The Chinese University of Hong Kong, Shenzhen

Semester 1: AY2022/2023

CSC 3100: Data Structures

Final Exam (close book)

Jan 2023 Total time allowed: 120 mins

Good luck!

4. [**27 marks**] Multiple choices questions.

Each correct question is worth 3 points. Choose the one BEST answer for each question. Remember not to devote too much time to any single question! GOOD LUCK!

B

1) What does the following function do for a given Linked List with first node as head?

```
voice func(Node head) {
    if(head == null) {
       return;
    }

    func(head.next);

    System.out.print(head.data);
}
```

$\boxed{1} \to \boxed{2} \to \boxed{3}$

func(1) → func(2) → func(3) → func(null)

↳ print(3)   ↳ return.

A. Prints all nodes of linked lists

B. Prints all nodes of linked list in reverse order

C. Prints alternate nodes of Linked List

D. Prints alternate nodes in reverse order

A

2) Suppose we need to sort a list of student records in ascending order, using the student ID number (a 9-digit number) as the key (i.e., sort the records by student ID number). If we need to guarantee that the running time will be no worse than n log n, which sorting methods could we use?

A. mergesort

B. quicksort

C. insertion sort

D. Either mergesort or quicksort

E. None of these sorting algorithms guarantee a worst-case performance of n log n or better

B

3) Which of the following patterns in a computer program suggests that a hash data structure could provide a significant speed-up (check all that apply)?

A. Repeated maximum computations

B. Repeated lookups

C. Repeated minimum computations

D. None of the other options

A

4) Which of the following is not a property that you expect a well-designed hash function to have?

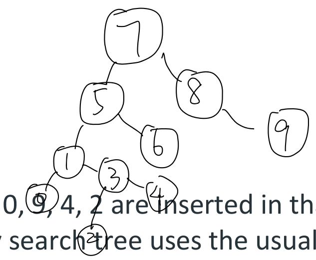
A. The hash function should “spread out” every data set (across the buckets/slots of the hash table).

B. The hash function should “spread out” most (i.e., “non-pathological”) data sets (across the buckets/slots of the hash table).

C. The hash function should be easy to store (constant space or close to it).

D. The hash function should be easy to compute (constant time or close to it).

5) Suppose you want to alphabetize the following list of three letters words in ascending a-z alphabetical order with radix sort: [dog, lot, caw, log, dot]. Recall that radix sort calls bucket sort as a subroutine. After the first call to bucket sort, what is the order of the list?
A. [dog, log, dot, lot, caw]
B. [caw, dog, dot, log, lot]
C. [caw, dog, dot, lot, log]
D. [dog, log, lot, dot, caw]

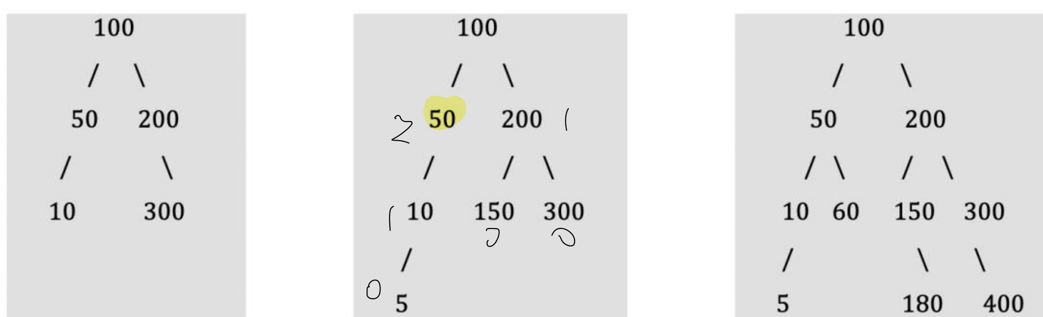


C

6) Suppose the numbers 7, 5, 1, 8, 3, 6, 0, 9, 4, 2 are inserted in that order into an initially empty binary search tree. The binary search tree uses the usual ordering on natural numbers. What is the in-order traversal sequence of the resultant tree?
A. 7 5 1 0 3 2 4 6 8 9
B. 0 2 4 3 1 6 5 9 8 7
C. 0 1 2 3 4 5 6 7 8 9
D. 9 8 6 4 2 3 0 1 5 7

0 1 2 3 4 5 6 7 8 9

7) What is the worst-case possible height of AVL tree? Assume base of log is 2 and N could be an arbitrary integer.
A. $2logN$
B. $1.44logN$
C. N/2
D. N



a b c

8) Which of the following is an AVL Tree?
A. only a
B. only c
C. a and c
D. a,b, and c

9) An array consists of n elements. We want to create a heap using the elements. The best time complexity of building a heap will be in order of

A. O(n*n*logn)

B. O(n*logn)

C. O(n*n)

D. O(n)

2.

