CUHK(SZ)-CSC3100 Final Exam

1st Semester, 2021-2022

Note:
a. No notes or calculators are allowed in the exam.
b. This exam paper has four pages, in double-sided printing.
c. Answer all questions within 150 minutes in an **answer book**.

B

1. (10 points) Choose **ONE** solution that best suits each question. (2 points each)

(1) The typical complexity of searching an element in a balanced binary search tree is:

A. $O(1)$

B. $O(\log n)$

C. $O(n)$

D. $O(n \log n)$

C/D

(2) For input sequence: $\{2, 3, 5, 6, 9, 11, 15\}$, which algorithm needs the largest number of comparisons?

A. Bubble Sort

B. Insertion Sort

C. Merge Sort

D. Quick Sort

?

A/C

(3) For input sequence: $\{2, 3, 5, 6, 9, 11, 15\}$, which algorithm runs in $O(N)$ time? Note: throughout the paper, N (or n) denotes the number of input elements.

A. Bubble Sort

B. Selection Sort

C. Insertion Sort

D. ~~Quick Sort~~

C

(4) Which algorithm typically has the most space requirement on the same input?

A. Selection Sort

B. Insertion Sort

C. Merge Sort

D. Quick Sort

B

(5) Which of the following typically runs the fastest?

A. Searching for an element in a singly linked list $O(N)$

B. Computing the hash value for a given input $O(1)$

C. Searching for a value in a binary search tree

D. Emptying all elements from a stack $O(N)$

2. (10 points) Answer the following questions with either **true** or **false** (2 points each).

   (1) One can implement a stack based on a linked list so that each individual push/pop operation is in time $O(1)$. √ 有pointer.

   (2) One can implement a stack (of unlimited size) based on a linear array so that each individual push/pop operation is in time $O(1)$. √

   (3) The core data structure used in Depth First Search is a ~~queue~~. × stack

   (4) One can reverse the order of the elements in a linked list in time $O(n)$. √

   (5) It is possible to append two linked lists in time $O(1)$. √ (pointer)

3. (15 points) Consider the graph in Figure 1 with nodes **A** to **K** and weights shown on edges. Answer the following questions (5 points each).

   (1) In what order are vertices visited for Depth First Search starting at node A.

   (2) In what order are vertices visited for Breadth First Search starting at node A.

   (3) Show the minimum spanning tree (MST) derived by Kruskal's or Prim's algorithm.

   Note: If a node has multiple adjacent nodes to visit, always visit in **alphabetic** order!

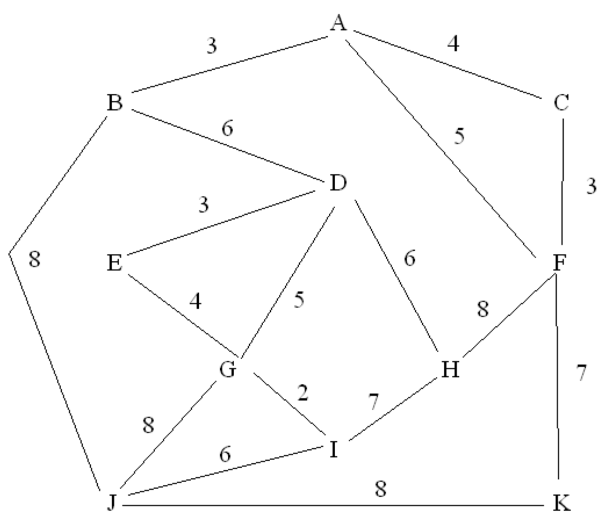


Figure 1: A Weighted Graph.

4. (10 points) For the directed graph shown in Figure 2, answer the following questions.

   (1) Draw both the adjacency matrix and adjacency list representations of this graph.

   (2) Give two valid topological orderings of the nodes in the graph.

3. (15 points) Consider the graph in Figure 1 with nodes **A** to **K** and weights shown on edges. Answer the following questions (5 points each).

(1) In what order are vertices visited for Depth First Search starting at node A.

(2) In what order are vertices visited for Breadth First Search starting at node A.

(3) Show the minimum spanning tree (MST) derived by Kruskal's or Prim's algorithm.

Note: If a node has multiple adjacent nodes to visit, always visit in **alphabetic** order!

