Ques 1. Give the Best, Average and Worst case examples and complexities of the following algorithms:

- **a) Insertion Sort**
- **b) Quick Sort**
- **c) Bucket Sort**

Ans:

Insertion Sort

a) Best Case:	$O(n)$	Eg: [1,2,3,4,5]

(The best case input is an array that is already sorted)

b) Average Case:	$O(n^2)$	Eg: [5,4,3,2,1]
c) Worst Case:	$O(n^2)$	Eg: [3,2,1,5,4]

(The simplest worst case input is an array sorted in reverse order.)

Quick Sort

a) Best Case:	O(n log n)	Eg: [2,1,5,4,3]

(The best case of quicksort occurs when the pivot we pick happens to divide the array into two exactly equal parts, in every step)

b) Average Case:	O(n log n)	Eg: [2,1,4,3,5]
c) Worst case:	$O(n^2)$	Eg: [1,2,3,4,5]

The worst case of quicksort occurs when the pivot divides the array into exactly n-1 and 1 sub-parts and this happens when the array is already sorted.

Bucket Sort

a) Best Case:	O(n + k)	[1,2,3,4,5]
b) Average Case:	$O(n + k)$	[1,4,2,11,14]
c) Worst case:	$O(n^2)$	[1,11,21,31,41,51]

(Worst case occurs when all elements are allocated to the same bucket.)

Solve the following Recurrence Relations:

a) $T(n) = 16\,T(n/4) + n^2$

Ans:

We have a = 16, b = 4, and $f(n) = n^2$.

Case (ii) of the Master Theorem applies because $n^2 = \Theta(n^{\log_4 16} = n^2)$. Thus, $\mathbf{T(n) = \Theta(n^2 \log n)}$**.**

b) $T(n) = T(9n/10) + n$

We have a = 1, b = 10/9, f(n) = n. Case 3 applies, thus $\mathbf{T(n) = \Theta(n)}$**.**

Ques 2:

a) What makes an algorithm Randomized? Is Randomized Quicksort better from normal Quicksort? Give appropriate examples to prove your points.

Ans:

1) An algorithm is randomized if its behavior is determined in part by values produced by a random-number generator.
2) Yes, Randomized Quicksort is better than normal quicksort as Randomization of quicksort stops any specific type of array from causing worst case behavior.
3) For example, an already-sorted array causes worst-case behavior in non-randomized QUICKSORT, but not in RANDOMIZED-QUICKSORT.

b) *"For any choice of hash functions, there exists a bad set of keys that assign all hash to the same slot."* Justify the above statement by giving appropriate examples. Suggest a solution with examples, in order to remove the above weakness of hashing.

Ans:

1) Eg: In case of division method in Hashing: H(k) = k mod m, where m is the no. of slots.
   Say m=10 and the no. are as follows: [2, 12, 22, 32, 42, 52] these all result in being allocated to the same slot. Hence the statement is justified.
2) Solution: To choose hash functions at random which is independent of the number.

Ques 3:

a) **Show the insertion of following elements in an empty BST, along with algorithm:**

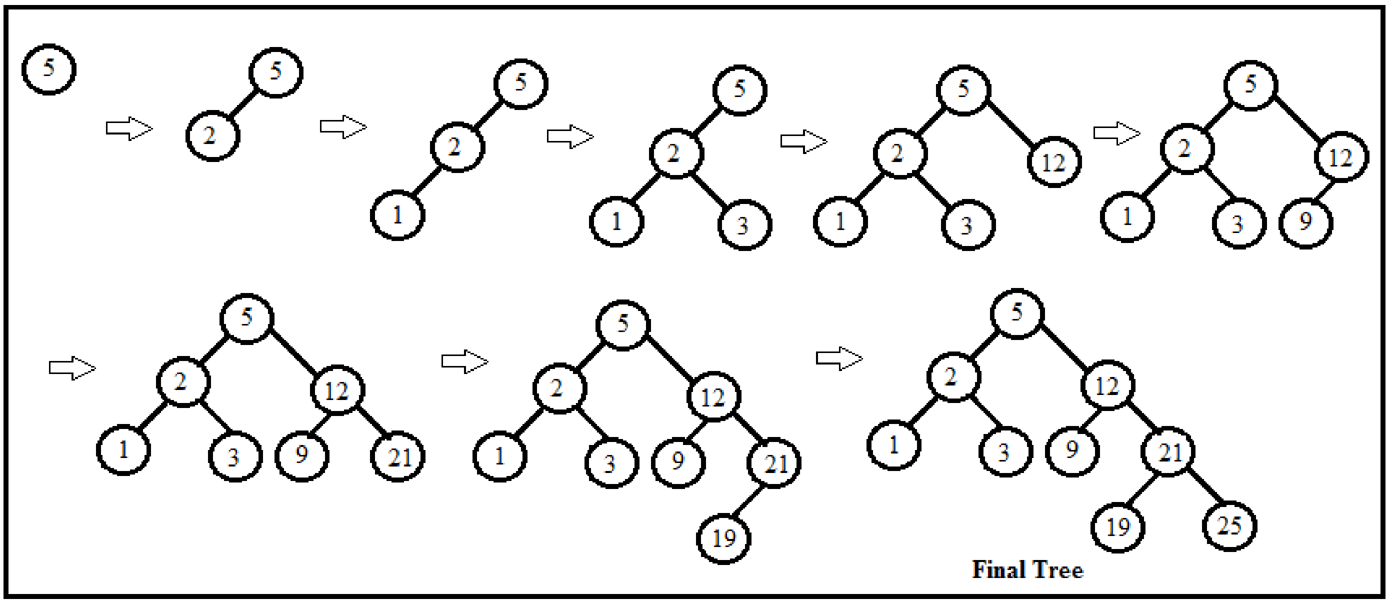
5, 2, 1, 3, 12, 9, 21, 19, 25

Now, delete the following elements, in order, from the above BST. Also give the algorithm for different deletion cases in BST: 9, 12, 21, 5

Ans:

a) TREE-INSERT(T, z)

```
1 y ← NIL
2 x ← root[T]
3 while x ≠ NIL
4       do y ← x
5             if key[z] < key[x]
6                   then x ← left[x]
7                   else x ← right[x]
8  p[z] ← y
9  if y = NIL
10      then root[T] ← z ⊹ Tree T was empty
11      else if key[z] < key[y]
12            then left[y] ← z
13            else right[y] ← z
```

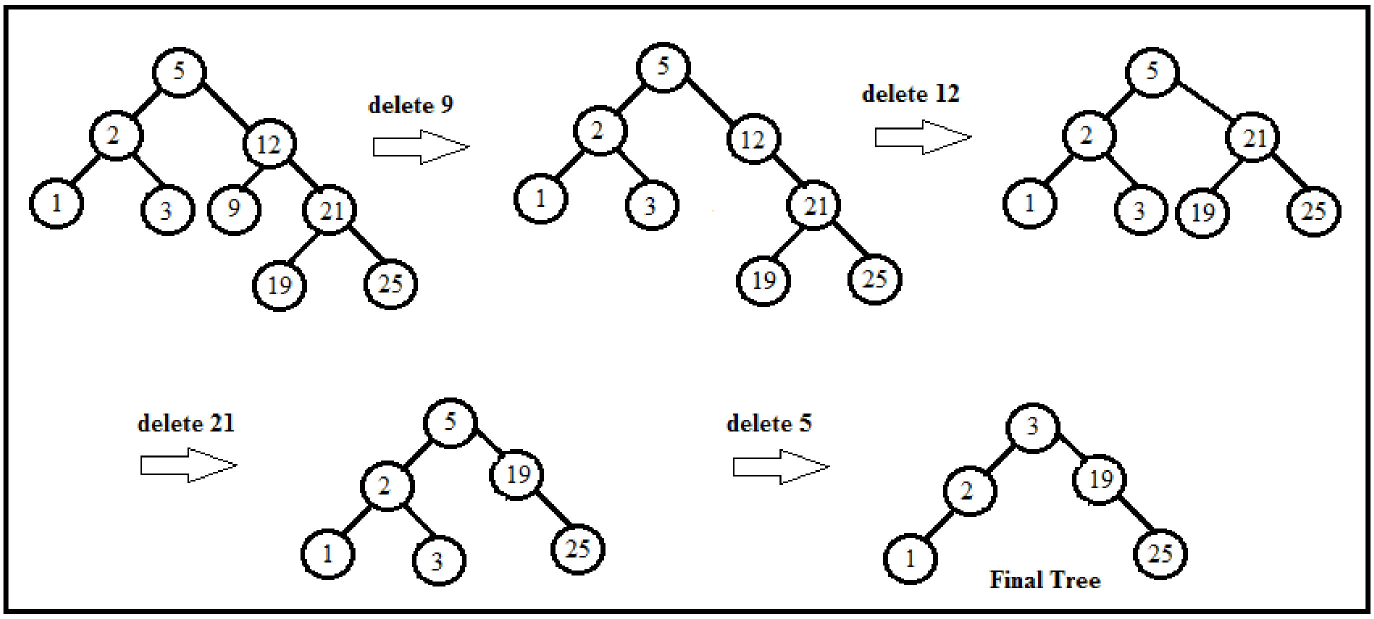


TREE-DELETE(T, z)

```
1  if left[z] = NIL or right[z] = NIL
2       then y ← z
3       else y ← TREE-SUCCESSOR(z)
4  if left[y] ≠ NIL
5       then x ← left[y]
6       else x ← right[y]
7  if x ≠ NIL
8       then p[x] ← p[y]
9  if p[y] = NIL
10      then root[T] ← x
11      else if y = left[p[y]]
12            then left[p[y]] ← x
13            else right[p[y]] ← x
14 if y ≠ z
15      then key[z] ← key[y]
16            copy y's satellite data into z
17 return y
```

```
TREE-SUCCESSOR(x)
    1 if right[x] ≠ NIL
    2     then return TREE-MINIMUM (right[x])
    3 y ← p[x]
    4 while y ≠ NIL and x = right[y]
    5     do x ← y
    6         y ← p[y]
    7 return y

TREE-MINIMUM (x)
    1 while left[x] ≠ NIL
    2     do x ← left[x]
    3 return x
```



b) **Justify the following statement with appropriate examples or algorithms:**
"Divide and conquer is not suitable where the solution of size n depends upon n sub-solutions, each of size (n-1) and Overlapping Sub Problems."

Ans: Divide & Conquer is not where the solution of size n depends upon n sub-solutions each of size (n-1) and overlapping sub-problems as it is expected in D&C that the problem will be divided into two equal sub-parts which reduces the size of the problem ideally into half which further helps to decrease the time complexity as well as helps in parallelization of the problem. But in case of sub-problem the size (n-1), the problem is not reducing instead it is just traversing all the elements just like a normal iterative approach. For example in Quick Sort, if the pivot is well selected then the pivot divides the problem into approx. two equal parts which results in O (n log n) time complexity, but otherwise, in worst case it generates the sub-problems of 1 and n-1 sizes which leads to a complexity of O(n^2). Similarly in case of overlapping sub-problems, there will be division issues and the answers will be calculated through interdependency between the solutions, which will hamper the principle of D&C.

Ques 4: Given a decimal number n, convert the number to its binary representation.

a) **Suggest the data structure that will be used to solve this problem.**
b) **Write the pseudo-code of the algorithm to solve the above problem using the suggested data structure.**

Ans:

a) Stack is the data structure that will be used to solve this.

b)

STEP 1:

```
1) if NUM > 1
2)     push NUM on stack
3)     recursively call function with 'NUM / 2'
```

STEP 2:

```
1) pop NUM from stack, divide it by 2 and print it's remainder.
```

Ques 5:

a) **Linear Search:** $T(N) = T(N-1) + 1$

Binary search: $T(N) = T(N/2) + 1$

b)

Linear Search:

$T(N) = T(N-1) + 1$

Expanding we have:

$T(N) = T(N-1) + 1$

$T(N) = T(N-2) + 1 + 1$

$T(N) = T(N-3) + 1 + 1 + 1$

:

$T(N) = T(N-k) + k$

To stop at $T(0)$, we have $N-k=0 \rightarrow N=k$

$T(n) = T(N-N) + N$

$T(n) = T(0) + N = N + c_0 = O(N)$ ($T(0)$ is some constant, c_0)

Binary search:

$T(N) = T(N/2) + 1$

Repeat the recurrence...

$T(N) = T(N/4) + 2$

$= T(N/8) + 3$

...

$= T(N/2^k) + k$

So, let:

$\rightarrow N/2^k = 1$

$\rightarrow N = 2^k$

$\rightarrow k = \log_2 N = \lg n$

Let's consider $T(1) = c_0$

$T(n) = T(1) + c \lg(n)$

$= c_0 + c \lg(n)$

$= O(\lg(n))$

Ques 6: Given a binary tree T (not a binary search tree) and two values say n1 and n2. The lowest common ancestor between two nodes n1 and n2 is defined as the lowest node in T that has both n1 and n2 as descendants (where we allow a node to be a descendant of itself).

a) **Write the algorithm to find the lowest common ancestor (LCA).**

b) **Give the complexity of your algorithm**.

Ans:

Method 1 (By Storing root to n1 and root to n2 paths):

Following is simple algorithm to find LCA of n1 and n2.

1) Find path from root to n1 and store it in a vector or array.

2) Find path from root to n2 and store it in another vector or array.

3) Traverse both paths till the values in arrays are same. Return the common element just before the mismatch.

Complexity: O(n).