2. **[15 marks]** Assume that we have a sequence of 7 keys: 6, 12, 34, 29, 18, 17, 19 and the size of the hash table is m = 11.
(6, 1, 1, 7, 7, 6, 8)
   (1) **[3 marks]** Use chaining with $h(k) = k\%11$ and show the table after inserting the above keys.
   (2) **[3 marks]** Use linear probing with $h(k) = k\%11$ and show the table after inserting the above keys.
   (3) [3 marks] Use double hashing with $h(k, i) = (h(k) + i \cdot h'(k))\%11$, where $h(k) = k\%11$ and $h'(k) = k\%7$ with i ranging from 0 to 10, and show the table after inserting the above keys.
   (4) [3 marks] A hash family H (mapping U into n buckets) is called 2-universal, if for all x ≠ y ∈ U and for all a, b ∈ {1, ..., n}, they satisfy $P[(h(x), h(y)) = (a, b)] = \frac{1}{n^2}$. Prove that if H is 2-universal, then it is universal as well.
   (5) **[3 marks]** Show that the converse statement of (4) is not true by a counter example. That is, show that there is a universal family that is not 2-universal.

(4) $P(h(x), h(y)) = (a, b) = \frac{1}{n^2}$

$P(h(x) = h(y)) = \frac{1}{n^2} < \frac{1}{n}$ ✓

(5)

T

(1)

T[0]
T[1] → head ⇔ 12 ⇔ 34 ⇔ tail
T[2]
T[3]
T[4]
T[5]
T[6] → head ⇔ 6 ⇔ 17 ⇔ tail
T[7] → head ⇔ 29 ⇔ 18 ⇔ tail
T[8] → head ⇔ 19 ⇔ tail
T[9]
T[10]

2.

	0	1	2	3	4	5	6	7	8	9	10	11
(1)		12	34				6	29	18	17	19	

$6\%11 = 6$
$12\%11 = 1$
$34\%11 = 1$
$29\%11 = 7$

$18\%11 = 7$
$17\%11 = 6$
$19\%11 = 8$

(2) index
0
1 = 12, 34
2
3
4
5
6 = 6, 17
7 = 29, 18
8 = 19
9
10
11

(3) $h(6,0) = 6\%11 = 6$
$h(12,0) = 12\%11 = 1$
$h(34,0) = 34\%11 = 1$ $\quad h(34,1) = (1 + 34\%7)\%11 = 7$
$h(29,0) = 29\%11 = 7$ $\quad h(29,1) = (7 + 29\%7)\%11 = 8$
$h(18,0) = 18\%11 = 7$ $\quad h(18,1) = (7 + 18\%7)\%11 = 0$
$h(17,0) = 17\%11 = 6$ $\quad h(17,1) = (6 + 17\%7)\%11 = 9$
$h(19,0) = 19\%11 = 8$ $\quad h(19,1) = (8 + 19\%7)\%11 = 2$

(2)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	12	34				6	29	18	17	19						

(3)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
18	12	19				6	34	29	17							

$h(34,1) = (h(34) + 1 \times h'(34))\%11 = (1+6)\%11 = 7$

$h(34) = 1 \qquad h'(34) = 6$

$h(29,1) = (7 + 1 \times 1)\%11 = 8$

$h'(29) = 29\%7 = 1 \qquad h'(18) = 18\%7 = 4$

$h(18,1) = (7 + 1 \times 4)\%11 = 0$

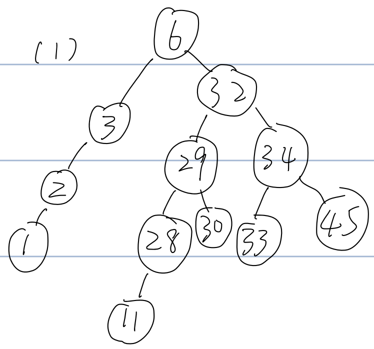
$h(17,1) = (6 + 1 \times 3)\%11 = 9$

$h(19,1) = (8 + 1 \times 5)\%11 = 2$

3. **[20 marks]** Binary Search Tree
   (1) **[4 marks]** For a given array A = [6, 32, 29, 28, 34, 11, 3, 2, 1, 33, 30, 45], Please draw the final binary search tree based on the given order of A.
   (2) **[1 mark]** What is the time complexity (on average) to build a binary search tree for an array?
   (3) **[4 marks]** Please check whether the resulting binary search tree of (1) is balanced or not, and also provide the reason.
   (4) **[1 mark]** One wants to sort an array by first building a binary search tree (a.k.a., tree sort). Which traversal order of a binary tree can give a ~~sorted~~ sequence? preorder, inorder, or postorder?
   (5) **[4 marks]** What is the worst time complexity of tree sort? In which scenarios the time complexity of tree sort will degrade to be the worst case? Hint: the final structure of a binary search tree is highly dependent on the order of such an array. When the resulting binary search tree is unbalanced, the operations over the tree might be inefficient.
   (6) **[6 marks]** Find the k'th smallest element in a binary search tree. Write code or pseudocode.

