Which of the following is the result of adding the following values (in the order that they are given) to a Binary Search Tree.

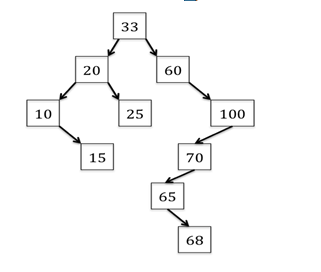
 50, 5, 45, 10, 62, 100, 81, 2

[treeQ#1.pdf](https://oregonstate.instructure.com/courses/1524715/files/62694516/preview)

Your Answer:

D

For the following **Binary Search Tree**, remove the node with the value 60 and show the resulting BST. You may use the following table to save time -- [table\_tree-2.docPreview the documentView in a new window](https://oregonstate.instructure.com/courses/1524715/files/62694353/download?wrap=1)[View in a new window](https://oregonstate.instructure.com/courses/1524715/files/62694527/download?wrap=1)



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

[BST.pdf](https://oregonstate.instructure.com/courses/1524715/files/62694645/preview) /[BST.docx](https://oregonstate.instructure.com/courses/1524715/files/62694630/preview)

Your Answer:

pre-order: node first, then left children, then right children

in-order: examine the left children, then the node, then finally the right children

post-order: examine left children, then right children, then the node

a: Read ***carefully*** and choose the***best*** answer . You will use answers only once.

**Correct!**

**The maximum path length from a node to a leaf node**



**Correct!**

**Data structure used to implement a knowledgebase as in the “guess the animal game”**



**Correct!**

**A tree traversal that produces a reverse polish notation expression for an expression tree**



**Correct!**

**Tree nodes that have the same parent**



**You Answered**

**A tree node that does not have a parent**



**Correct Answer**

Root node

**Correct!**

**Term that describes the path length from the root to a given node**



**Correct!**

**ADT used when implementing an iterative in-order tree traversal iterator.**



**Correct!**

**Number of leaves in a full binary tree of height *n***

2^n

 Write a ***recursive*** function **sumTreeNodeHelper** that sums the elements of a binary search tree starting with the smallest element and adding elements ***in order*** up to the largest element and ***prints the running sum*** as each new value (ie. Node->val) is added to the sum (We’re assuming double values are stored in the tree). ***You will not use an iterator****.*

**void sumTreeNodeHelper(struct BNode \*node, double \*sum)**

**{**

**printf(“ Running sum = %f\n”, \*sum);**

**}**

**void sumTree(struct BSTree \*tree, double \*sum)**

**{**

**sumTreeNodeHelper(tree->root, sum);**

**}**

/\* Example use \*/

struct BSTree myTree;

int sum = 0;

/\* Initialize and Add elements to the tree \*/

addTree(&myTree, 5);

addTree(&myTree, 1);

addTree(&myTree 10);

sumTree(&myTree, &sum);

printf(“ Final sum of tree values = %f\n”, sum);

…< /span>

The output of the above should be:

Running sum = 1

Running sum = 6

Running sum = 16

Final sum of tree values = 16

[Question 5.pdf](https://instructure-uploads.s3.amazonaws.com/account_10020000000097009/attachments/62831224/Question%205.pdf?AWSAccessKeyId=AKIAJFNFXH2V2O7RPCAA&Expires=1439852698&Signature=TeXcJptXNvApRzG3JfBT2cKWTpE%3D)

This question is for getting order of recursive calls correctly w.r.t. to the sum and print. function pass \*sum, which is a pointer here. void sumTreeNodeHelper(struct BNode \*node, double \*sum) { if(node!=0) { sumTreeNodeHelper(Node->left,sum); \*sum = \*sum + node->val; printf("Running sum = %f\n",\*sum); sumTreeNodeHelper(node->right,sum); } } void sumTree(struct BSTree \*tree, double \*sum) { sumTreeNodeHelper(tree->root, sum); }

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

Your Answer:

8 -> 22

8 -> 22 & 30

8 -> 14 & 22 -> 30 is coming off 22

2 -> 8 & 22 -> 14 coming off 8 and 30 coming off 22

2-> 6 & 14 -> 8 coming off 6 and 22 & 30 coming off 14

1 -> 2 & 14 -> 6 & 8 coming off 2, 22 & 30 coming off 14

1

/ \

2  14

/\  / \

6 8 22 30

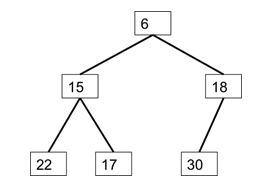
1. Assume the given array in the following represents a min-heap.

[treeQ#8.pdf](https://oregonstate.instructure.com/courses/1524715/files/62694647/preview)

Your Answer:

c

Show the following heap (in graphical tree form) after a call to removeMin (please show all steps ) . Use the table provided in Question #2.