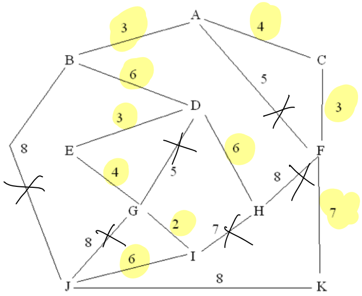


Figure 1: A Weighted Graph.

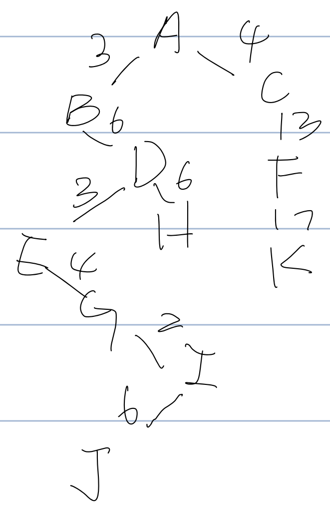
(1) DFS: ABDEGIHFCKJ

(2) BFS: ABCFDJHKEGI

Prim's:

(3) Q:

A	B	C	D	E	F	G	H	I	J	K
0	3	4	6	3	3	4	6	2	6	7

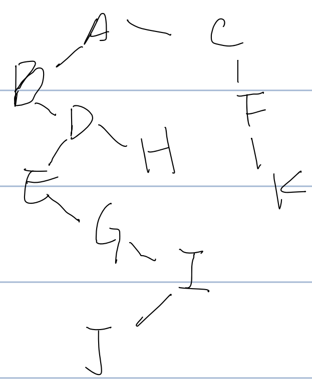


$3+4+6+3+7+6+3$
$+4+2+6$
$=7+9+10+10+8$
$=16+28$
$=44$

Prim (s, v, E, w):

```
Q ← ∅
for all v∈V:
    do key[v] ← ∞
       π[v] ← 0 NIL
       insert (Q, v, key[v])
Decrease-key (Q, S, 0)
while Q is not empty:
    u = extract_min(Q)
    for v∈neighbor[u]:
        if v is in the Q and d[v]>w[u,v]:
            ~~d[v] ← w[u,v]~~ & π[v] ← u
            decrease_key (Q, v, w[u,v])
```

Kruskal: {A, B, C, F, D, G, I, H, E, D, J, K}



Kruskal (V, w, s, E):

```
    R = { }
    sort(E) edge in Graph
    for v∈V:
        label[v] = v
        set-array[v] = {v}
    for (u,v) in E:
        if label[u] = label[v]:
              continue
        R.add((u,v))
        if set_array[u].size ≥ set_array[v].size:
              for w in set-array[v]:
                  label[w] = label[u]
                  see-array[u].append(w)

    else:
        for w in set-array[u]:
            label[w] = label[v]
            set-array[v] = append(w)
```

4. (10 points) For the directed graph shown in Figure 2, answer the following questions.

(1) Draw both the adjacency matrix and adjacency list representations of this graph.

(2) Give two valid topological orderings of the nodes in the graph.

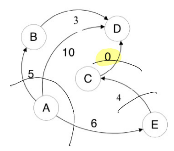
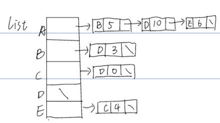


Figure 2: A Directed Graph.

(1) matrix

	A	B	C	D	E
A	0	5	0	10	6
B	0	0	0	3	0
C	0	0	0	0	0
D	0	0	0	0	0
E	0	0	4	0	0

list



(2) ABECD; AECBD

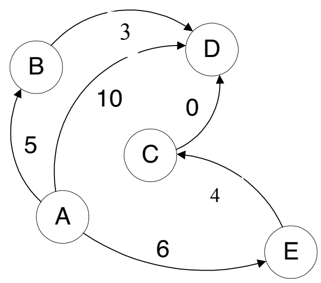


Figure 2: A Directed Graph.

5. (15 points) Suppose the class **java.util.LinkedList** is implemented by a doubly linked list, maintaining a reference to the first and last node in the list, along with its size.

```
public class LinkedList<Item> {
   private Node first;   // the first node in the linked list
   private Node last;    // the last node in the linked list
   private int N;        // number of items in the linked list
   private class Node {
      private Item item;         // the item
      private Node next, prev;   // next and previous nodes
   }
   ...
}
```

What are the best estimates of the worst-case running time of the following operations in big-O notation? (3 points each, choose among $O(1), O(\log N), O(N), O(N \log N), O(N^2)$)

(1) **addFirst(item)**: Add the item to the beginning of the list.

