
Name

PRACTICE

CSE 373 FINAL EXAM: 150 POINTS
Two Double-Sided 8.5 x 11 Crib Sheets Allowed

1. For each of the following applications, choose the best structure from the following list. If some other structure may also be a good choice, list it after the best one.

- linked list
- stack
- queue
- AVL tree
- B^+-tree
- hash table
- binary min-heap
- adjacency matrix
- adjacency list
- union-find (up-tree) structure

(a) A structure, which can be easily updated, for keeping track of which students went to the same high school.

up-tree

(b) A structure for storing the nodes with no more incoming edges in a topological sort algorithm.

queue

(c) A structure that allows rapid access to all nodes adjacent to a given one in a graph without going through every node.

adjacency list

(d) A structure that has multiple keys per node and log N levels.

B+-tree

2. Give the time complexity of each of the following in Big-Oh notation, assuming **worst-case** unless otherwise specified:

(a) Producing an ordering of nodes in a directed acyclic graph such that each node comes after all its "prerequisite" nodes.

$O(|E| + |V|)$ fn $|E|$ edges and $|V|$ vertices

(b) Finding the correct leaf node to search in a B+-tree.

$O(\log N)$ fn N items

(c) Finding a key in a hash table that uses chaining.

$O(N)$ in worst case if all N hash to one bucket

(d) Finding the minimal cost paths from a start node to all other nodes in a digraph using the most efficient algorithm known.

$O(|E| \log |V| + |V| \log |V|)$

or $O(|E + |V| \log |V|)$

(e) Performing the union operation on two sets after performing the two finds.

$O(1)$

(f) Deleting a key from a min-heap.

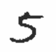
$O(\log N)$ fn N elements

(g) Removing an item from a queue implemented with a circular array.

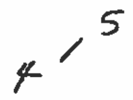
$O(1)$

3. Show how to insert the following sequence of keys into an initially empty AVL tree. Show each step and show all separate rebalancing operations. The sequence is: 5, 4, 3, 2, 1

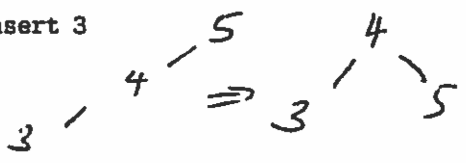
Insert 5



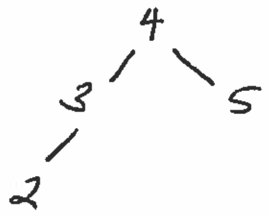
Insert 4



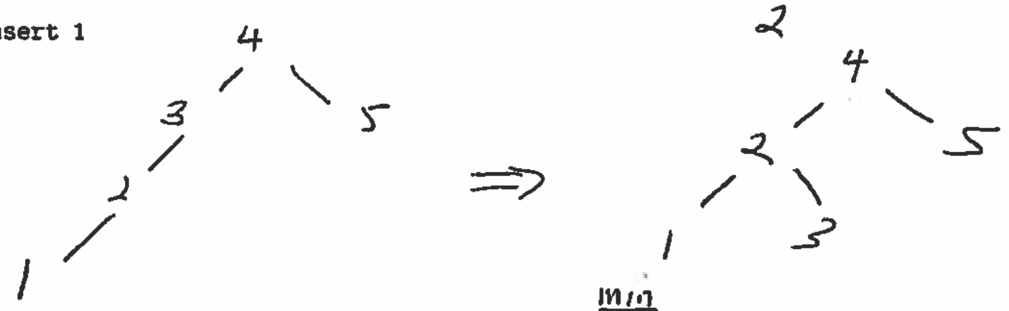
Insert 3



Insert 2

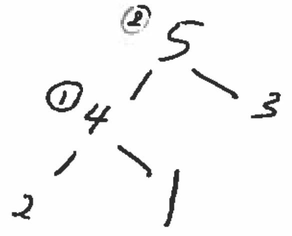


Insert 1

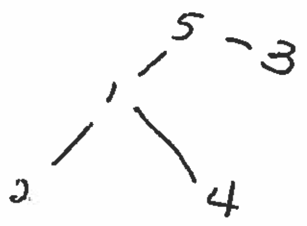


4. Show how to insert the same 5 elements into an empty $\overset{\text{min}}{\text{heap}}$, starting with an array, using the efficient BuildHeap. First convert the initial array 5 4 3 2 1 to its tree form. Then show each step in the BuildHeap procedure with trees. Finally, convert back to an array.

