CS-261: Data Structure in CS

Practice Final Exam Solution

Question 1:

Find the Big-O complexity for -

$$n ! + 2^n + 5n^3 + 4\ n * \log n$$

Answer: O(n!)

Question 2:

Find the Big-O complexity of the following code fragment -

```
int n, sum = 0;
for (int j = 0; j < 100 * n; j++)
{
for (int i = n; i > 0; i /= 2)
  {
     sum++;
  }
}
```

Answer: O(n * logn)

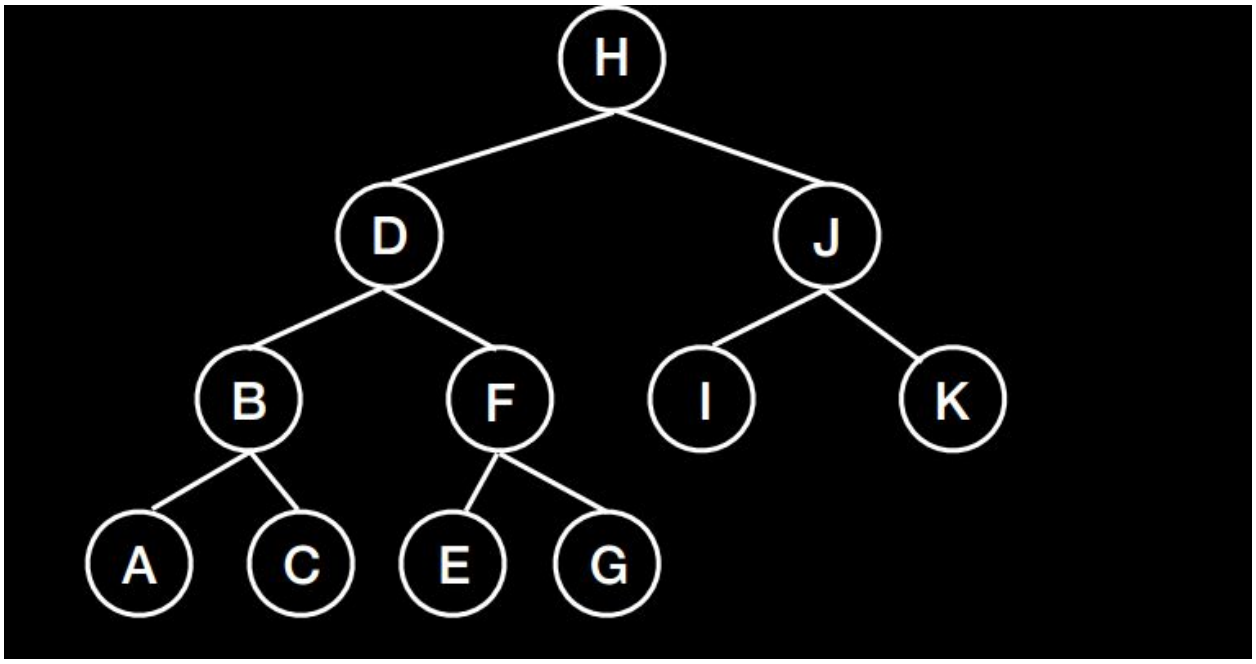
Question 3:

A complete binary search tree is a binary tree in which every level, *except possibly the last*, is completely filled, and all nodes are as far left as possible. Draw a complete binary tree that contains the following letters:

A, B, C, D, E, F, G, H, I, J, K.

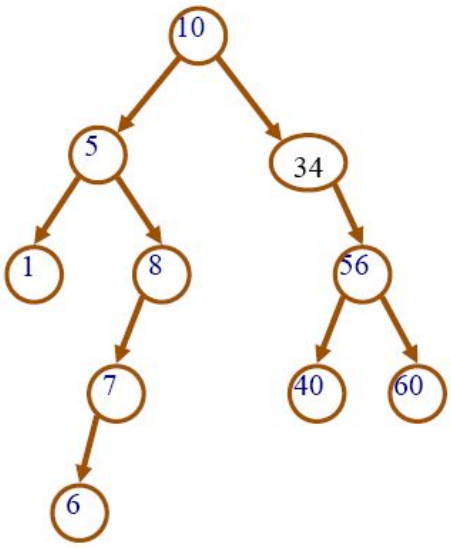
Answer:

See the following tree diagram -



Question 4:

For the following Binary Search Tree, list the nodes as they are visited in a pre-order, in-order, and post-order traversal.