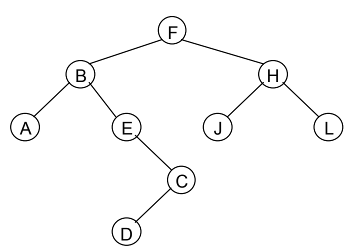
(2) **get(i)**: Return the item a position i of the list.

(3) **set (i, item)**: Replace position i of the list with item.

(4) **removeLast()**: Delete and return the item at the end of the list.

(5) **contains(item)**: Is the item in the list?

6. (10 points) Consider a binary tree shown in Figure 3. For each of the **preorder**, **inorder** and **postorder** traversals, give the order in which the nodes are visited.



preorder: FBAECDHJL
inorder: ABEDCFJHL
postorder: ADCEBJLHF

Figure 3: A Binary Tree.

5\.

5. (15 points) Suppose the class **java.util.LinkedList** is implemented by a doubly linked list, maintaining a reference to the first and last node in the list, along with its size.

```
public class LinkedList<Item> {
   private Node first;   // the first node in the linked list
   private Node last;    // the last node in the linked list
   private int N;        // number of items in the linked list
   private class Node {
      private Item item;         // the item
      private Node next, prev;   // next and previous nodes
   }
   ...
}
```

What are the best estimates of the worst-case running time of the following operations in big-O notation? (3 points each, choose among $O(1), O(\log N), O(N), O(N \log N), O(N^2)$)

(1) **addFirst(item)**: Add the item to the beginning of the list.

(2) **get(i)**: Return the item a position i of the list.

(3) **set (i, item)**: Replace position i of the list with item.

(4) **removeLast()**: Delete and return the item at the end of the list.

(5) **contains(item)**: Is the item in the list?

(1) O(1)

(2) O(N)

(3) O(N)

(4) O(1)

(5) O(N)

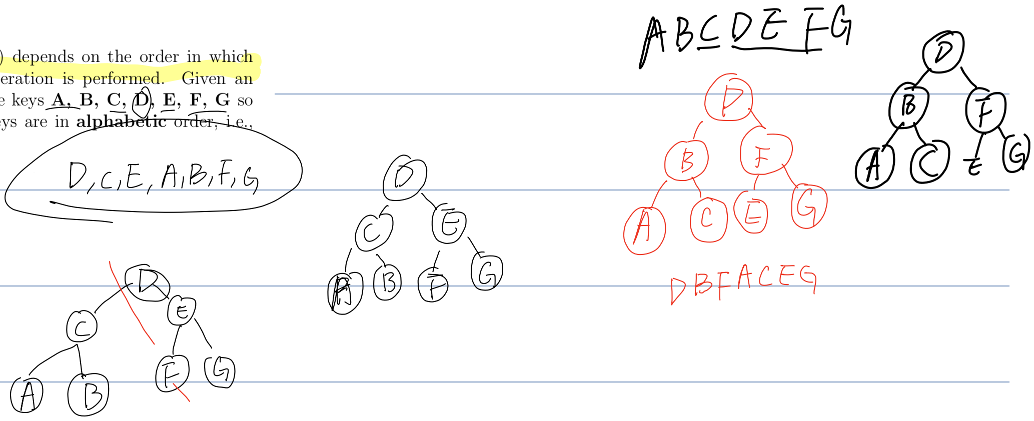
7\.

7. (10 points) The height of a binary search tree (BST) depends on the order in which the keys are inserted into a tree if no balancing operation is performed. Given an initially empty BST, in what order will you insert the keys **A, B, C, D, E, F, G** so that the height of the BST is minimal. Note: the keys are in **alphabetic** order, i.e., $A < B < C < D < E < F < G$.

写算法

ABCDEFG

D,C,E,A,B,F,G



DBFACEG

8\.

8. (10 points) A connected component of a graph is a set of nodes where each node can reach every other node in the component along the given edges, and which is connected to no additional nodes. For example, the graph in Figure 4 has three connected components.