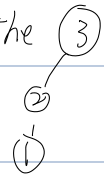
```
//Please assume the binary search tree was already built.
void find_K_smallest(BSTNode root) {

}
```

(1)



(2) $O(n \log n)$ n is the number of array item

(3) no since the



|left height − right height| = 2 ✓

balance factor.

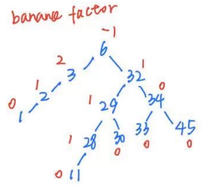
(4) inorder

(6)

(5) $O(n^2)$ linear tree highly imbalanced.

(6)
```
def find k_smallest(BSTnode, root):
    Inorder_list = [ ]
    def inorder(node, k)
        if node not None:
            Inorder(node.left)
            Inorder_list.append[node.key]
            Inorder(node.right)

    Inorder(node, root)
    return Inorder_list[k-1]
```

3.(1)



(2) O(nlogn)

(3) Unbananced, one node's banana factor is 2

(4) inorder

(5) O(n²), in a line

(6) inOrder(root)
put elements into array A
print A[k-1]

(6)

findAndDele[redacted]

2. [**15 marks**] Assume that we have a sequence of 7 keys: 6, 12, 34, 29, 18, 17, 19 and the size of the hash table is m = 11.
   (1) [**3 marks**] Use chaining with $h(k) = k\%11$ and show the table after inserting the above keys.
   (2) [**3 marks**] Use linear probing with $h(k) = k\%11$ and show the table after inserting the above keys.
   (3) [3 marks] Use double hashing with $h(k,i) = \left(h(k) + i \cdot h'(k)\right)\%11$, where $h(k) = k\%11$ and $h'(k) = k\%7$ with i ranging from 0 to 10, and show the table after inserting the above keys.
   (4) [3 marks] A hash family H (mapping U into n buckets) is called 2-universal, if for all x ≠ y ∈ U and for all a, b ∈ {1, ..., n}, they satisfy P[(h(x), h(y)) = (a, b)] = $\frac{1}{n^2}$. Prove that if H is 2-universal, then it is universal as well.
   (5) [**3 marks**] Show that the converse statement of (4) is not true by a counter example. That is, show that there is a universal family that is not 2-universal.

3. [**20 marks**] Binary Search Tree
   (1) [**4 marks**] For a given array A = [6, 32, 29, 28, 34, 11, 3, 2, 1, 33, 30, 45], Please draw the final binary search tree based on the given order of A.
   (2) [**1 mark**] What is the time complexity (on average) to build a binary search tree for an array?
   (3) [**4 marks**] Please check whether the resulting binary search tree of (1) is balanced or not, and also provide the reason.
   (4) [**1 mark**] One wants to sort an array by first building a binary search tree (a.k.a., tree sort). Which traversal order of a binary tree can give a sorted sequence? preorder, inorder, or postorder?
   (5) [**4 marks**] What is the worst time complexity of tree sort? In which scenarios the time complexity of tree sort will degrade to be the worst case? Hint: the final structure of a binary search tree is highly dependent on the order of such an array. When the resulting binary search tree is unbalanced, the operations over the tree might be inefficient.
   (6) [**6 marks**] Find the k'th smallest element in a binary search tree. Write code or pseudocode.

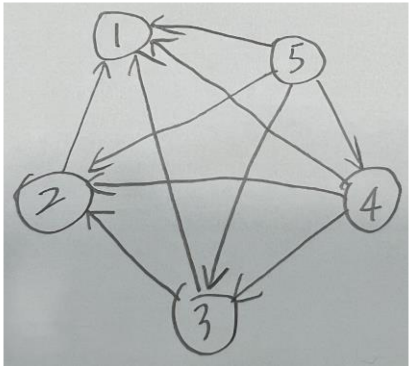
```
//Please assume the binary search tree was already built.
void find_K_smallest(BSTNode root) {

}
```

