# CS 102: Tier - 1 Examination

Full time for this examination is 90 mts
Mention your name and roll number in each page of the paper in the space provided.
Answer the questions in the spaces provided on the question sheets.
**No partial credit without appropriate (be brief) explanation**

Name: Soultions

Roll No: Solutions

(Feb 05, 2020)

1. (5 points) Select the dominant term of each expression in the table below which gives running time of an algorithm for solving a problem having input size $n$. Specify the Big-Oh complexity in each case.

| Expression | Dominant term | O(···) |
|---|---|---|
| $25 + 0.0017n^3 + 0.021n$ | $0.0017n^3$ | $O(n^3)$ |
| $n^2 \log n + n(\log n)^2$ | $n^2 \log n$ | $O(n^2 \log n)$ |
| $0.01n + 110n^2$ | $110n^2$ | $O(n^2)$ |
| $0.3n \log n + 1.5n^{1.5} + 2.5n^{1.75}$ | $2.5n^{1.75}$ | $O(n^{1.75})$ |
| $0.004 \log_4 n + \log_2 \log_2 n$ | $0.004 \log_4 n$ | $O(\log n)$ |

2. (6 points) Algorithms A and B spend exactly $T_A(n) = 5n^2 log_{10} n$ and $T_B(n) = 25n^2$ microseconds, respectively, for a problem of size $n$. Choose the algorithm, which is better in the Big-Oh sense, and find out a problem size $n_0$ such that for any larger size $n > n_0$ the chosen algorithm outperforms the other. If your problems are of the size $n \leq 10^9$, which algorithm will you recommend to use?

**Solution:** In the Big-Oh sense, the algorithm B is definitely better. It outperforms the algorithm A when $T_B(n)$ is exceeded by $T_A(n)$. So, $25n \leq 5n \log_{10} n$ implies $\log_{10} n \geq 5$, or $n \geq 100,000$. So, for any value of $n > n_0 (= 10^5)$ algorithm B is better. Hence, if problem size is $10^9$, algorithm B is recommended.

3. (6 points) Find the running time of following code snippet in big-Oh notation.

```
for(i=n; i>0; i/=2) {
    for(j=1; j<n; j*=2) {
        for(k=0; k<n; k+=2) {
            // constant number of operations
        }
    }
}
```

**Solution**: The outer loop repeatedly halves the loop index starting with value $n$. So it is performed $\log n$ times. The inner loop is also executed $\log n$ times because it initializes $j = 1$ and continues untill $j < n$. Since, $j$ being doubled after each iteration, the number of iterations can be $\log n$. But the innermost loop executed for $n/2$ times. So, the total time $(n/2)(\log n)^2$ times. In big-Oh sense the running time is $O(n(\log n)^2)$.

4. (6 points) A pseudo code for finding stack size is give below.

```
int stackSize(STACK S) {
    int size = 0;
    while (!isEmpty(S)) {
        Pop(S);
        size++;
    }
    return size;
}
```

The problem with the above code is that it destroys the stack.

A template for a recursive algorithm for computing stack size without destroying the stack is given below. Unfortunately, some missing parts of the code is missing.

```
int stackSize(STACK S) {
    int x;
    int count;
    if (isEmpty(S))
        return 0;    // Base case of recursion
    ---- = pop(S);
    count = ----;
    push(S, x);
    return -----;
}
```

Provide the missing part of the pseudo code for functional algorithm. Use of extra variable or functions are not permitted except for a recursive call.

**Solution:**

1: x

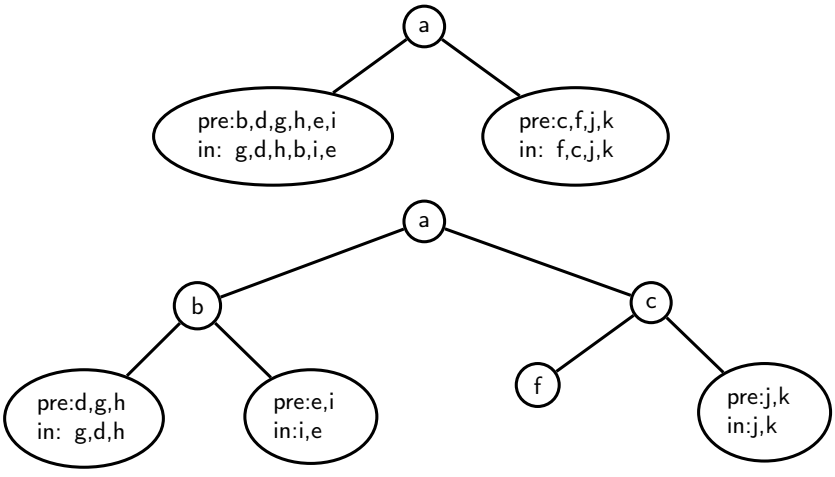
2: stackSize(S)

3: count+1

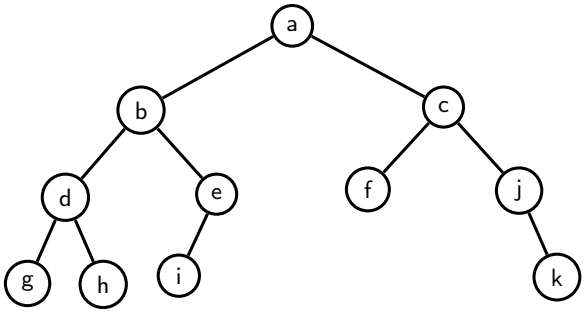
5. (6 points) The inorder and preorder traversal lists of a binary tree are give below:

   1. Preorder list: a, b, d, g, h, e, i, c, f, j, k
   2. Inorder list: g, d, h, b, i, e, a, f, c, j, k

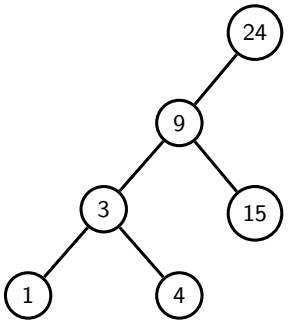
**Solution:** Preorder traversal follows rLR traversal rule whereas inorder traversal follows LrR traversal rule. So from preorder we recover the root of the remaining subtree and from inorder sequence we can recover its left and right subtrees. We follow this principles for recursively decomposing the subtrees till we can recover individual nodes whose LST and RST will be null. So the sequence of steps for recovery of tree will be as shown below:



**Solution (contd):**




6. (18 points) Consider the following binary search tree.




(a) (6 points) Find out all possible insertion sequences that may have produced the above tree.

(b) (6 points) Draw the tree after insertion of keys 6, 45, 32, 98, 55, 69 to the above tree.

(c) (6 points) Draw the tree after deletion of keys 9 and 55. Briefly describe the steps executed for each deletion in plain English.

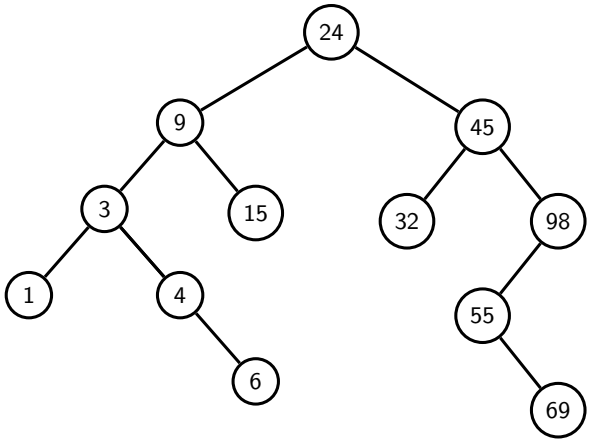
**Solution (a):** Insertions of 24, 9 and 3 must in the that order. Furthermore, 15 can be inserted only after 24 and 9 have been inserted. So effectively,

- 1, 4 can be inserted in any order
- 15 can come in any order between insertion of 1 and 4.

Implying that for generating above tree, 1, 4, and 15 in can be inserted in any order after 24, 9 and 3 have been. Hence total 3! = 6 possible sequences can be found, namely:

1. 24, 9, 3, 1, 4, 15
2. 24, 9, 3, 4, 1, 15
3. 24, 9, 3, 15, 1, 4
4. 24, 9, 3, 15, 4, 1
5. 24, 9, 15, 3, 1, 4
6. 24, 9, 15, 3, 4, 1

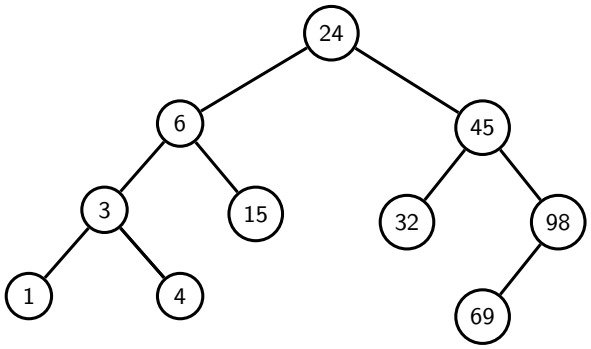
**Solution (b):**




**Solution (c):** The three steps for deletion of 9 are:

1. Find the inorder predecessor of 9, i.e., 6.
2. Swap 9 and 6. 9 becomes leaf.
3. Delete 9.

For deletion of 55, just splice out 55 by letting 98 adopt 69 as its left child.

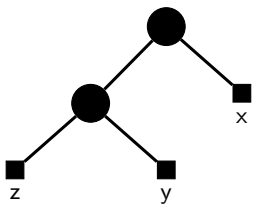



7. (3 points) Give the range of the number of nodes that a binary heap of height $h$ may have?

**Solution:** A full binary tree of height $h-1$ has $2^h - 1$ nodes. Similarly a full binary tree of height $h$ has $2^{h+1} - 1$ nodes. So the number of nodes in a binary heap may be between $2^h$ and $2^{h+1} - 1$.

8. (5 points) Can a legal red black tree have one black child and no red child? If not, explain why? Use appropriate illustrations in support of your answer.

**Solution:** If a black node has exactly one black child and no red child then the black depth black child is 1 greater than the external node of its parent. The configuration of such a red black tree is shown in the diagram below.



**Solution (contd):** As may be seen from the diagram black depth of x is 2, whereas black depth of both y and z are 2 each. Therefore, configuration violates the property of that each external node should have equal black depth.

Space for Rough Work