor must have a degree of 1 or 0 (1%).

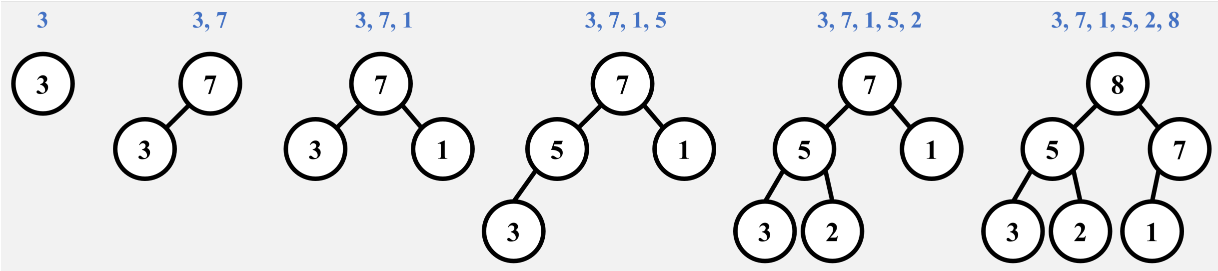
3. A **binary search tree** can be uniquely decided given its **preorder** or **postorder** traversal result. Please draw the tree if its **preorder** is C → B → A → F → E → D → G (6%)
The answer is shown in below figure:



4. Please answer the following questions about **Heap**:

   (a) Explain which data structure (array or pointer) is more suitable to implement a heap. (4%)
   Array is more suitable. Heap is a complete binary tree so no space will be wasted when using array (2%). Most importantly, using arrays allows faster access to the parent and children of a node, which is cruical for heap operations (2%).

   (b) Please build a max heap for the input sequence: 3, 7, 1, 5, 2, 8. You should draw the tree after **each** number is inserted into the heap (6%)
   The answer is shown in below figures: 1% for each figure



5. Please transform the **forest** in Fig. 2a into a single **binary tree** using the **left-child-right-sibling** approach. (6%)

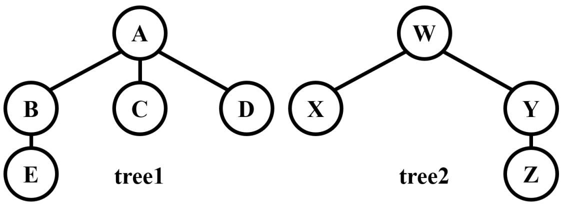


Fig.2a: forest (Q5)

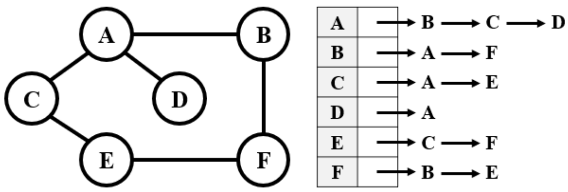
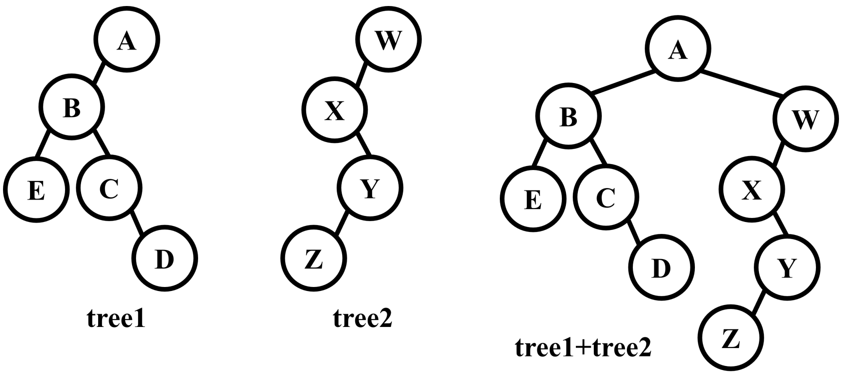


Fig.2b: graph traversal (Q6)

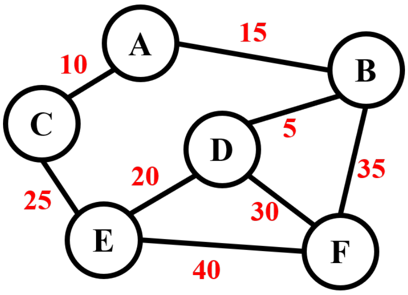
The answer is shown in below figures: drawing correct tree1+tree2 can get 6%. If tree1+tree2 is incorrect, any correct figure of tree1 and tree2 can get 2%

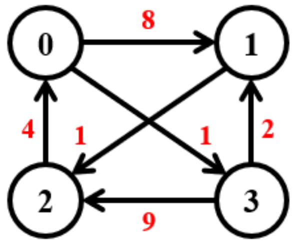


6. Traverse the graph (starting from **A**) shown in Fig. 2b using **depth-first search (DFS)** and **breadth-first search (BFS)**. The traversal order should depend on the adjacency list (8%)
DFS: A → B → F → E → C → D
BFS: A → B → C → D → F → E

7. Consider an array of 8 elements being sorted using quicksort. It has just finished the first pass of partitioning and pivot swapping, thus the original array into [7, 11, 16, 10, 17, 1, 18, 30]. Write down all possible elements that could have been the pivot in the first pass (6%)
18, 30