Your Answer:

remove 6

                                             15

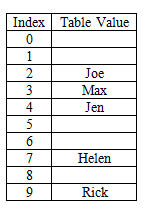
                                           /       \

                                         17       18

                                         /           /

                                        22        30

Assuming that you are using an Open Addressing Hash Table with linear probing and that the hash function uniformly distributes keys. For each ‘unfilled’ position in the hash table below, fill in the probability (as a ratio ... e.g. 5/10) that the next hashed word will be assigned to that position. Note, the hash function is unknown (and not necessary to answer this question). 



What is the load factor of the table in the question above.

5.0

10

0.5

0.75

none of the above

1. For this problem, you must implement the hashTableAdd and hashTableContains functions for an open address hash table with linear probing. Assume you have access to a hash function named \_HASH that returns an integer value and assume that removals are not allowed in this hash table implementation.  **You will find the openHashTable implementation attached with the question.**

[Pseudocode.pdf](https://oregonstate.instructure.com/courses/1524715/files/62694560/preview)

int \_HASH ( TYPE newValue);

void openHashTableAdd (struct openHashTable \* ht, TYPE newValue) {

}

/\* returns 1 for true, 0 for false \*/

int openHashTableContains (struct openHashTable \*ht, TYPE testValue) {

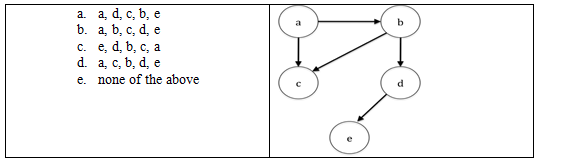
}

A Map ADT must be implemented with a HashTable.

 True

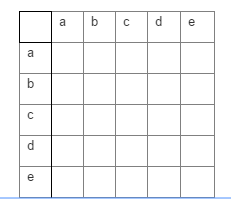
False

 Simulate the depth first search (DFS) reachability algorithm on the following graph starting at the node labeled a (space is given below). For this problem, **NODES MUST BE ADDED TO THE STACK/QUEUE IN CLOCKWISE** order starting at the node directly above. For example, if pushing the neighbors of node a onto a stack, we would push b then c, leaving c on the top of the stack. The generic DFS/BFS pseudocode is provided at the end of the exam. Note that in this version, we do not add neighbors to the stack/queue/pqueue if they have already been visited. Which of the following is the correct order in which the nodes are marked ‘reachable’? 



D

Give the adjacency matrix representation of the graph shown in (12). Assume that any node is connected to itself.



a b c d e a 1 1 1 0 0 b 0 1 1 1 0 c 0 0 1 0 0 d 0 0 0 1 1 e 0 0 0 0 1

What is the big-oh for the space requirements of adjacency matrix representation of a graph?

O(V)

O(E)

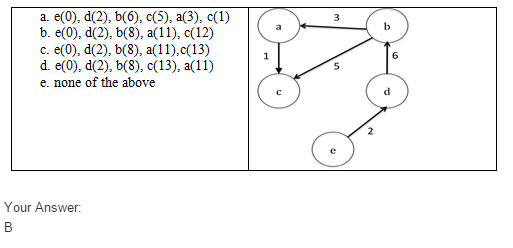
O(E2)

 O(V2)

 O(V+E)

None of the above

 For the following graph, simulated Dijkstra’s algorithm, staring at node *e*. Which of the following represents the order that nodes will be marked ‘reached’ and the correct costs for those nodes.



Suppose you have a graph representing a maze that is infinite in size, but there is a finite path from the start to the finish. Which search, a DFS or a BFS, is ***guaranteed*** to find the path?

Your Answer:

BFS

Fill each black space with the letter from the right side that best matches. Letters may be used ONLY ONCE

**Correct!**

**Time for *finding* an element in a Dynamic Array implementation of a Sorted Bag**



**Correct!**

**Data structure that implements the priority queue interface**



**Correct!**

**Subtrees that are guaranteed to already be heaps**



**Correct!**

**A height-balanced binary search tree**



**You Answered**

**Data structure (not ADT) with O(n) search time**



**Correct Answer**

linked list

**You Answered**

**Resolves collisions by storing them in a collection**



**Correct Answer**

hash table with chaining (or buckets)

**Correct!**

**ADT with the LIFO property**



**You Answered**

**Must have a load factor 1**



**Correct Answer**

open address hash table

**You Answered**

**O(nlog n) in-place general–purpose sorting algorithm**



**Correct Answer**

heap sort

**Correct!**

**Number of nodes in a full binary tree of height n**



**You Answered**

**An in-order traversal of this data structure produces elements in sorted order**



**Correct Answer**

binary search tree

**You Answered**

**ADT used in Dijkstra’s algorithm to implement a cost-first search**



**Correct Answer**

priority queue

**Correct!**

**Time complexity of finding a particular value in a Sorted Linked List**



**You Answered**

**Divide and conquer technique for quickly finding elements (similar to a phonebook search)**



**Correct Answer**

binary search

**You Answered**

**ADT that associates a key with a value**



**Correct Answer**

Map or Dictionary