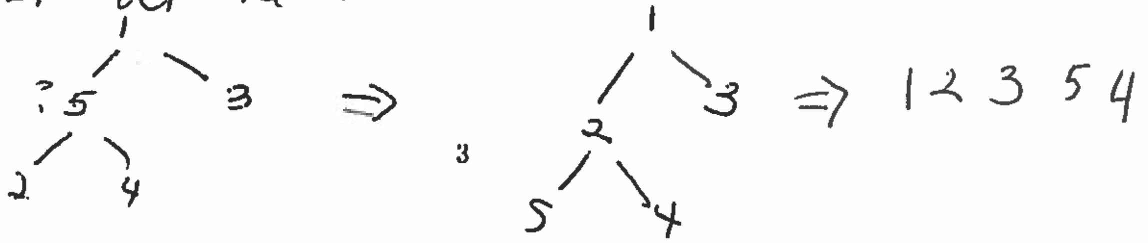
5 4 3 2 1 ⟹



Step 1. at the 4

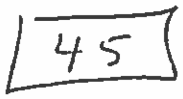


step 2. at the 5

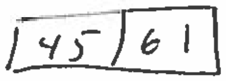


5. Show how to insert the following sequence into an initially empty B+-tree that has a max of 2 keys (and 3 pointers) per interior node and 2 keys per leaf. Show the tree at each step. 45, 61, 32, 89, 100, 2, 6