8. Please describe how to use **straight-radix sort (sorting from the least significant bit)** to sort the following numbers: 63, 49, 783, 7, 543, 132, 898 (6%)
First run: sort with the digits: 132 → 63 → 783 → 543 → 7 → 898 → 49
Second run: sort with the ten digits while preserving the previous order of digits: 7 → 132 → 543 → 49 → 63 → 783 → 898
Third run: sort with the hundred digits while preserving the previous orders of ten digits and digits: 7 → 49 → 63 → 132 → 543 → 783 → 898





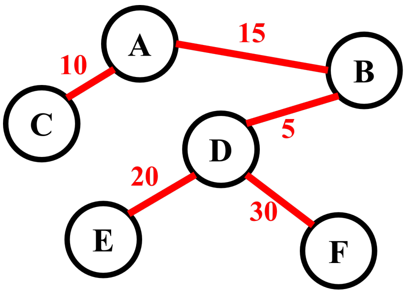
to

from	0	1	2	3
0	0	8	∞	1
1	∞	0	1	∞
2	4	∞	0	∞
3	∞	2	9	0

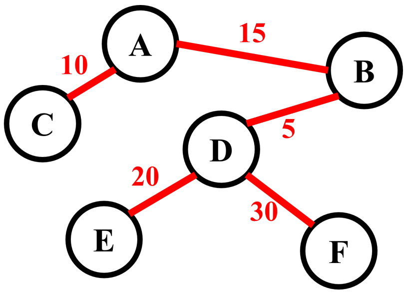
A^{-1}

Fig.3a: minimum cost spanning tree (Q9) Fig.3b: shortest paths (Q10)

9. Please find the minimum cost spanning tree of the graph shown in Fig. 3a using (a) **Kruskal's algorithm** and (b) **Prim's algorithm** (starting from **A**). You can just draw the results. (8%)
   The answer is shown in below figures: 4% for each figure



Kruskal



Prim

10. Consider the directed graph shown in Fig. 3b, answering the following questions:
    (a) Write down each step of **Dijkstra's algorithm** to find the shortest path from vertex **0** to vertex **2**. (6%)
    The answer is shown in below figures: 2%, 2%, 1%, 1% for the four passes

Pass	Explored Set	0	1	2	3
1	{0}	0	8	∞	1
2	{0, 3}	0	3	10	1
3	{0, 3, 1}	0	3	4	1
4	{0, 3, 1, 2}	0	3	4	1

    (b) The matrix A^{-1} shows the initial values to find all-pairs shortest paths with **Floyd-Warshall's algorithm**. Derive A^0, A^1, A^2, and A^3 (8%) **The answer is shown in below figures: 2% for each pass**

	0	1	2	3
0	0	8	∞	1
1	∞	0	1	∞
2	4	12	0	5
3	∞	2	9	0

A^0

	0	1	2	3
0	0	8	9	1
1	∞	0	1	∞
2	4	12	0	5
3	∞	2	3	0

A^1

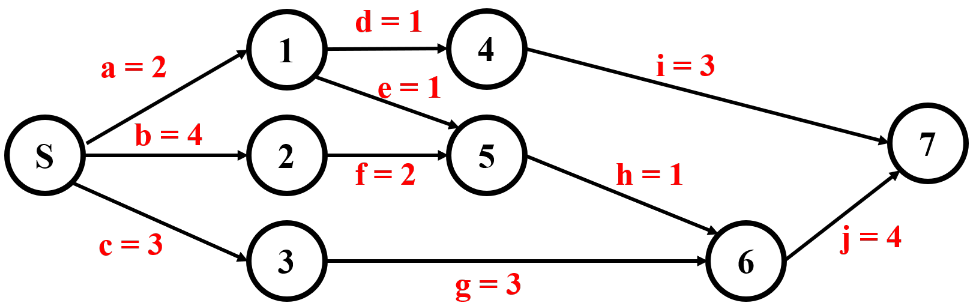
	0	1	2	3
0	0	8	9	1
1	5	0	1	6
2	4	12	0	5
3	7	2	3	0

A^2

	0	1	2	3
0	0	3	4	1
1	5	0	1	6
2	4	7	0	5
3	7	2	3	0

A^3

11. The following figure shows an **active-on-edge (AOE)** network, please determining the earliest completion time of **node5**, **node6**, and **node7** (6%)



node5: 6 (2%); node6: 7 (2%); node7: 11 (2%)

12. The following figure shows a **10-bucket hash table** implemented with a **circular array** and each bucket can store only **one** record. A hash function $h(K) = K\%N$ is used with N being the number of buckets. To handle collision, **quadratic probing** with the following modified hash function $H_i(K) = (h(K) + i^2)\%N$ is used with i being the times of collisions. Please insert the following numbers into the hash table in order: 38, 29, 58, 49, 69, 18, 75 (6%)

								38	
0	1	2	3	4	5	6	7	8	9

The answer is shown in below figures: 1% for 29, 58, 49, 69, 18, 75

49		58	69		75		18	38	29
0	1	2	3	4	5	6	7	8	9

13. Select **all correct** statements about **sorting** a list of numbers with an **increasing order** (10%)

(a) **Insertion sort** encounters its worst case if the input list is in **decreasing order**

(b) If average-case time complexity is the main consideration, **quick sort** is a good choice

(c) If worst-case time complexity is important, **merge sort** is more suitable than **quick sort**

(d) When using **heap sort**, it is better to use a **max-heap** for sorting numbers in an **increasing order**

(e) All comparison-based sorting algorithms have a time complexity $\Omega(n \log n)$
abcde