Data Structures – Final Exam

(Example)

Name:

NYU NetID:

A. Please read the following questions and respond with either T or F. (Correct answers earn +2 points each, incorrect responses result in -1 point, and if left unanswered, no points will be awarded.)
   1. Tree is a non-linear hierarchical data structure. ( )
   2. Level of a Tree means the number of edges from the leaf node to a particular node. ( )
   3. Complete binary tree means a binary tree that is fully occupied. ( )
   4. All key values in each node of Binary Search Tree (BST) are unique. ( )
   5. Time complexity of search, delete and insert operations of BST is always O(log n). ( )
   6. In AVL Tree, the Balance Factor (BF) at each node is. -1, 0 and 1 ( )
   7. In AVL Tree, there are three generalized imbalanced cases ( )
   8. When using Chaining as collision resolution, the size of hash table can be changed ( )
   9. Path of Graph is a sequence of adjacent vertices from vertex u to vertex v. ( )
   10. To get a minimum spanning tree from Graph, Dijkstra's algorithm can be used. ( )

B. Please read the following questions and options, then check the one that corresponds to the correct answer.

1. Choose all operations with a time complexity (Big-O) of log n (e.g., O(log n)). Ignore the worst case of Tree or collision case of Hash Table.

   ① Insert() operation of Hash
   ② Delete() operation of Hash
   ③ Search() operation of Hash
   ④ Insert() operation of BST
   ⑤ Delete() operation of BST
   ⑥ Search() operation of BST

2. The following code is the implementation of search() operation of BST. Choose the most appropriate code to fill in the blank spaces in the following code

```
public TreeNode search(Comparable searchKey) {
    return searchItem(root, searchKey);
}

private TreeNode searchItem(TreeNode tNode, Comparable searchKey) {
    if (tNode == null)
        return null;
    else if (searchKey.compareTo(tNode.key) == 0)
        return tNode;
    else if (/* blank space [1] */)
        return searchItem(/* blank space [2] */, searchKey);
    else
        return searchItem(/* blank space [3] */, searchKey);
}
```

① [1] searchKey.compareTo(tNode.key) > 0, [2] tNode.right, [3] tNode.left

② [1] searchKey.compareTo(tNode.key) > 0, [2] tNode.left, [3] tNode.right

③ [1] searchKey.compareTo(tNode.key) < 0, [2] tNode.left, [3] tNode.right

④ [1] searchKey.compareTo(tNode.key) < 0, [2] tNode.right, [3] tNode.left

C. Please read the following question and fill in the blank space with the appropriate code. (Node: Strict adherence to JAVA syntax is not necessary).

1. The following code is the implementation of insertItem() of AVLTree. Fill in the blank spaces ([1] and [2]) with the appropriate code.

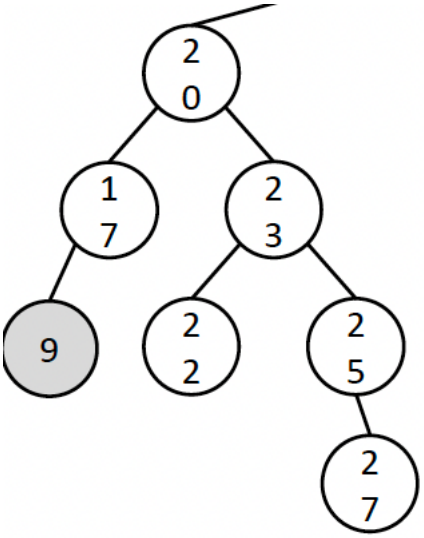
```
private AVLNode insertItem(AVLNode tNode, Comparable x) {
    if (tNode == NIL) {
        tNode = new AVLNode(x);
    } else  if (x.compareTo(tNode.item) < 0) {
        tNode.left = /* blank space [1] */;
        tNode.height = /* blank space [2] */;
        int type = needBalance(tNode);
        if(type != NO_NEED)
            tNode = balanceAVL(tNode, type);
    } else { // branch right
        tNode.right = /* blank space [1] */;
        tNode.height = /* blank space [2] */;
        int type = needBalance(tNode);
        if(type != NO_NEED)
            tNode = balanceAVL(tNode, type);

    }
    return tNode;
}
```

2. The following code is the implementation of Prim's algorithm. Fill in the blank spaces ([1] and [2]) with the appropriate code.

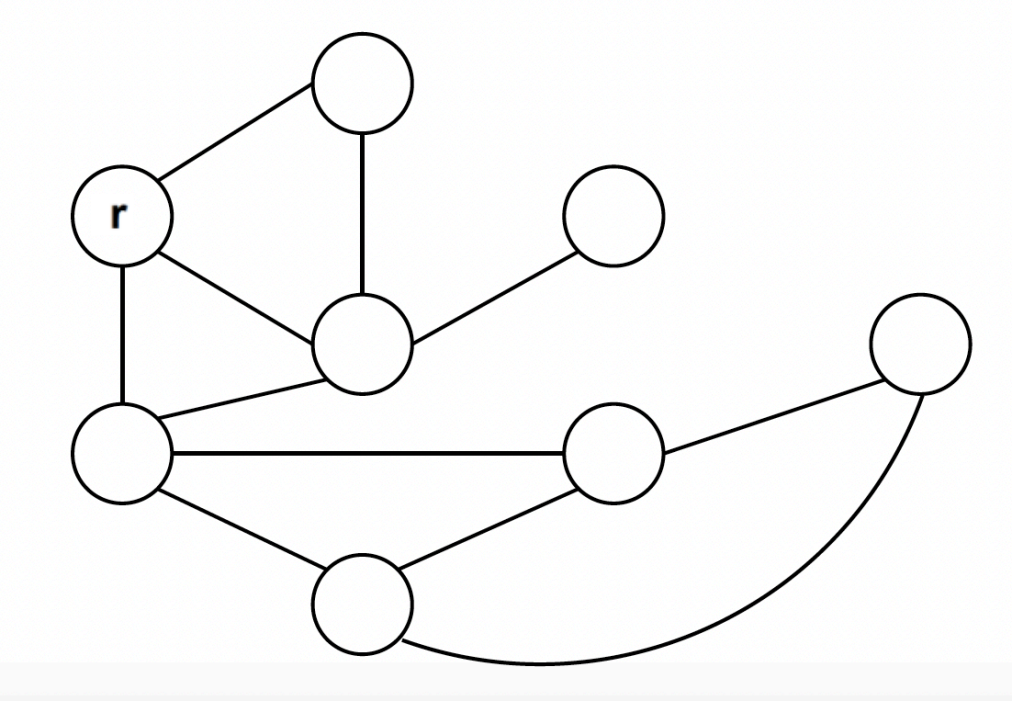
```
// G: Graph
// r: Starting vertex
// V: a set of vertex in G
Prim(G, r):
    S <- {r} // S: a set of vertex
    r.cost <- 0
    for each u in V - {r}
        // weight_{r,u}: weight of edge
        // between vertex r and u
        u.cost <- weight_{r,u}
    while (S != V)
        // deleteMin(Q): returns a vertex u
        // which has smallest u.cost in Q
        u <- deleteMin(V-S)

        include u in set S

        // update the cost of adv vertices
        for each v in u.adj
            if (v in V-S and weight_{u,v} < v.cost)
                /* blank space [1] */
                /* blank space [2] */

                v.cost <- weight_{u,v}
                v.tree <- u
```

D. Please carefully read the following problem and draw an appropriate diagram.

1. The below figure shows the part of an AVL tree. After executing delete(9), the below subtree becomes imbalanced. Draw the balanced subtree after repairing this subtree with the appropriate rotation.



2. The below figure shows an undirected Graph G=(V,E). Mark the visited edges after conducting Breadth-First Search (BFS). Then, we consider that the vertex r is starting vertex.



3. The below figure shows a weighted directed Graph G=(V,E). Shows the shortest path determined by applying Dijkstra's algorithm. Then, we consider that the vertex r is starting vertex.