4. [**18 marks**] Graph analysis

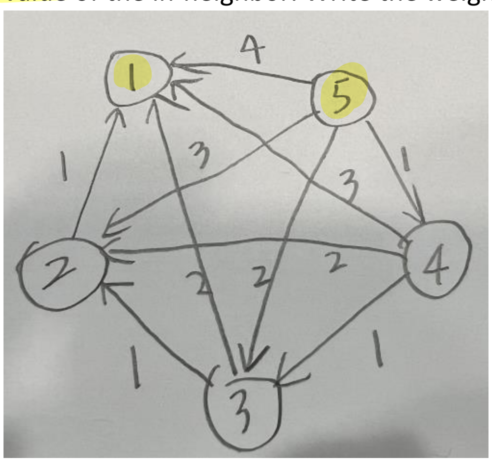
   a) [**2 marks**] What is the minimum number of edges possible in a connected undirected graph that does NOT have self-connected edges? Give an exact answer in terms of the number of vertices |V|, not in big-O notion. You need to explain the number.
   |V| - 1

   b) [**7 marks**] Suppose a directed graph has ***k*** nodes, where each node corresponds to a number (1, 2, ..., k) and there is an edge from node ***i*** to node ***j*** if and only if ***i>j***.

I. [**1 mark**] Draw the graph using circles and arrows, assuming *k = 5*.



II. [**1 mark**] An edge connects two neighbors, out-neighbor and in-neighbor. Let's assume the weight of an edge equals to the value of the out-neighbor minus the value of the in-neighbor. Write the weight for each edge that you draw above.



III. [**3 marks**] Draw both an adjacency list and an adjacency matrix presentation of the graph assuming k=5. You don't need to care about the edge weight. The matrix and list need to be organized by following the number order.

	1	2	3	4	5
1	F	F	F	F	F
2	T	F	F	F	F
3	T	T	F	F	F
4	T	T	T	F	F
5	T	T	T	T	F

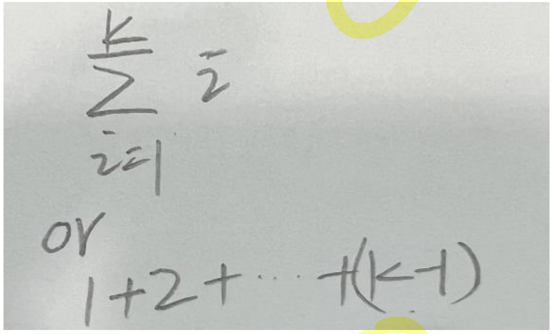
1	→				
2	→	1			
3	→	1	2		
4	→	1	2	3	
5	→	1	2	3	4

IV. [**1 mark**] What is the space consumption of the two representations (i.e. adjacency list and adjacency matrix). Give an exact answer in term of k, not in big-O notion.

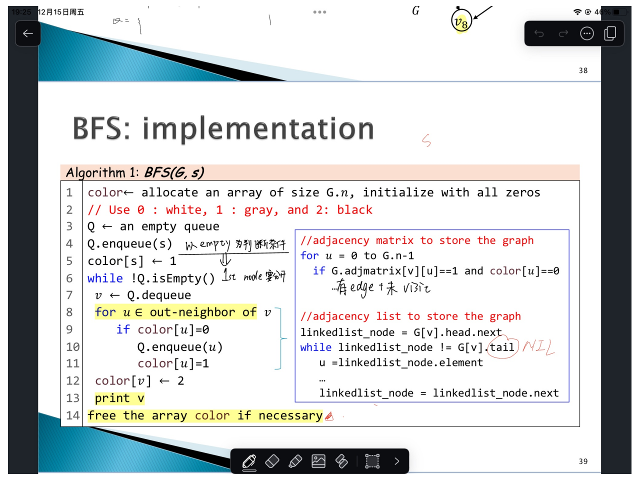
Adjacency matrix: k * k

Adjacency list: k * k / 2

V. [**1 mark**] **In terms of k** (if k is relevant), exactly how many edges are in the graph assuming an **arbitrary** value of k?



c) [**5 marks**] Write the pseudocode for a BFS method to print out an ordering of the graph vertices that can be reached in a breadth-first search given some a starting vertex v.



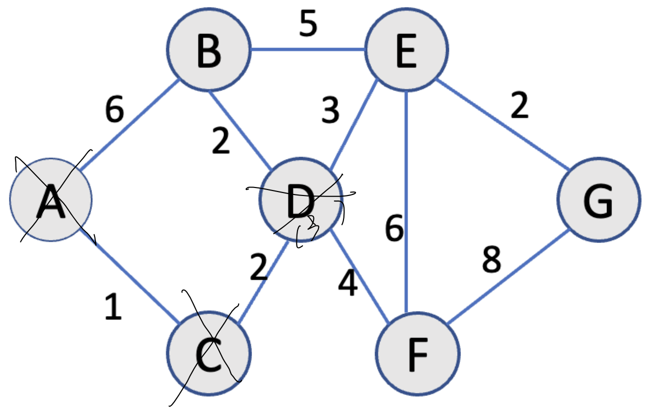

```
//assuming Vertex v is valid
void bfs(Vertex v) {

}
void bfs(Vertex v) {
    Queue pending = new Queue();
    v.visited = true;
    pending.add(v);
    while (!pending.isEmpty()) {
        Vertex next = pending.remove();
        print(next);
        if (!v.visited) {
            next.visited = true;
            pending.add(next);
        }
    }
}
```

d) [**2 marks**] what is the runtime of your pseudocode algorithm in terms of |E| and |V|? You need to provide the tightest bound in big-O notion and discuss the reason.
O(|E|) O(node+edge) ↳ BFS

e) [**2 marks**] What is the visiting order if DFS is used? Assuming the traversal is from a smaller id to a larger id. The traversal is starting from vertex k (i.e., 5).

5, 1, 2, 3, 4

5. **[20 marks]** Consider a weighted undirected graph, which is shown as below, where each edge carries a weight denoting the distance between two nodes.



(a) [**6 marks**] We want to use Dijkstra's algorithm to determine the shortest path from vertex A to each of the other vertices. Update the entries in the following table to indicate the current shortest known distance and predecessor vertex on the path from vertex A to each vertex as the algorithm progresses. (~~Cross out the old entries when you~~ add new ones.). The initial values for the distances are given for you.

Vertex	D (distance from vertex A)	Predecessor vertex
A	0	0
B	6	A
C	1	A
D	3	C
E	~~11~~ 6	~~B~~ D
F	7	D
G	8	E

(b) [**6 marks**] Implement the Floyd's algorithm and analyze its time complexity in terms of |E| and |V| in big-O notion.

```
//Assuming the graph G is using the adjacency matrix representation
//You implementation should be less than 15 lines of code.
void Floyd(int [][] G, int N) {
```

j

j

k

w

```
1. D ← W    // initialize D array to W [ ]
2. P ← 0    // initialize P array to [0]
3. for k ← 1 to n
      // Computing D' from D
4.    do for i ← 1 to n
5.       do for j ← 1 to n
6.          if (D[ i, j ] > D[ i, k ] + D[ k, j ] )
7.             then  D'[ i, j ] ← D[ i, k ] + D[ k, j ]
8.                   P[ i, j ] ← k;
9.             else D'[ i, j ] ← D[ i, j ]
10. Move D' to D
}
```

(c) [**6 marks**] Construct an MST using Kruskal's algorithm. You need to implement the core algorithm of the Kruskal's algorithm.

```
// construct an MST using Kruskal's algorithm
// You only need to implement the core algorithm
void KruskalMST(int [][] G, int N) {
      // G is an adjacency matrix
```

```
Assume vertices are 1, 2, …, n, and E >= V
    Sort all the edges                          ← O(ElogE)
    for each v ∈ V                               } O(V)
        map.add(v, {v})
    for each edge (u, v) ∈ E                                          } O(VE)
        setU = map.get(u), setV = map.get(v)    } O(1)
        isConnected = false
        for each vertex w ∈ setU                } O(setU.size)
            if w == v
                isConnected = true
                break
        if isConnected == false                 } O(setV.size)
            setU = setU  setV
            map.add(u, setU), map.add(v, setU)
            R = R  {(u, v)}
    Output R
```

Can we do better?

```
Assume vertices are 1, 2, …, n, and E >= V
    Sort all the edges                     ← O(ElogE)
    for each v ∈ V                          } O(V)
        label[v] = v
        setArray[v] = {v}
    for each edge (u, v) ∈ E                } O(E)                    } O(ElogE)
        uL = label[u], vL = label[v]
        if uL == vL  continue
        R.add((u, v))
       if setArray[uL].size >= setArray[vL].size      } O(ElogV)
          for each vertex w ∈ setArray[vL]
                label[w] = uL
                setArray[uL].add(w)
        else
            for each vertex w ∈ setArray[uL]
                label[w] = vL
                setArray[vL].add(w)
    Output R
```

}

(d) [**2 marks**] In your implementation in (c), what's the time complexity in terms of |E| and |V| in big-O notion (the tightest bound)? You need to provide the reason.

O(N^3)

My CSC3100 students: It has been a fantastic journey with you all. Wish you all the best!