Answer:

Pre-order:
10 – 5 – 1 – 8 – 7 – 6 – 34 – 56 – 40 - 60

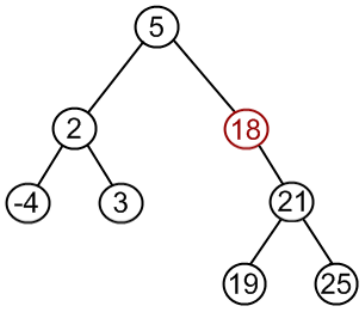
In-order:
1 – 5 – 6 – 7 – 8 – 10 – 34 – 40 – 56 - 60

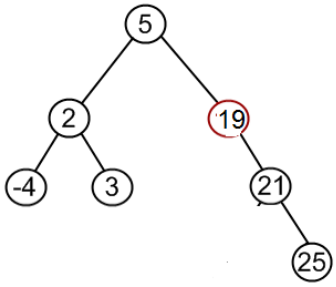
Post–order:
1 – 6 – 7 – 8 – 5 – 40 – 60 – 56 – 34 – 10

Question 5:

For the following **Binary Search Tree**, remove the node with the value 18 and show the resulting BST.

Answer:





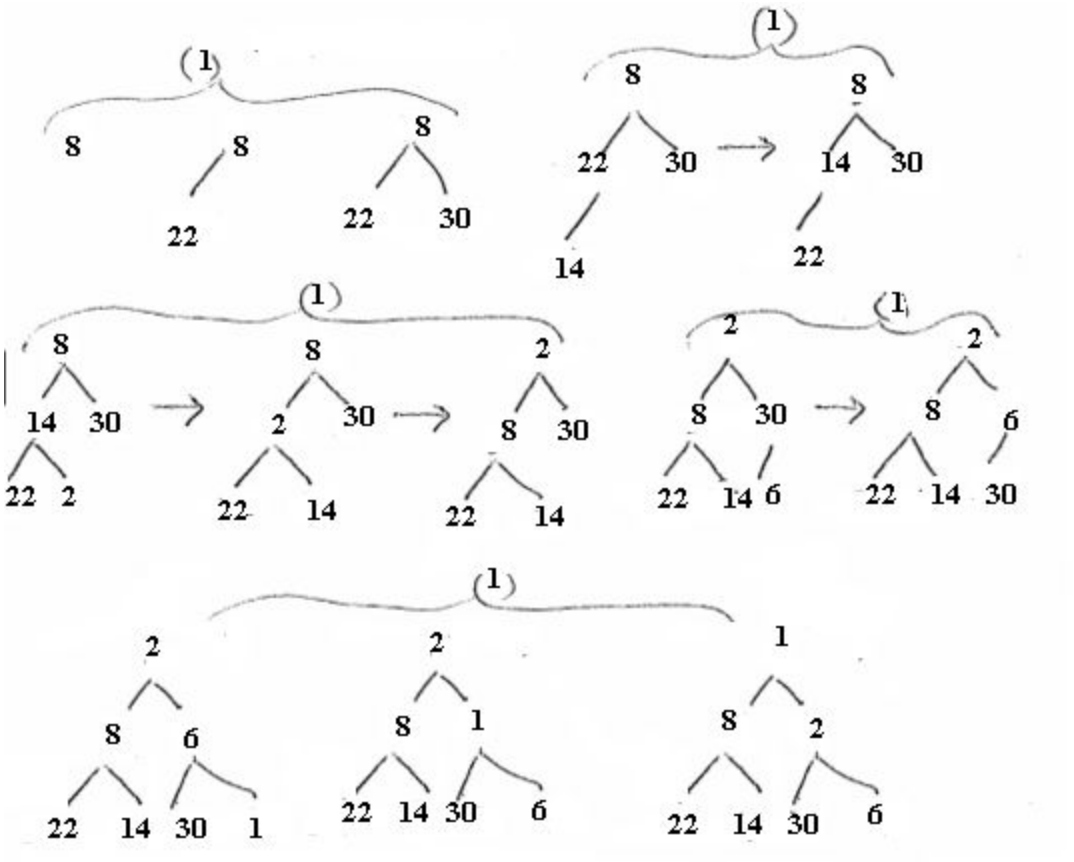
Question 6:

Add the following numbers to a heap (in tree form) in the order given. (Show steps for partial credit)

8, 22, 30, 14, 2, 6, 1

Percolating up:
while new valueis less than parent, swap value with parent

Answer:



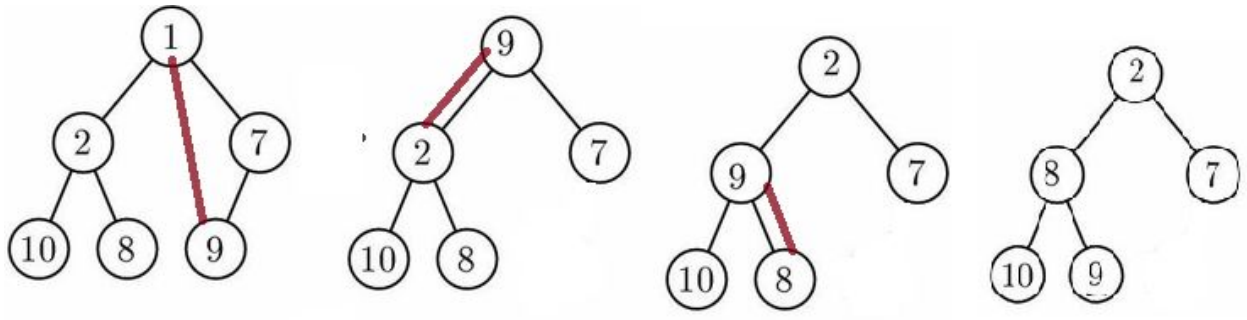
Question 7:

Show the following heap (in graphical tree form) after a call to removeMin (please show all steps) .

Heap removal (removeMin):

1. Replace root with the element in the last filled position
2. Fix heap by "percolating down"

Answer:



Question 8:

Write a treeSort function that takes an array of elements and returns those elements in sorted order. Assume you do not have an iterator (your approach must be recursive). You will need a helper function.

```
struct AVLTree* newAVLTree();

void addAVLTree(struct AVLTree *tree, TYPE val);

void treeSort (TYPE data[], int n)

{ // WRITE ME

}

void _treeSortHelper(AVLNode *cur, TYPE *data, int *count)

{ // WRITE ME

}
```

Answer:

```
void treeSort(TYPE data[], int n){
   int i; int count = 0;

            /* declare an AVL tree */
   struct AVLTree *tree = newAVLtree();
   assert(data != NULL && n > 0);

            /* add elements to the tree */
   for (i = 0; i < n; i++)
      addAVLTree(tree, data[i]);

            /* call the helper function */
   _treeSortHelper(tree->root, data, &count);
}
/* *count goes from 0 to n-1 */

void _treeSortHelper(AVLNode *cur, TYPE *data, int *count){
    if (cur != NULL) {
      _treeSortHelper(cur->left, data, count);
      data[*count] = cur->val;
      (*count)++;
      _treeSortHelper(cur->right, data, count);
   }
```

}

Question 9:

Given the following integer elements:

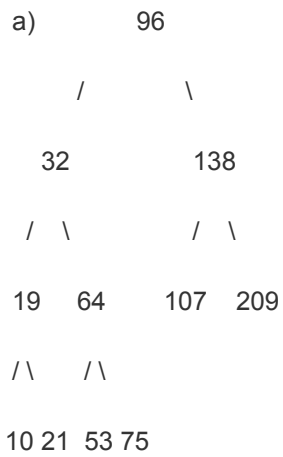
• 138, 21, 10, 96, 209, 107, 64, 32, 75, 53, 19