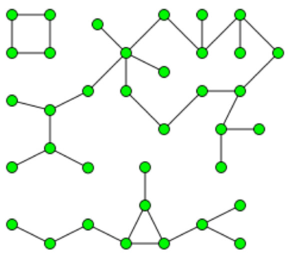


Figure 4: An Undirected Graph with Three Connected Components.

Explain, in words, how to use Kruskal's algorithm to compute the number of connected components in an undirected graph.

Kruskal

+ 遍利不同 label 数.

Master theorem: examples

$T(n) \le a \cdot T(n/b) + O(n^d)$

$$T(n) = \begin{cases} O(n^d \log n) & \text{if } a = b^d \\ O(n^d) & \text{if } a < b^d \\ O(n^{\log_b(a)}) & \text{if } a > b^d \end{cases}$$

- $g(1) = c_0, g(n) \le 2 \cdot g(n/2) + c_1 \cdot n^{0.5}$
  - We have that $a = 2, b = 2, d = 0.5$
  - Since $\log_b a > d$, we have that: $g(n) = O(n^{\log_b a}) = O(n)$
- $g(1) = c_0, g(n) \le 2 \cdot g(n/4) + c_1 \cdot \sqrt{n}$
  - We have $a = 2, b = 4, d = 0.5$
  - Since $\log_b a = d$, we have that: $g(n) = O(n^d \cdot \log n) = O(\sqrt{n} \cdot \log n)$

9\.

9. (10 points) The algorithm (pseudo code) in Figure 5 sorts an array (given as parameter *seq*) of n numbers. Estimate the time complexity of the algorithm as a function of input size n. Briefly show your calculation. Note: The function parameter *seq* is passed by reference, and $2n/3$ will be rounded up to the nearest integer in execution.

```
def triplesort(seq):
        if n <= 1: return
        if n == 2:
                replace seq by [min(seq),max(seq)]
                return
        triplesort(first 2n/3 positions in seq)
        triplesort(last 2n/3 positions in seq)
        triplesort(first 2n/3 positions in seq)
```

Figure 5: A Triple Sort Algorithm.

$T(n) = 3T(\frac{2}{3}n) + f(n)$, $f(n) = O(1)$

$O(n^{\log_{\frac{3}{2}} 3})$

$a = 3, b = \frac{3}{2}$ $d = 0$

7. (10 points) The height of a binary search tree (BST) depends on the order in which the keys are inserted into a tree if no balancing operation is performed. Given an initially empty BST, in what order will you insert the keys **A, B, C, D, E, F, G** so that the height of the BST is minimal. Note: the keys are in **alphabetic** order, i.e., $A < B < C < D < E < F < G$.

8. (10 points) A connected component of a graph is a set of nodes where each node can reach every other node in the component along the given edges, and which is connected to no additional nodes. For example, the graph in Figure 4 has three connected components.

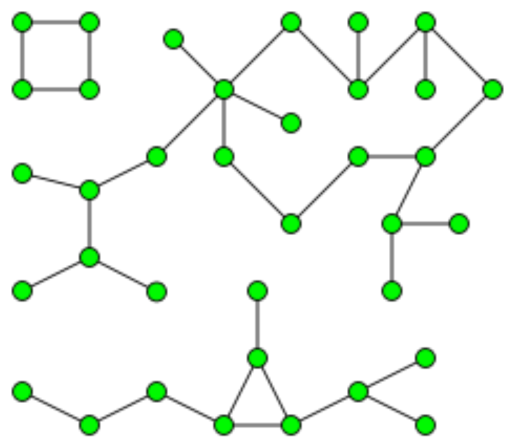


Figure 4: An Undirected Graph with Three Connected Components.

Explain, in words, how to use Kruskal's algorithm to compute the number of connected components in an undirected graph.

9. (10 points) The algorithm (pseudo code) in Figure 5 sorts an array (given as parameter *seq*) of n numbers. Estimate the time complexity of the algorithm as a function of input size n. Briefly show your calculation. Note: The function parameter *seq* is passed by reference, and $2n/3$ will be rounded up to the nearest integer in execution.

```
def triplesort(seq):
        if n <= 1: return
        if n == 2:
                replace seq by [min(seq),max(seq)]
                return
        triplesort(first 2n/3 positions in seq)
        triplesort(last 2n/3 positions in seq)
        triplesort(first 2n/3 positions in seq)
```

Figure 5: A Triple Sort Algorithm.