NANYANG TECHNOLOGICAL UNIVERSITY

SEMESTER 2 EXAMINATION 2021-2022

EE2008 / IM1001– DATA STRUCTURES AND ALGORITHMS

April / May 2022 Time Allowed: 2½ hour

INSTRUCTIONS

1. This paper contains 4 questions and comprises 4 pages.
2. Answer all questions.
3. All questions carry equal marks.
4. This is a closed book examination.
5. Unless specifically stated, all symbols have their usual meanings.

---

1. (a) Consider the mathematical series

$$S(n) = 2 + 3 \cdot 2^3 + 4 \cdot 3^3 + \cdots + (n+1)n^3, \qquad n \geq 1.$$

(i) Design a non-recursive algorithm to compute $S(n)$. If multiplication is identified as the basic operation of the computation, determine the number of multiplications for $S(n)$. What is the time complexity of the algorithm? Use the asymptotic notation $O(g(n))$ with the simplest $g(n)$ possible in your answer.

(ii) Similarly, design a recursive algorithm to compute $S(n)$. If multiplication is again identified as the basic operation of the computation, and denote the number of executions as $M(n)$, set up a recurrence relation for $M(n)$ with appropriate initial condition. What is the time complexity of the algorithm? Use the asymptotic notation $O(g(n))$ with the simplest $g(n)$ possible in your answer.

(iii) Discuss the pros and cons of recursive and non-recursive algorithms for computing $S(n)$ in terms of time complexity and space complexity.

(18 Marks)

Note: Question No. 1 continues on page 2.

(b) Using the set of integers, $A = \{4, 7, 16, 5, 10, 12\}$, build a binary search tree (BST) one by one (i.e., one node at a time) and timely convert it into a balanced tree when necessary. Accordingly, the difference between the heights of each node's left subtree and right subtree is 0, +1 or -1. Draw the final version of the BST and traverse the tree such that the output sequence is in non-decreasing order.

(7 Marks)

2. (a) For a heap of height h, determine the maximum number $n_{max}(h)$ and the minimum number $n_{min}(h)$ of elements in the heap, such that

$$n_{min}(h) \leq n < n_{max}(h) + 1.$$

Determine the height of a heap when the number of elements, n, is specified.

(8 Marks)

(b) Suppose a maxheap has n elements and these elements are stored in an array $H[1..n]$. Some of these elements are leaves. Try some values of n (say, 8, 9, 10, etc.) and determine the array indexes of leaves.

(5 Marks)

(c) A maxheap is implemented using an array $H[1..n]$. The heap elements are distinct integers. In particular, $H[1]$ stores the largest value of the heap. Design an efficient algorithm to find the smallest value of the maxheap and delete it from the heap. Note that the smallest element must reside in one of the leaves.

(12 Marks)

3. (a) Preorder and inorder traversals of a binary tree are given in the following.

Preorder: A B D G E H J C F K L

Inorder: D G B H E J A C K F L

Construct the binary tree from the given preorder and inorder traversals and show the content of the postorder traversal.

(13 Marks)

Note: Question No. 3 continues on page 3.

(b) Consider the AVL tree shown in Figure 1. Show the steps taken when the keys 1, 20 and 35 are inserted in order. Maintain the AVL tree property after each insertion.

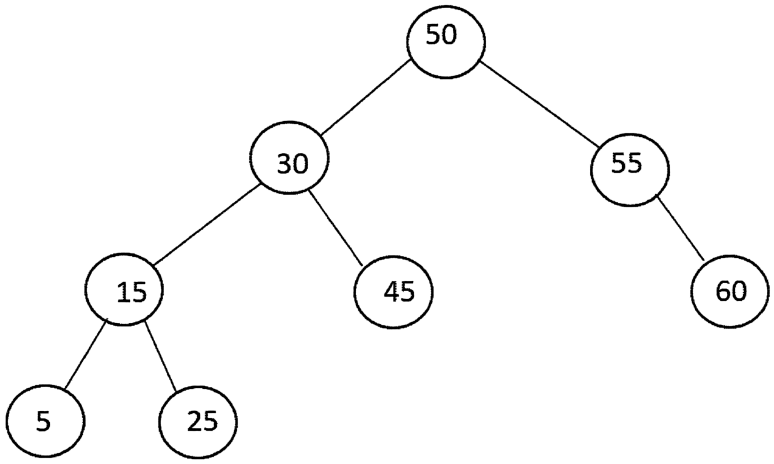


Figure 1

(12 Marks)

4. (a) Determine the asymptotic upper bound for the number of times the statement "$y = y + 1$" is executed in each of the following algorithms:

(i)

```
x = 1
y = 0
while ( x <= n ) {
     x = x * 3
     y = y + 1
}
```

(ii)

```
y = 0
for p = 1 to n-1
   for q = 1 to n-p
     y = y + 1
```

(iii)

```
x = n
y = 0
while (x ≥ (y+1) * (y+1))
  y = y + 1
```

(6 Marks)

Note: Question No. 4 continues on page 4.

(b) Draw the 19-item hash table resulting from hashing the keys 211, 313, 175, 237, 40, 12 and 21 using the hash function $h(x) = x \bmod 19$.

(i) Assume that collisions are handled by linear probing.
(ii) Assume that collisions are handled by double hashing using a second hash function $d(x) = (x \bmod 17) + 1$.

(9 Marks)

(c) An undirected graph $G = (V, E)$ with n nodes is described by a set V of vertexes and a set E of edges. The connectivity of G can be represented in one of two ways: the adjacency lists and the adjacent matrix. The adjacency lists are organized by an array $adj[1..n]$, where the element $adj[i]$ is a pointer pointing to the ith singly linked list. Suppose that the array $adj[1..n]$ and the corresponding n linked lists are given, write an algorithm

```
adjList2Matrix(adj[1..n], A)
```

to generate the $n \times n$ adjacency matrix A. Note that $A[u, v] = 1$ if nodes u and v are connected, and $A[u, v] = 0$ otherwise.

(10 Marks)

END OF PAPER

EE2008 DATA STRUCTURES & ALGORITHMS
IM1001 DATA STRUCTURES & ALGORITHMS

Please read the following instructions carefully:

1. **Please do not turn over the question paper until you are told to do so. Disciplinary action may be taken against you if you do so.**

2. You are not allowed to leave the examination hall unless accompanied by an invigilator. You may raise your hand if you need to communicate with the invigilator.

3. Please write your Matriculation Number on the front of the answer book.

4. Please indicate clearly in the answer book (at the appropriate place) if you are continuing the answer to a question elsewhere in the book.

IE2108-Data Structure and Algorithms AY2021/22 Semester 2 Solution

Author: *CSH*

Disclaimer: *This solution is just for reference. There are various ways to solve a question, so do consult the professor if you find your solution is different from mine, or you feel that my solution is incorrect. All the best in your exams :)*

1(a) Given: $S(n) = 2 + 3 \cdot 2^3 + \ 4 \cdot 3^3 + \cdots + (n+1) \cdot n^3, and\ n \geq 1$

(i) For a non-recursive algorithm, one can run a for loop while loops from 1 to n, then update the value of S(n) in each iteration. The solution is as follow:

```
Input: positive integer S(n)
Output: S(n)
Algorithm S(n) {
    sum = 0;
    //loop through the value i=1 to n
    for (i = 1 to n) {
      //keep track of the value of sum
      sum = sum + (i + 1) · i^3;
    }
    return sum;
}
```

As seen from the above algorithm, since the multiplication is executed once every iteration, thus the number of multiplications is n. Also, since the for loop ran n times, the time complexity is then $O(n)$.

(ii) For a recursive-algorithm, one first need to think of the base condition. The base condition occurs when n=1, in this situation, $S(1) = 2$. For the rest of the cases, the algorithm should return $(n+1) \cdot n^3 + S(n-1)$. The algorithm is as follow:

```
Input: Positive Integer n
Output: S(n)
Algorithm S(n) {
    //base case, S(1) = 2
    if (n == 1):
       return 2;
    else:
      return (n + 1) · n^3 + S(n − 1);
}
```

Since multiplication is the basic operation, and as seen from the else statement, it is only executed once for every recursion. Therefore, the recurrence relation of $M(n)$ is expressed as:

$$M(n) = M(n-1) + 1, and\ M(1) = 0$$

By further expansion, we can get:

$$\begin{aligned} M(n) &= M(n-1) + 1 \\ &= M(n-2) + 2 \\ &= M(n-3) + 3 \\ &\vdots \\ &= M(1) + n - 1 \\ &= n - 1 \end{aligned}$$

For the time complexity, as the multiplication is the basic operation, and we have shown that $M(n) = n - 1$. Therefore, the time complexity of this recursive algorithm is $O(n)$.

(iii)

	Recursive Algorithm	Non-recursive algorithm
Pros	More clarity, thus easier to debug	Smaller space complexity as compared to recursive algorithm.
Cons	Requires more memory to store the previous value, thus greater space complexity	Might be smaller than recursive algorithm for large value of n

1(b) Properties of a Binary Search Tree (BST):

(i) Left subtree of a node contains only nodes with keys less than the node's key

(ii) Right subtree of a node contains only nodes with keys greater than the node's key.

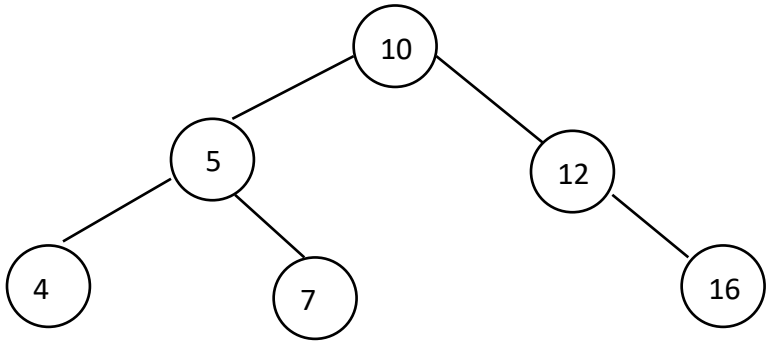
(iii) Left and right subtree must also be a binary search tree

For a balanced tree, the heigh difference between left subtree and right subtree of each node cannot be greater than 1.

To construct this BST based on the given integers, $A = \{4, 7, 16, 5, 10, 12\}$, we first sort it in ascending order, such that we can think of these sorted integers as the in-order traversal of a binary tree. The sorted integers is:

$$A = \{4, 5, 7, 10, 12, 16\}$$

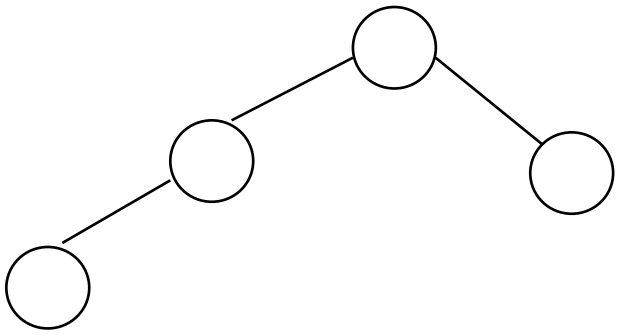
Then, since the in-order traversal is left-root-right. The value at index $\left\lfloor \frac{n}{2} \right\rfloor + 1$ is the root of the BST. Therefore, in this case, the root of the BST is 10. Then, we further divide this array into left-subtree and right-subtree, where left-subtree is $\{4, 5, 7\}$ with root 5 ($\left\lfloor \frac{n}{2} \right\rfloor + 1 = 2$), and right-subtree is $\{12, 16\}$ with root 12 (16 as a root is also correct, but the question requires an non-decreasing order of traversal, so make 12 as the root). Therefore, the final BST will look like:



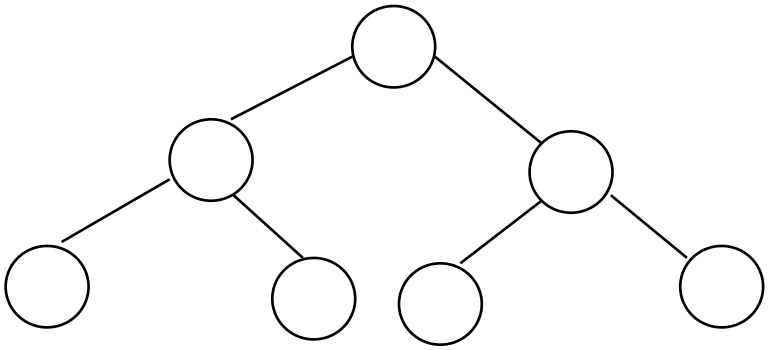
2(a) Given that: $n_{min}(h) \leq n \leq n_{max}(h) + 1$

To find the maximum number of nodes and minimum number of nodes in terms of the heap height, we consider two cases:

(i) Minimum number of nodes occur when the leaf level only consists of leftmost node.



(ii) Maximum number of nodes occur when the leaf level is full



As seen from the above examples, one can easily note that the height of the heap is 2, and $n_{min}(2) = 4$. Meanwhile, $n_{max}(2) = 7$. Therefore, it is intuitive to see that:

$$n_{min}(h) = 2^h, n_{max}(h) = 2^{h+1} - 1$$

If you feel that this approach is not "mathematic" enough, recall that $h = \lfloor \lg(n) \rfloor$. Therefore, we can deduce:

$$h = \lfloor \lg(n) \rfloor$$
$$\because \lg(n) - 1 < \lfloor \lg(n) \rfloor \leq \lg(n)$$
$$\therefore \lg(n) - 1 < h \leq \lg(n)$$

From the above inequality, we can get:

$$\mathrm{h} > \lg(n) - 1$$
$$\therefore \mathrm{n} < 2^{h+1}$$

Also,

$$h \leq \lg(n)$$
$$\therefore \mathrm{n} \geq 2^h$$

As a result, the range of n is given as:

$$2^h \leq \mathrm{n} < 2^{h+1}$$

By simple algebra, one can easily solve for $n_{min}(h)$ and $n_{max}(h)$. The results obtained should match with the result shown above.

2(b) Given a heap array $H[1 \ldots n]$, since the index of the left child of a root with index i, is $2i$, while the right child is $2i + 1$. We first prove that $\left\lfloor \frac{n}{2} \right\rfloor + 1$ is the index for the first leaf (leftmost)

To prove that $\left\lfloor \frac{n}{2} \right\rfloor + 1$ is the index for the first leaf, we simply prove that the left child of $\left\lfloor \frac{n}{2} \right\rfloor + 1$ is greater than n. If this statement is true, then the index for leaf node is from $\left\lfloor \frac{n}{2} \right\rfloor + 1$ to n.

According to the fact that $\left\lfloor \frac{n}{2} \right\rfloor + 1 > \frac{n}{2} - 1 + 1 = \frac{n}{2}$. Therefore, left child of $\left\lfloor \frac{n}{2} \right\rfloor + 1 = 2(\left\lfloor \frac{n}{2} \right\rfloor + 1) > n$.

As a result, one can conclude that the index for leaf node is $\left\lfloor \frac{n}{2} \right\rfloor + 1 \leq i \leq n$.

2(c) According 2(b), we know that the index of the leaf node is $\left\lfloor \frac{n}{2} \right\rfloor + 1 \leq i \leq n$. Therefore, one can simply run a for loop from $i = \left\lfloor \frac{n}{2} \right\rfloor + 1$ to n, then keep track of the smallest value, then delete it. For the deletion, one can simply replace the deleted element by the last element, then heapify again to remain the heap structure. The algorithm is as follow:

```
Input: Max heap array H[1 … n]
Output: Max heap array with minimum element removed H[1 … n-1]
remove_min(n) {
        min_index = ⌊n/2⌋ + 1
        for (i = ⌊n/2⌋ + 2 to n) {
           if (H[i] < H[min_index]) {
                 min_index = i;
            }
        }

        //replace the deleted element by the last element
        H[min_index] = H[n];

        //decrease the size of the array by 1
        n = n – 1;
        heapify(H[n]);

        return H[n]
}
```

3(a) Given that:

(i) Pre-order traversal: A B D G E H J C F K L

(ii) In-order traversal: D G B H E J A C K F L

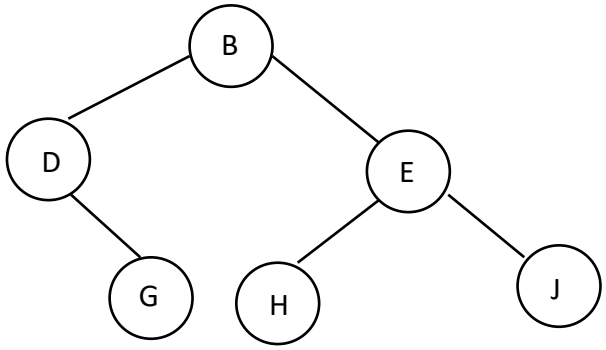
This question may seem difficult at the first look, some of my friends even spend 30 minuets to draw out the projected version of in-order traversal to get the binary tree. However, this question is actually very simple, one just need to know the basic concept. That is, the sequence for pre-order traversal is root-left-right, while the sequence for in-order traversal is left-root-right. With this knowledge, let's start solving this question:

The first element of the pre-order traversal must be the root of the binary tree. Therefore, we split the in-order traversal array into left-subtree and right-subtree corresponds to the position of A. Why? This is because the sequence of in-order is left-root-right, so to the left of A must be its left-subtree, while to the right of A must be its right-subtree. The in-order traversal array now looks like this:

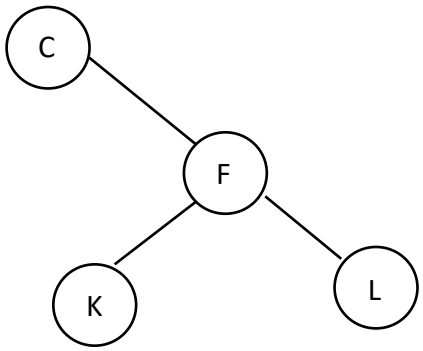
D G B H E J A C K F L

We first solve for the left-subtree of A (yellow highlighted) For the second element of the pre-order traversal, it must be the root of the left-subtree of A. Therefore, using the same analogy stated above, {D, G} is the left-subtree of B, while {H, E, J} is the right-subtree of B. Also note that in both the pre-order traversal and in-order traversal, G is traversed after D. This simply means that G must be the right child of D. To prove this, one can see that if G is the left-child of D, then the in-order traversal will be: G D B H …

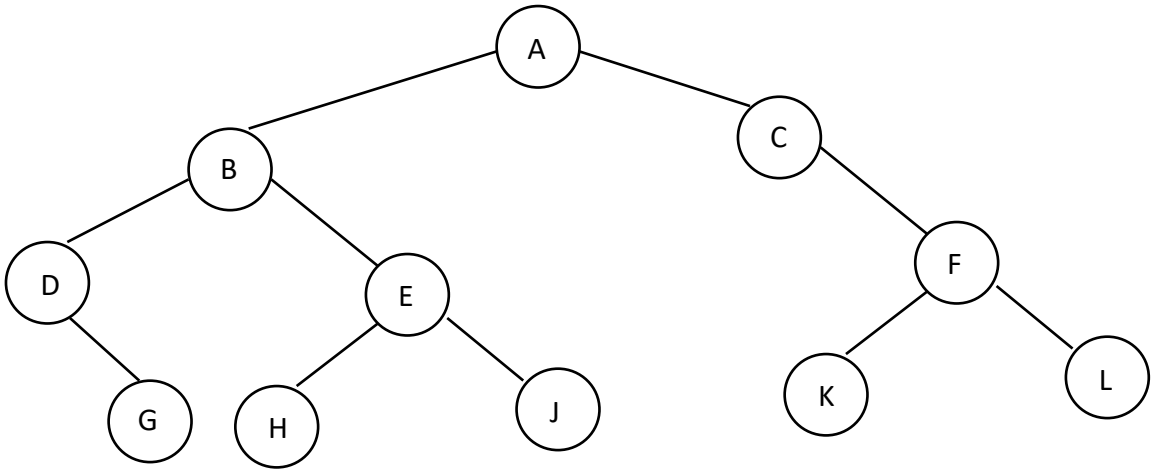
As for {H, E, J}. Since E is the first to be traversed in pre-order traversal, so it must be the root of the right-subtree. Meanwhile, H and J are the left and right child of H respectively. This result can be easily seen by applying the previous analogy. Therefore, the left-subtree of A will look like:



We now deal with the right-subtree of A, i.e. {C, K, F, L}. Since C is the first to be traversed according to pre-order traversal. Therefore, the root of this right-subtree must be C and C and no left-subtree but right-subtree of {K, F, L}. Applying the same analogy stated above, one can conclude that the root of the right-subtree of C is F while K and L are its left and right child respectively. Therefore, the right-subtree of A looks like:

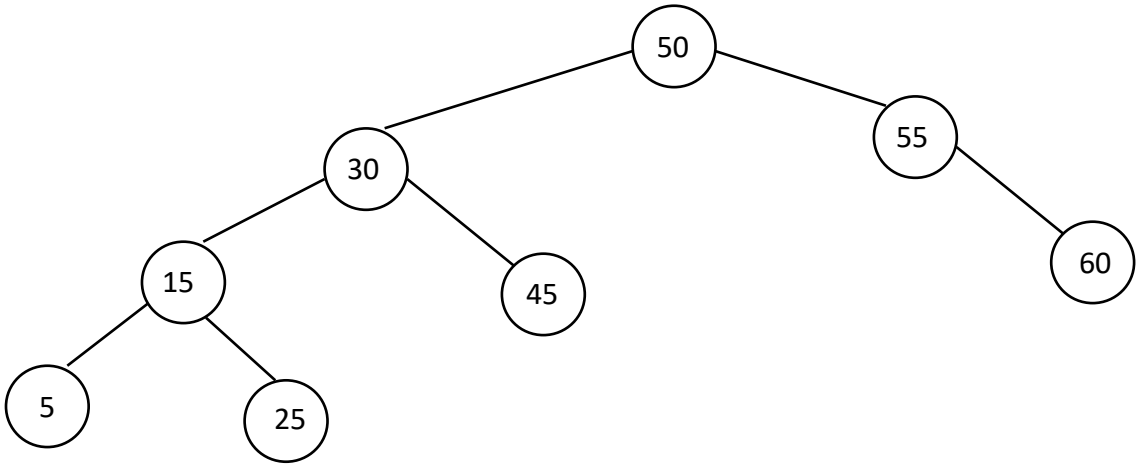


By combining these two sub-trees, we get the final binary tree to be:

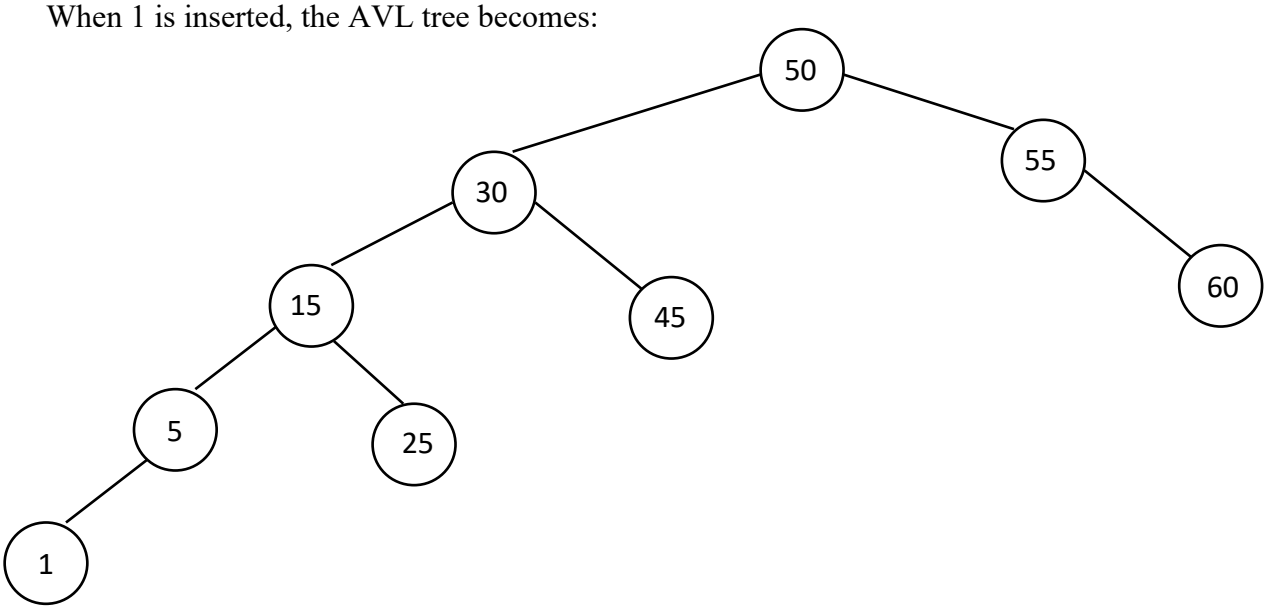


According to the binary tree, the post-order traversal will be:
Post-order traversal: G D H J E B K L F C A

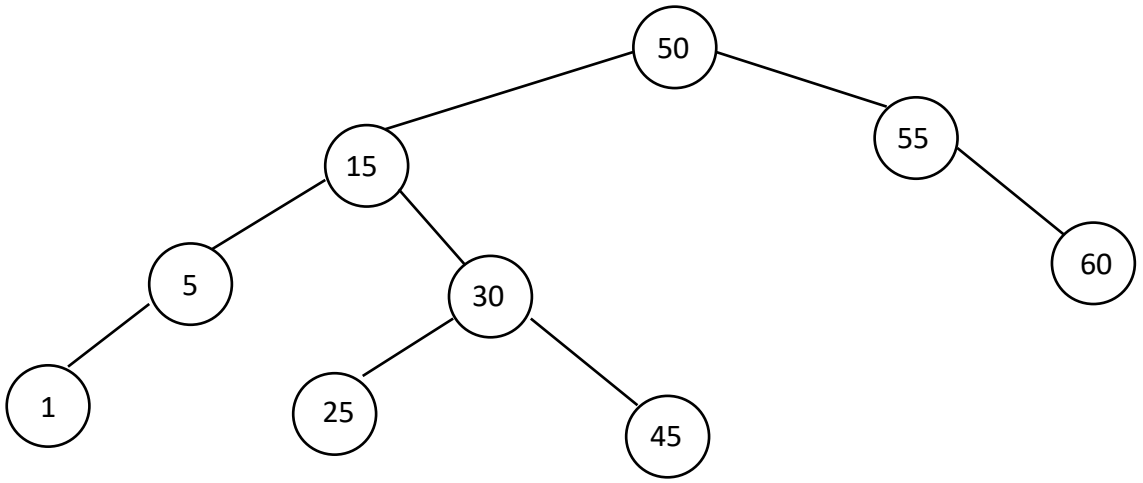
3(b)



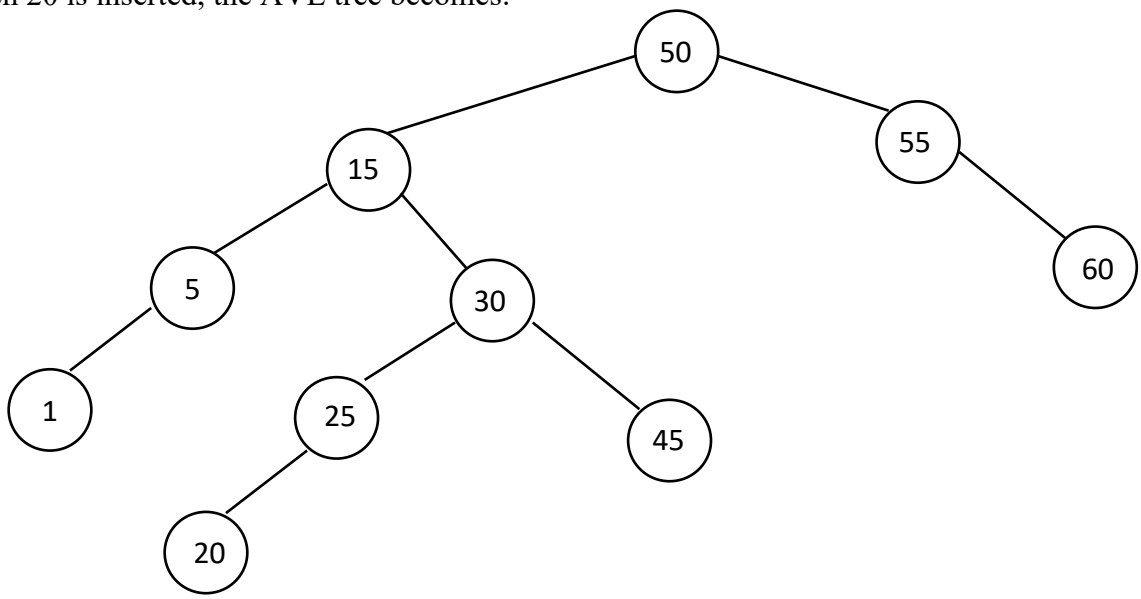
(i) When 1 is inserted, the AVL tree becomes:



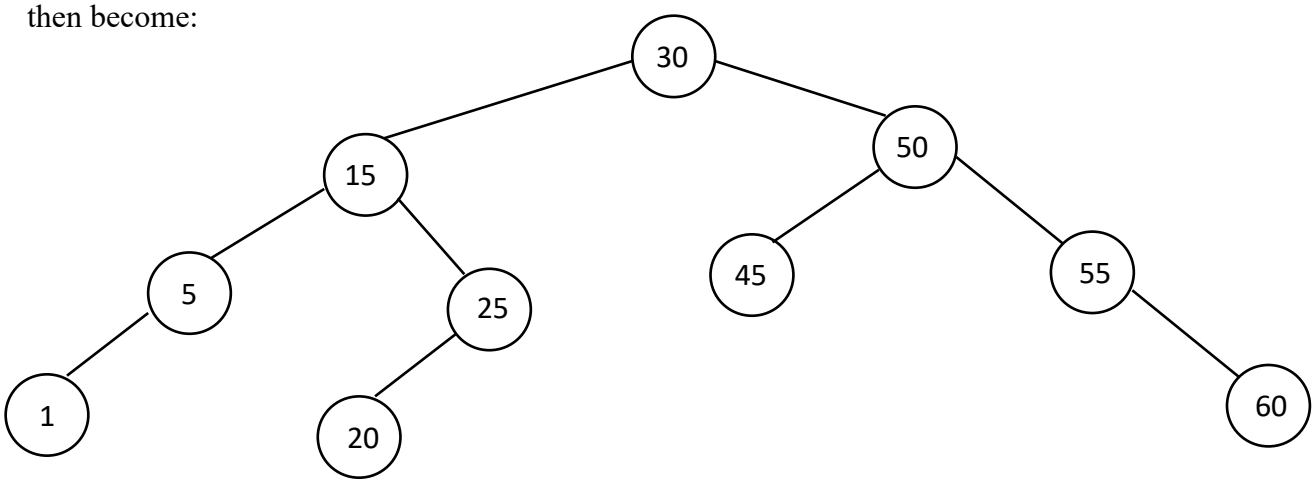
This is a left-left case where the imbalance occurs at node 30. Therefore, we swap the position of node 15 and node 30, then make node 30 the right-subtree of node 15, and node 25 the left-child of node 30. The AVL tree will become:



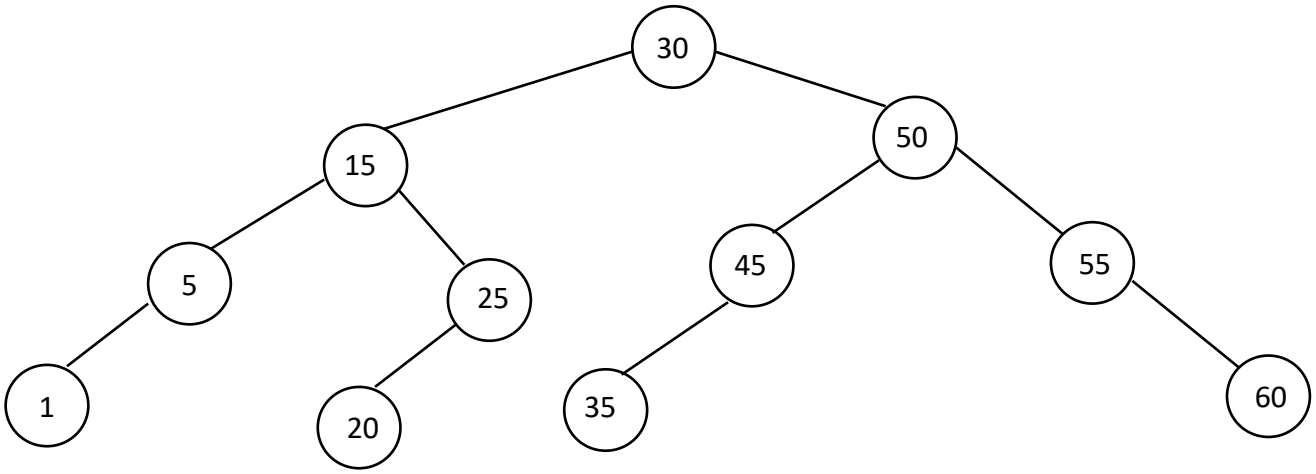
(ii) When 20 is inserted, the AVL tree becomes:



This is a left-right case where the imbalance occurs at node 50. Therefore, we swap the position of node 30 and node 50. Make node 50 the right-subtree of node 30, node 45 the left-child of node 55. Make node 15 the left-subtree of node 30, node 25 the right-subtree of node 15. The AVL tree will then become:



(iii) When 35 is inserted, the AVL tree becomes:



Since this AVL tree has no imbalance, this is also the final AVL tree.

4(a)

(i) The algorithm will stop when $3^k > n$ where k is the number of executions of $y = y + 1$. Therefore:

$$k > \ log_3 n$$

The upper bound of the number of times of the statement is then: $O(log_3 n)$

(ii) Let the number of executions of $y = y + 1$ be k.

$$\begin{aligned} k &= \sum_{p=1}^{n-1} \sum_{q=1}^{n-p} 1 \\ &= \sum_{p=1}^{n-1} (n-p) \\ &= n(n-1) - \left(\frac{n(n-1)}{2}\right) \\ &\leq n^2 \end{aligned}$$

Therefore, the asymptotic upper bound is $O(n^2)$

(iii) For this algorithm, it will stop when $(y+1)^2 > n$, one can note that each addition of $y + 1$ inside the while loop corresponds to one execution of $y = y + 1$. Therefore, we can think of this condition as:

$$k^2 > n, where\ k\ is\ the\ number\ of\ execution\ of\ y = y + 1$$

Therefore, $k > \sqrt{n}$. Thus, the asymptotic upper bound is $O(\sqrt{n})$

4(b) Keys = {211, 313, 175, 237, 40, 12, 21}

(i) If the collisions are handled by linear probing

	0	1	2	3	4	5	6	7	8	9
Probes			211	40	175	21				313

	10	11	12	13	14	15	16	17	18
Probes	237		12						

(ii) If the collisions are handled by double hashing

Probes = $h(x) + i \cdot d(x)$

	0	1	2	3	4	5	6	7	8	9
Probes			211		175			237		313

	10	11	12	13	14	15	16	17	18
Probes			12				40	21	

4(c) This question is relatively easier to solve as compared to convert an adjacency matrix to an adjacency list. To solve this question, one can simply loop through each pointer pointing to the ith singly linked list. Then, we check for all the neighbors in the linked list and update $A[i][neighbor] = 1$. When we reach the end of the linked list (i.e., there is no more neighbors), we move on to the next singly linked list. The algorithm is as follow:

```
Input: adjacency list and adjacency matrix A
Output: adjacency matrix A
adjList2Matrix(adj[1…n], A) {
        n = adj.last;
        A = new_matrix[n][n];

        //initialize the value of matrix to be 0
        for (i = 1 to n) {
            for (j = 1 to n) {
                A[i][j] = 0;
           }
        }

        //loop through the adjacency list
        for (i=1 to n) {
            ref = adj[i];
           //continue to check for neighbors until there is none
           while (ref != null) {
                A[i][ref.data] = 1;
                ref = ref.next
           }
        }
        return A;
}
```