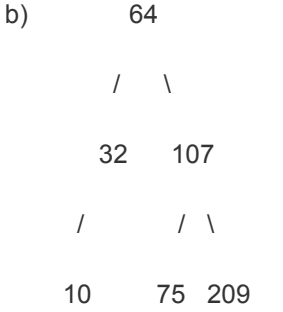
a) Draw the AVL tree that results when the above elements are added (in the given order) to an empty AVL tree.

b) Draw the AVL tree from part (a) after all of the following elements are removed.

• 19, 53, 96, 138, 21

Answer:





Question 10:

The height of any binary search tree with n nodes is always O(log n)

Answer:

False, unbalanced trees may have height n.

Question 11:

Inserting into an AVL tree with n nodes requires looking at O(log n) nodes.

Answer:

True, because AVL trees are balanced.

Question 12:

Inserting into an AVL tree with n nodes require O(log n) rotations

Answer:

False, we need at most 2 rotations.

Question 13:

When do we need to do a double rotation?

Answer:

1) The node is unbalanced and

2) The node's balance factor is positive, but its right subtree's balance factor is negative, but its left subtree's balance factor is positive

Balance factor = height(right subtree) - height(left subtree)

Question 14:

How did we represent a binary heap in memory?

Answer:

Array.

Question 15:

What are the indices for the children of node i?

Answer:

2 * i + 1 and 2 * i + 2

Question 16:

What is the index of the parent of node i?

Answer:

(i - 1) / 2

Question 17:

How do we add a node to a heap?

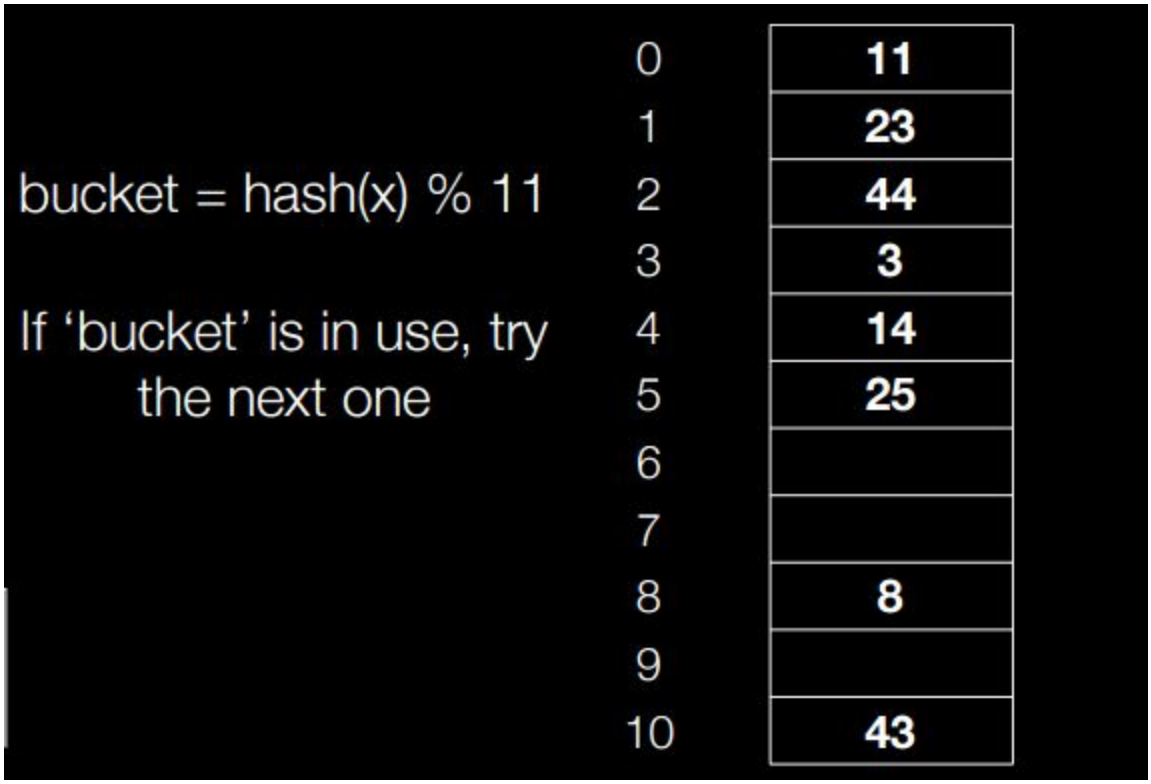
Answer:

Insert it after the last item, then percolate it up.

Question 18 :

If we have a hash table with 11 buckets and the silly hash function hash(x) = x, what will the hash table look like after inserting 3, 43, 8, 11, 14, 25, 23, 44 using open addressing?

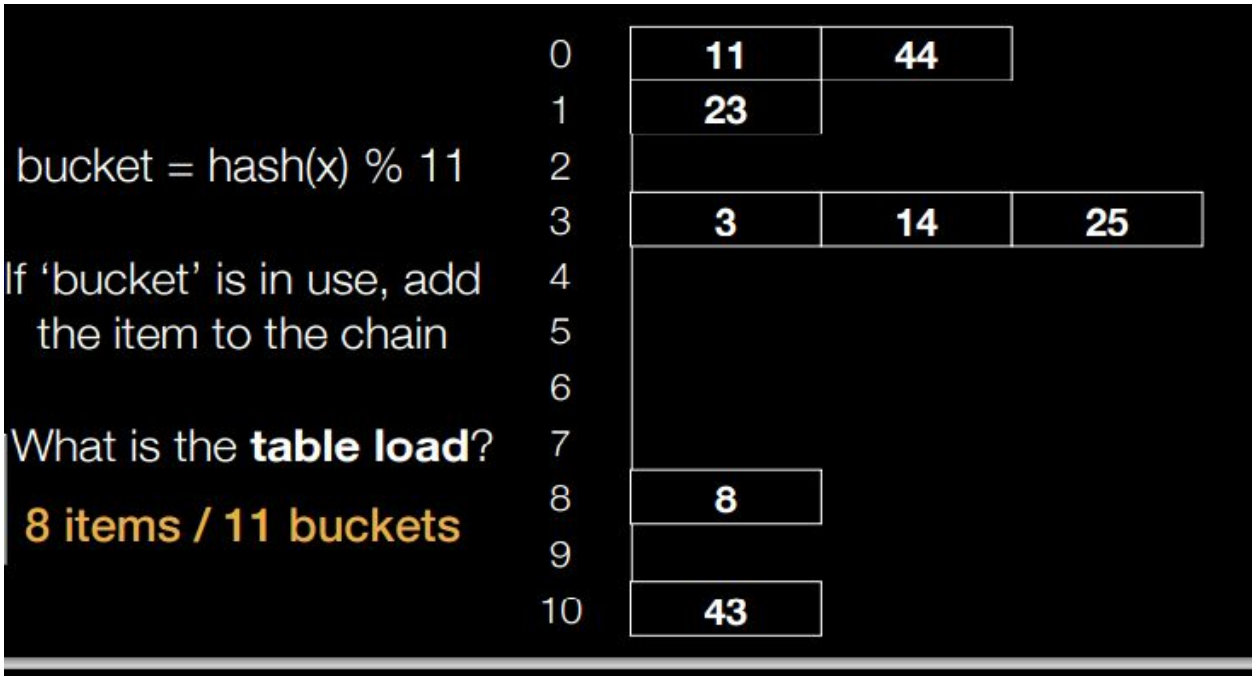
Answer:



Question 19:

If we have a hash table with 11 buckets and the silly hash function hash(x) = x, what will the hash table look like after inserting 3, 43, 8, 11, 14, 25, 23, 44 using buckets + chaining?

Answer:



Question 20:

Does every key in a hash table need to be unique?

Answer:

Yes, that's the point of storing key/value pairs.

Question 21:

Does each key in a hash table need to hash to a unique value?