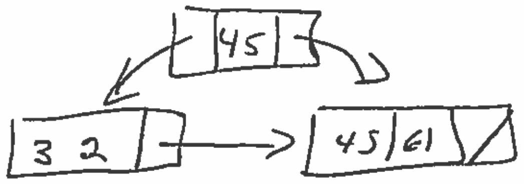
1



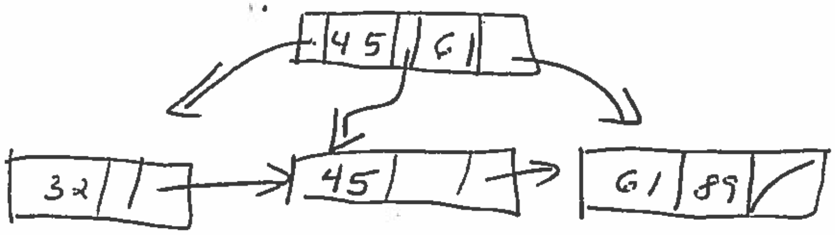
2



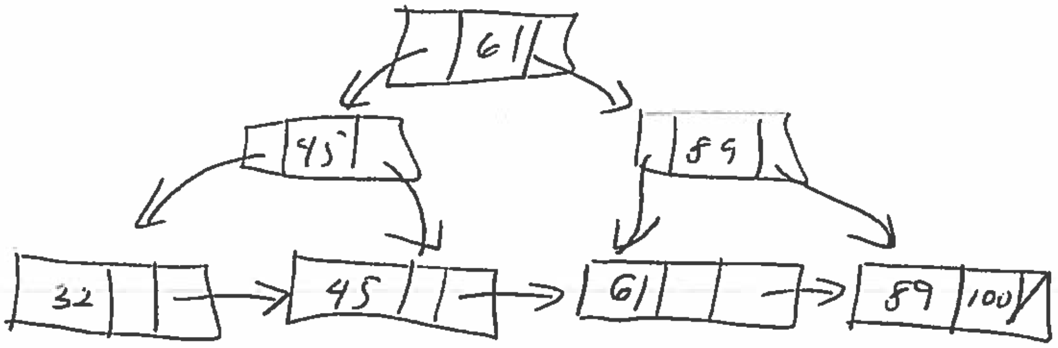
3



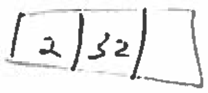
4



5

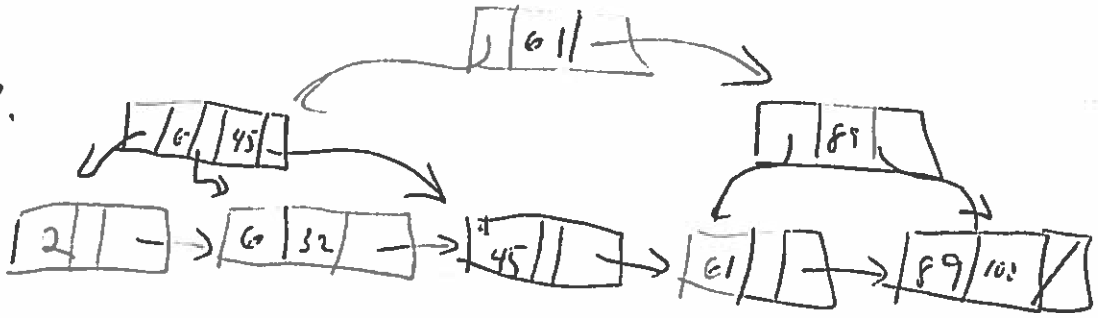


6



No other change

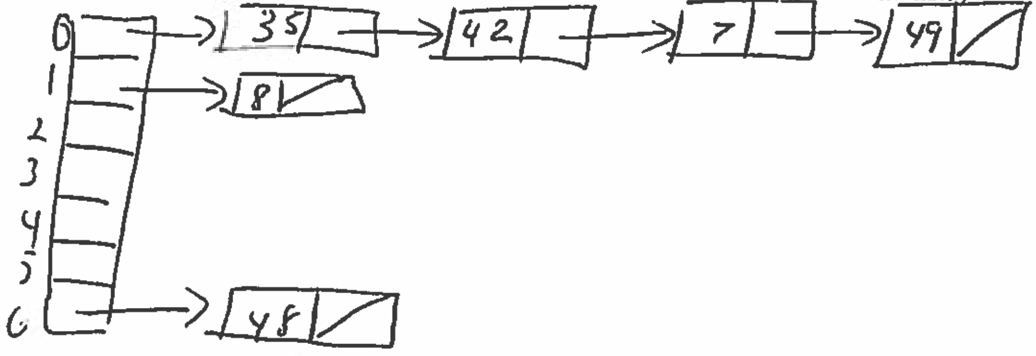
7.



4%7=0 42 % 7 = 0
8 % 7 = 1 48 % 7 = 6
7 % 7 = 0 35 % 7 = 0

6. Show how to insert the following keys into an initially empty hash table of size 7 with indexes 0-6 using the hash function h(i) = i % 7: 49, 8, 7, 42, 48, 35

(a) with chaining (just show the final result and insert at the front of the lists)



(b) with linear probing (show each of the 6 steps horizontally).

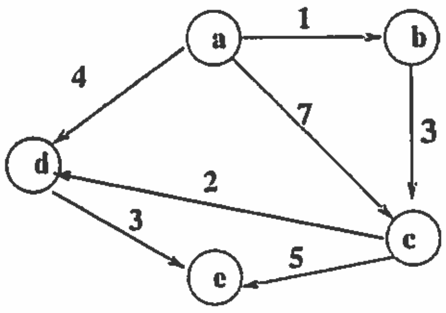
	1)	2)	3)	4)	5)	6)
0	49	49	49	49	49	49
1		8	8	8	8	8
2			7	7	7	7
3				42	42	42
4						35
5						
6					48	48

(c) with quadratic probing (show each of the 6 steps horizontally) and indicate if it cannot insert.

	1)	2)	3)	4)	5)	6) 35
0	49	49	49	49	49	49
1		8	8	8	8	8
2				42	42	42
3						
4			7	7	7	7
5						
6					48	48

(1 4 2 2 4 1 0)* repeats

7. Given the following weighted digraph:



(a) Starting at node a, list the nodes in the order of a depth-first traversal (when there is a choice, choose the smaller-lettered node first).

a b c d e

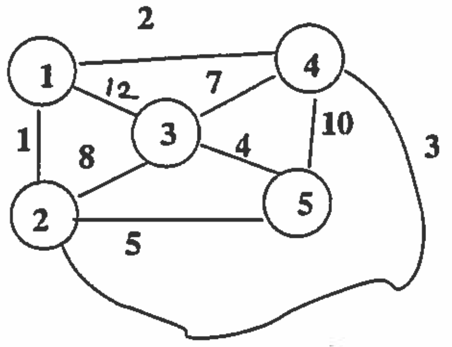
(b) Starting at node a, list the nodes in the order of a breadth-first traversal (when there is a choice, choose the smaller-lettered node first).

a b c d e

(c) Show how the Dijkstra algorithm finds the shortest path from node a to all other nodes. At each step, show what node is selected, what gets changed, and the new distance on each node. If there is a tie, choose alphabetically.

Step	Selected	a	b	c	d	e
1	a	0	∞	∞	∞	∞
2	b	0	1	7	4	∞
3	c	0	[illegible]	4	4	∞
4	d	0	1	4	4	7
5	e	0	1	4	4	7

8. Given the following weighted, undirected graph:

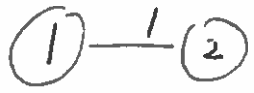


Use the Kruskal algorithm to find a minimal spanning tree of the graph. At each step, list the chosen edge, explain any that are considered but not chosen, and show the changes to the tree in progress. The last step would show the finished tree.

Step 1. Edge Chosen

(1,2)

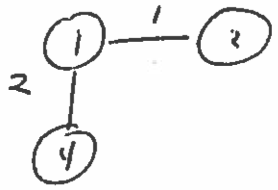
Tree in Progress



Step 2. Edge Chosen

(1,4)

Tree in Progress

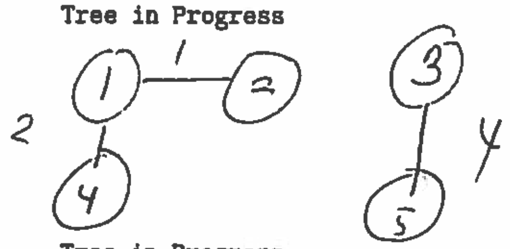


Step 3. Edge Chosen

(2,4) x loop

(3,5)

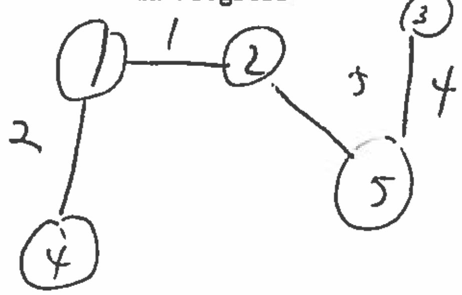
Tree in Progress



Step 4. Edge Chosen

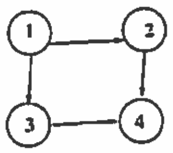
(2,5)

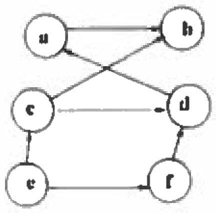
Tree in Progress



9. Given the following small digraph $G_A = (V_A, E_A)$ and larger digraph $G_B = (V_B, E_B)$, find a subgraph isomorphism from V_A to V_B. Veryify that your mapping is indeed a subgraph isomorphism by checking each edge. Show all your work. You only need to do it for **one** isomorphism if there are several.

Your answer is ONE ISOMORPHISM expressed as 4 pairs (no tree needed) and 4 edge checks worked out.





isomorphism

$h(1) = e$

$h(2) = f$

$h(3) = c$

$h(4) = d$

$(1,2) \in G_A$	and	$(e,f) \in G_B$	✓
$(1,3) \in G_A$	and	$(e,c) \in G_B$	✓
$(2,4) \in G_A$	and	$(f,d) \in G_B$	✓
$(3,4) \in G_A$	and	$(c,d) \in G_B$	✓

10. Given the following unsorted array:

0	1	2	3	4	5	6	7
8	57	2	24	100	28	95	1

(a) Show how it would be sorted by QuickSort with pivot 24. (Just the top level, no recursive calls.)

Exchange pivot with LO

24 57 2 8 100 28 95 1 swap

↑ ↑

24 1 2 8 100 28 95 57

↑ ↑ ↑ ↑ ↑ swap w pivot

ok ok ok ok

8 1 2 24 100 28 95 57

(b) Now sort each of the two sides using Insertion Sort, showing each step and the final result.

8 1 2

1 8 2

1 2 8

100 28 95 57

28 100 95 57

28 95 100 57

28 57 95 100

24

1 2 8 24 28 57 95 100

11. A binary tree is implemented with a TreeNode class that has the following public members.

```
public class TreeNode {
    public int value;
    public TreeNode left;
    public TreeNode right;
    ...
}
```

Write a **recursive** function CopyTree(TreeNode root) that is given a reference to the root of a (possibly empty - represented as null) binary tree to be copied. It should build and return a reference to the root of a duplicate copy of the tree. You may use the default constructor for the TreeRoot class to build new TreeNodes.

```
public TreeNode CopyTree(TreeNode root) {
    if (root == null) {
        return null;
    }
   TreeNode copiedRoot = new TreeNode();
   copiedRoot.value = root.value;
   CopiedRoot.left = CopyTree(root.left);
   copiedRoot.right=CopyTree(root.right);
    return copiedRoot;

}
```