[3] Draw the Binary Tree that produces the following traversals:

Pre-order: G, X, A, C, Z, I, K

In-order: X, C, A, G, I, K, Z

Order the following operations from fastest (#1) to slowest (#3)

\_\_\_ AVL Tree find

\_\_\_ Vector push\_back

\_\_\_ Linked List find

Order the following operations from fastest (#1) to slowest (#3)

\_\_\_ Vector based queue – dequeue

\_\_\_ Circular queue – dequeue

\_\_\_ Vector based stack – pop

What is the algorithmic efficiency of the following code whose purpose is to traverse a binary tree? **Explain your answer.**

|  |
| --- |
| void virtual traverse(BinaryNode<T> \*node){  if (node == nullptr){  return;  }  cout << node->getValue() << " ";  traverse(node->getLeftChild());  traverse(node->getRightChild());  } |

Insert the value 30 into the AVL tree below. Use the boxes below to show your work. Note: failing to show all of the required steps may result in reduced credit.

|  |  |
| --- | --- |
| **Original** | **After insert** |
|  |  |
| **After 1st Rotation** | **After 2nd Rotation (may or may not be required)** |
|  |  |

Enqueue the value **8** to the following **binary min-heap**:

Start:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 5 | 11 | 6 | 17 | 13 | 20 | 9 | 25 | 21 | 15 |  |

After enqueue:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |

List the three key factors that affect the runtime performance of a hash table:

Using Big-O notation, fill in the missing time complexities for the specified data structure operations.

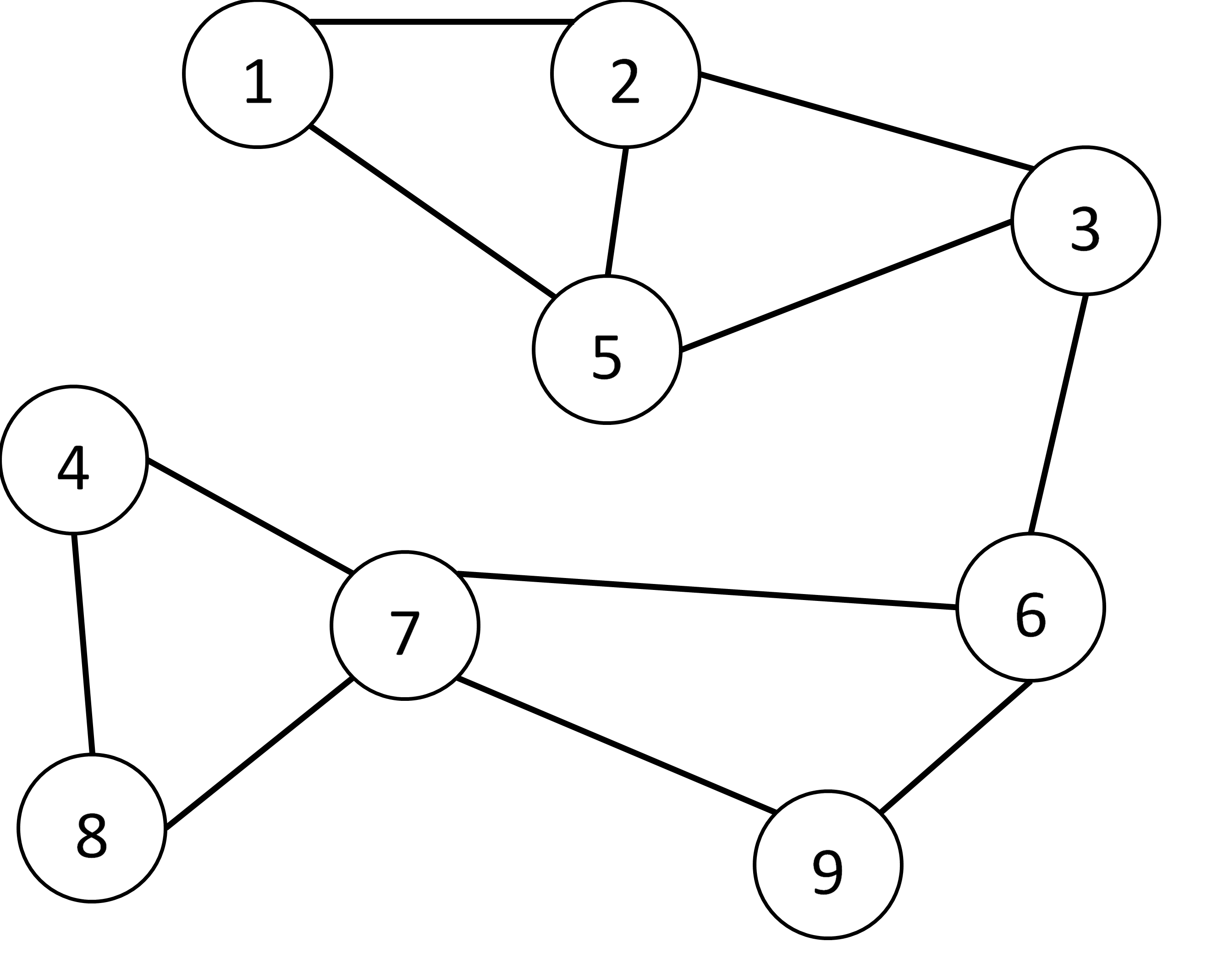
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **AddLast / Push / Enqueue** | **AddAt** | **RemoveLast / Pop / Dequeue** | **RemoveAt** | **Find** | **Resize** |
| **Vector** |  |  |  |  |  |  |
| **Linked List** |  |  |  |  |  | **N/A** |
| **Vector-Based Queue** |  | **N/A** |  | **N/A** | **N/A** |  |
| **Linked List-Based Queue** |  | **N/A** |  | **N/A** | **N/A** | **N/A** |
| **BST** |  | **N/A** |  | **N/A** |  | **N/A** |
| **AVL Tree** |  | **N/A** |  | **N/A** |  | **N/A** |
| **Hashtable** | **N/A** |  | **N/A** |  |  |  |
| **Priority Queue** |  | **N/A** |  | **N/A** | **N/A** | **N/A** |

**Use the space below to justify what you consider to be a controversial decision:**

Show the result of quicksort after one iteration.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | 9 | 12 | 1 | 2 | 15 | 11 | 8 | 10 | 17 | 4 | 6 | 5 | 13 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Articulation Points. Draw the DFS articulation tree for the following graph starting at vertex 7. Circle any articulation points in your tree.



Assume we start with the following unsorted vector

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 5 | 9 | 12 | 1 | 4 | 16 | 7 | 3 | 8 | 2 |

After a single iteration of an unknown sorting algorithm, our vector now looks like:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 5 | 9 | 1 | 4 | 12 | 7 | 3 | 8 | 2 | 16 |

What is the most likely runtime complexity of the sorting algorithm? Why? ***BONUS***: Identify the sorting algorithm that I used.