Answer:

No, but we should use a hash function with as few collisions as possible.

Question 22:

A hash table that uses buckets is really a combination of an array and a linked list. Each element in the array (the hash table) is a header for a linked list. All elements that hash into the same location will be stored in the list.

As with open address hash tables, the load factor (l) is defined as the number of elements divided by the table size. In this structure the load factor can be larger than one, and represents the average number of elements stored in each list, assuming that the hash function distributes elements uniformly over all positions. Since the running time of the contains test and removal is proportional to the length of the list, they

are O(l). Therefore the execution time for hash tables is fast only if the load factor remains small. A typical technique is to resize the table (doubling the size, as with the vector and the open address

hash table) if the load factor becomes larger than 10.

Complete the implementation of the following HashTable function based on these ideas.

```
struct hlink {
TYPE value;
struct hlink *next;
};
struct hashTable {
struct hlink ** table;
int tablesize;
int count;
};
int hashTableContains (struct hashTable * ht, TYPE testElement)
 {
}
```

Answer:

```
int hashTableContains(struct HashTable *ht, TYPE testElement)
 {
int hash = HASH(testElement);
int hashIndex = (int) (labs(hash) % ht.tablesize);
hlink *cur;
/* Also assuming you have a list iterator*/
```

```
cur= ht->table[hashIndex];

while(cur != 0)

{

if(cur->value ==  testElement)

return 1;

cur = cur->next;

}

return 0;

}
```

Question 23:

Does hash table performance increase or decrease as the number of buckets increases?

Answer:

It should increase.

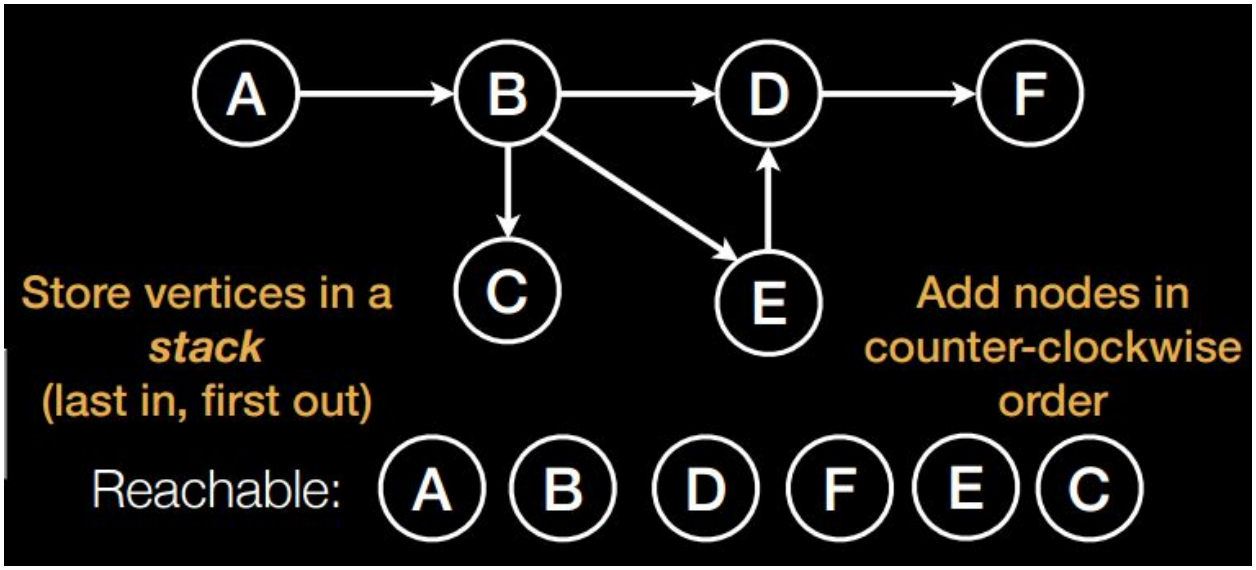
Question 24:

Simulate depth-first search and breadth-first search on this graph -

Graph.png

Answer:

Depth first search:



Breadth first search: