DATA STRUCTURES

UNIT 5

By- Dr. Reshu Agarwal

VISION

To build strong teaching environment that responds the need of industry and challenges of society.

MISSION

M1:Developing strong mathematical & computing fundamentals among the students.

M2: Extending the role of computer science and engineering in diverse areas.

M3: Imbibing the students with a deep understanding of professional ethics and high integrity to serve the nation.

M4: Providing an environment to the students for their growth both as individuals and as globally competent Computer Science professional.

M5: Outreach activities will contribute to the overall wellbeing of society.

Syllabus

- Searching: Sequential search, Binary Search, Comparison and Analysis Internal Sorting: Insertion Sort, Selection, Bubble Sort, Quick Sort, Two Way Merge Sort, Heap Sort, Radix Sort, Practical consideration for Internal Sorting.
- Search Trees: Binary Search Trees (BST), Insertion and Deletion in BST, Complexity of Search Algorithm, AVL trees, Introduction to m-way Search Trees, B Trees & B+ Trees .
- Hashing: Hash Function, Collision Resolution Strategies
- Storage Management: Garbage Collection and Compaction.

Learning Objectives

The objectives of the following slide is to make student aware about the :

- Various techniques of sorting and searching and their effective implementation.
- Search Trees
- Hashing
- Storage Management

Linear Search

Linear search or sequential search is a method for finding a particular value in a list that consists of checking every one of its elements, one at a time and in sequence, until the desired one is found.

Linear Search Algorithm

1. Repeat For J = 1 to N

2. If (ITEM == A[J]) Then

3. Print: ITEM found at location J

4. Return [End of If]

[End of For Loop]

5. If (J > N) Then

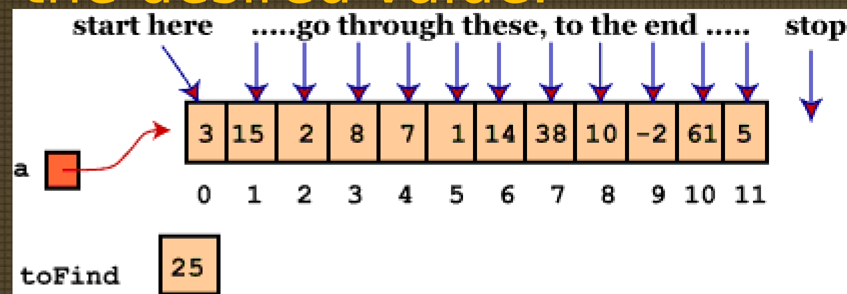
6. Print: ITEM doesn’t exist

[End of If]

7. Exit

How Linear Search works

Linear search in an array is usually programmed by stepping up an index variable until it reaches the last index. This normally requires two comparisons for each list item: one to check whether the index has reached the end of the array, and another one to check whether the item has the desired value.



Complexity of linear Search

- Linear search on a list of n elements. In the worst case, the search must visit every element once. This happens when the value being searched for is either the last element in the list, or is not in the list. However, on average, assuming the value searched for is in the list and each list element is equally likely to be the value searched for, the search visits only n/2 elements. In best case the array is already sorted i.e O

Algorithm	Worst Case	Average Case	Best Case
Linear Search	O(n)	O(n)	O(1)

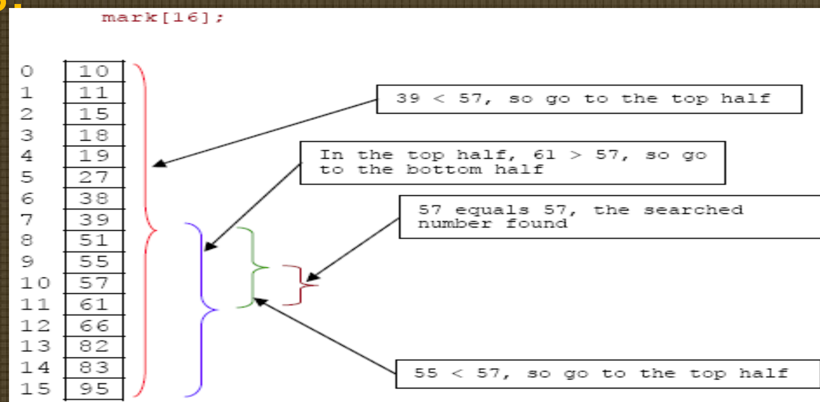
Binary Search

- A binary search or half-interval search algorithm finds the position of a specified input value (the search "key") within an array sorted by key value.

- For binary search, the array should be arranged in ascending or descending order.

How Binary Search Works

- Searching a sorted collection is a common task. A dictionary is a sorted list of word definitions.
- Given a word, one can find its definition. A telephone book is a sorted list of people's names, addresses, and telephone numbers. Knowing someone's name allows one to quickly find their telephone number and address.



Complexity of Binary Search

- A binary search halves the number of items to check with each iteration, so locating an item (or determining its absence) takes logarithmic time

Algorithm	Worst Case	Average Case	Best Case
Binary Search	O(nlogn)	O(nlogn)	O(1)

INTRODUCTION TO SORTING

- Sorting is nothing but storage of data in sorted order, it can be in ascending or descending order.

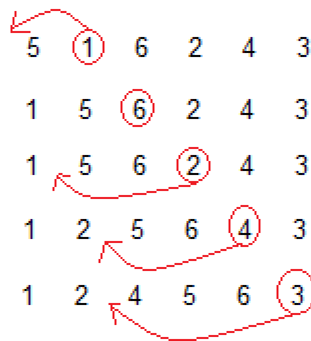
- The term Sorting comes into picture with the term Searching. There are so many things in our real life that we need to search, like a particular record in database, roll numbers in merit list, a particular telephone number, any particular page in a book etc.

Insertion sort

- It is a simple sorting algorithm that builds the final sorted array (or list) one item at a time. This algorithm is less efficient on large lists than more advanced algorithms such as quicksort, heap sort, or merge sort. However, insertion sort provides several advantages:
- Simple implementation
- Efficient for small data sets
- Stable; i.e., does not change the relative order of elements with equal keys
- In-place; i.e., only requires a constant amount O(1) of additional memory space.

How Insertion Sort Works

Lets take this Array.



As we can see here, in insertion sort, we pick up a key, and compares it with elemnts ahead of it, and puts the key in the right place

5 has nothing before it.

1 is compared to 5 and is inserted before 5.

6 is greater than 5 and 1.

2 is smaller than 6 and 5, but greater than 1, so its is inserted after 1.

And this goes on...

(Always we start with the second element as key.)

Complexity of Insertion Sort

- The number f(n) of comparisons in the insertion sort algorithm can be easily computed. First of all, the worst case occurs when the array A is in reverse order and the inner loop must use the maximum number K-1 of comparisons. Hence

  F(n)= 1+2+3+.....................................+(n-1)=n(n-1)/2= O(n2)

- Furthermore, One can show that, on the average, there will be approximately (K-1)/2 comparisons in the inner loop. Accordingly, for the average case. F(n)=O(n2)
- Thus the insertion sort algorithm is a very slow a

Algorithm	Worst Case	Average Case	Best Case
Insertion Sort	$n(n-1)/2 = O(n^2)$	$n(n-1)/4 = O(n^2)$	$O(n)$

Selection Sort

- Selection sorting is conceptually the simplest sorting algorithm.
- This algorithm first finds the smallest element in the array and exchanges it with the element in the first position, then find the second smallest element and exchange it with the element in the second position, and continues in this way until the entire array is sorted

How Selection Sort works

Original Array	After 1st pass	After 2nd pass	After 3rd pass	After 4th pass	After 5th pass
3	1	1	1	1	1
6	6	3	3	3	3
1	3	6	4	4	4
8	8	8	8	5	5
4	4	4	6	6	6
5	5	5	5	8	8

Complexity of Selection Sort Algorithm

- The number of comparison in the selection sort algorithm is independent of the original order of the element. That is there are n-1 comparison during PASS 1 to find the smallest element, there are n-2 comparisons during PASS 2 to find the second smallest element, and so on. Accordingly

F(n)=(n-1)+(n-2)+…………………………+2+1=n(n-1)/2 = O(n2)

Algorithm	Worst Case	Average Case	Best Case
Selection Sort	$n(n-1)/2 = O(n^2)$	$n(n-1)/2 = O(n^2)$	$O(n^2)$

Bubble Sort

Let us take the array of numbers "5 1 4 2 8", and sort the array from lowest number to greatest number using bubble sort. In each step, elements written in bold are being compared. Three passes will be required.

First Pass:

(5 1 4 2 8) (1 5 4 2 8), Here, algorithm compares the first two elements, and swaps since 5 > 1.

(1 5 4 2 8) (1 4 5 2 8), Swap since 5 > 4

(1 4 5 2 8) (1 4 2 5 8), Swap since 5 > 2

(1 4 2 5 8) (1 4 2 5 8),

Now, since these elements are already in order (8 > 5), algorithm does not swap them.

Second Pass:

(1 4 2 5 8) (1 4 2 5 8)

(1 4 2 5 8) (1 2 4 5 8), Swap since 4 > 2

(1 2 4 5 8) (1 2 4 5 8)

(1 2 4 5 8) (1 2 4 5 8)

Now, the array is already sorted, but our algorithm does not know if it is completed. The algorithm needs one whole pass without any swap to know it is sorted.

Third Pass:

(1 2 4 5 8) (1 2 4 5 8)

(1 2 4 5 8) (1 2 4 5 8)

(1 2 4 5 8) (1 2 4 5 8)

(1 2 4 5 8) (1 2 4 5 8)

Complexity of Bubble Sort Algorithm

- In Bubble Sort, n-1 comparisons will be done in 1st pass, n-2 in 2nd pass, n-3 in 3rd pass and so on. So the total number of comparisons will be

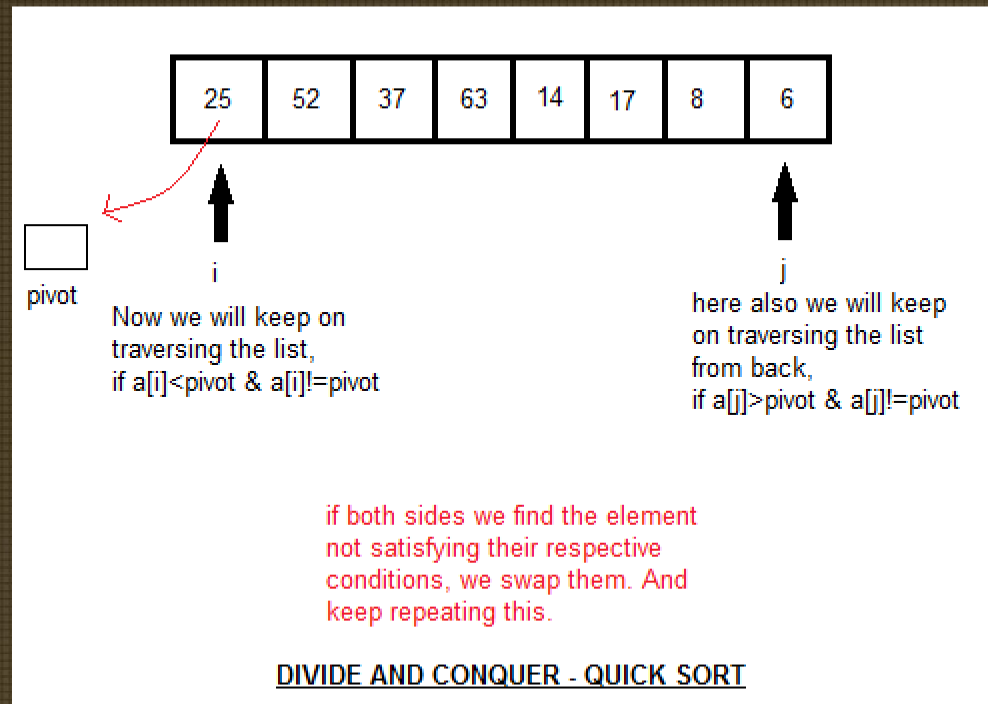
F(n)=(n- 1)+(n-2)+............................... +2+1=n(n-1)/2 = O(n2)

Algorithm	Worst Case	Average Case	Best Case
Bubble Sort	$n(n-1)/2 = O(n^2)$	$n(n-1)/2 = O(n^2)$	O(n)

Quick Sort

- Quick Sort, as the name suggests, sorts any list very quickly. Quick sort is not stable search, but it is very fast and requires very less additional space. It is based on the rule of Divide and Conquer (also called partition-exchange sort). This algorithm divides the list into three main parts
- Elements less than the Pivot element
- Pivot element
- Elements greater than the pivot element

How Quick Sort Works



Complexity of Quick Sort Algorithm

- The Worst Case occurs when the list is sorted. Then the first element will require n comparisons to recognize that it remains in the first position. Furthermore, the first sublist will be empty, but the second sublist will have n-1 elements. Accordingly the second element require n-1 comparisons to recognize that it remains in the second position and so on.

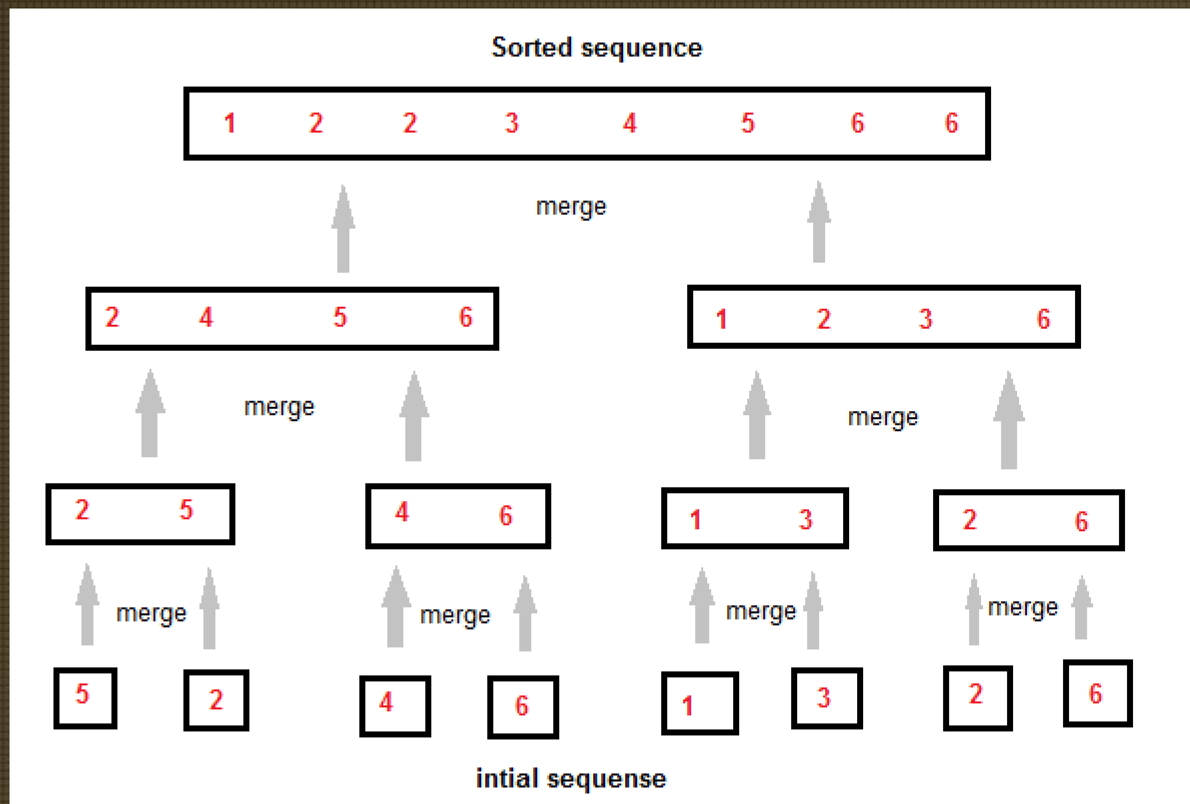
F(n)= n+(n-1)+(n-2)+.............................+2+1=n(n+1)/2 = O(n2)

Algorithm	Worst Case	Average Case	Best Case
Quick Sort	$n(n+1)/2 = O(n^2)$	O(n logn)	O(n logn)

Merge Sort

- Merge Sort follows the rule of Divide and Conquer. But it doesn't divide the list into two halves.
- In merge sort the unsorted list is divided into N sub lists, each having one element, because a list of one element is considered sorted. Then, it repeatedly merge these sub lists, to produce new sorted sub lists, and at lasts one sorted list is produced.
- Merge Sort is quite fast, and has a time complexity of O(n log n). It is also a stable sort, which means the equal elements are ordered in the same order in the sorted list.

How Merge Sort Works



Complexity of Merge Sort Algorithm

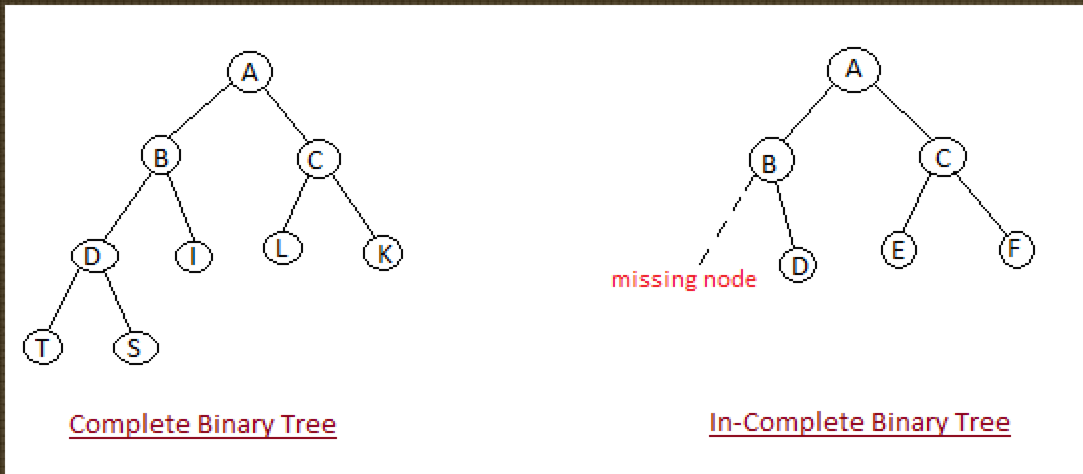
- Let f(n) denote the number of comparisons needed to sort an n-element array A using merge-sort algorithm. The algorithm requires at most log n passes. Each pass merges a total of n elements and each pass require at most n comparisons. Thus for both the worst and average case
- $F(n) \leq n \log n$
- Thus the time complexity of Merge Sort is O(n Log n) in all 3 cases (worst, average and best) as merge sort always divides the array in two halves and take linear

Algorithm	Worst Case	Average Case	Best Case
Merge Sort	O(n logn)	O(n logn)	O(n logn)

Heap Sort

- Heap Sort is one of the best sorting methods being in-place and with no quadratic worst-case scenarios. Heap sort algorithm is divided into two basic parts:
- Creating a Heap of the unsorted list.
- Then a sorted array is created by repeatedly removing the largest/smallest element from the heap, and inserting it into the array. The heap is reconstructed after each removal.

How Heap Sort Works



Complexity of Heap Sort Algorithm

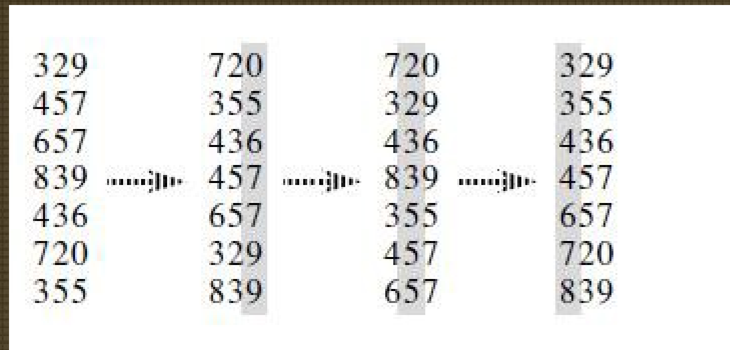
- The algorithm has two phases, and we analyze the complexity of each phase separately.
- Phase 1. Since H is complete tree, its depth is bounded by log2m where m is the number of elements in H. Accordingly, the total number g(n) of comparisons to insert the n elements of A into H is bounded as g(n) ≤ n log2n
- Phase 2. Reheaping uses 4 comparisons to move the node L one step down the tree H. Since the depth cannot exceeds log2m , it uses 4log2m comparisons to find the appropriate place of L in the tree H. h(n)≤4nlog2n
- Thus each phase requires time proportional to nlog2n, the running tim

Algorithm	Worst Case	Average Case	Best Case
Heap Sort	O(n logn)	O(n logn)	O(n logn)

Radix Sort

- The idea is to consider the key one character at a time and to divide the entries, not into two sub lists, but into as many sub lists as there are possibilities for the given character from the key. If our keys, for example, are words or other alphabetic strings, then we divide the list into 26 sub lists at each stage. That is, we set up a table of 26 lists and distribute the entries into the lists according to one of the characters in the key.

How Radix Sort Works



Complexity of Radix Sort

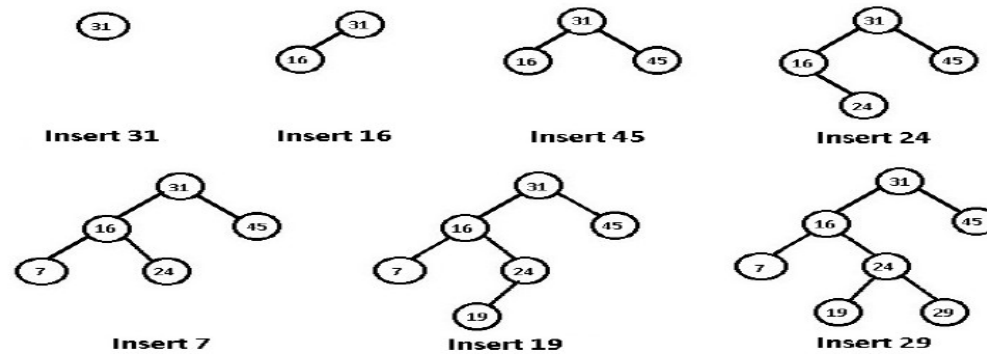
- The list A of n elements A1, A2,……………An is given. Let d denote the radix(e.g d=10 for decimal digits, d=26 for letters and d=2 for bits) and each item Ai is represented by means of s of the digits:
- Ai = di1 di2………………. dis
- The radix sort require s passes, the number of digits in each item . Pass K will compare each digit with each of the d digits. Hence C(n)≤ d*s*n

Algorithm	Worst Case	Average Case	Best Case
Radix Sort	$O(n^2)$	d*s*n	O(n logn)

Binary Search Tree

- A binary search tree (BST), sometimes also called an ordered or sorted binary tree, is a node based binary tree data structure where each node has a comparable key and satisfies the restriction that the key in any node is larger than the keys in all nodes in that node's left sub tree and smaller than the keys in all nodes in that node's right sub-tree. The properties of binary search trees are as follows:
- The left sub tree of a node contains only nodes with keys less than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- The left and right subtree each must also be a binary search tree.
- Each node can have up to two successor nodes.
- There must be no duplicate nodes.
- A unique path exists from the root to every other node.

Insertion in BST



AVL Trees

An AVL tree is a binary search tree which has the following properties:

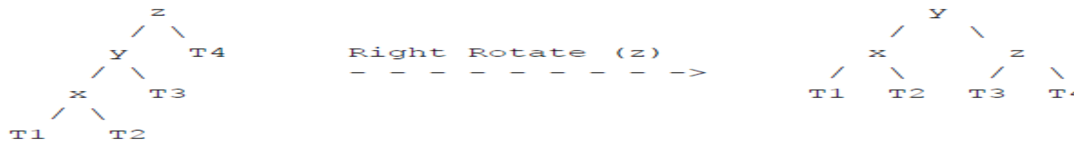
1. The sub-trees of every node differ in height by at most one.

2. Every sub-tree is an AVL tree.

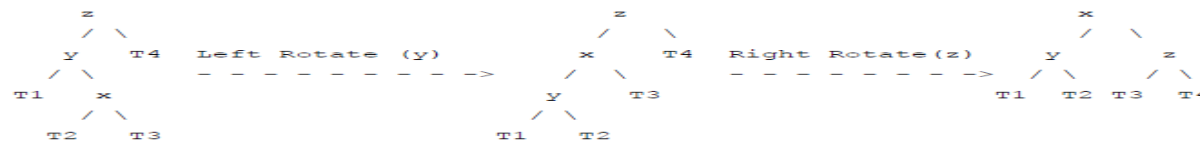
Insertion in AVL Tree

a) Left Left Case

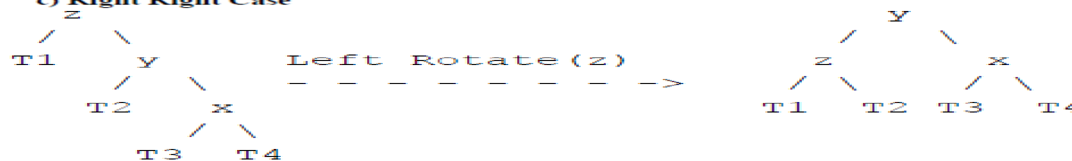
T1, T2, T3 and T4 are subtrees.



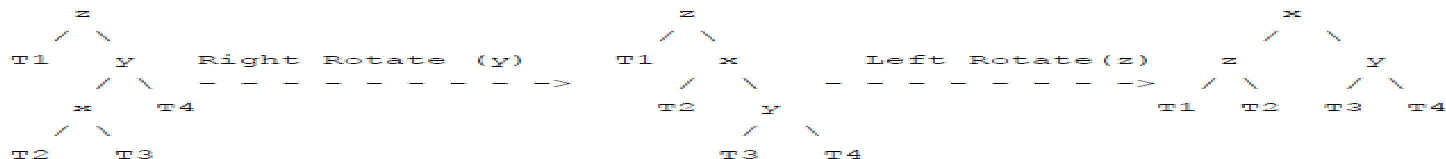
b) Left Right Case



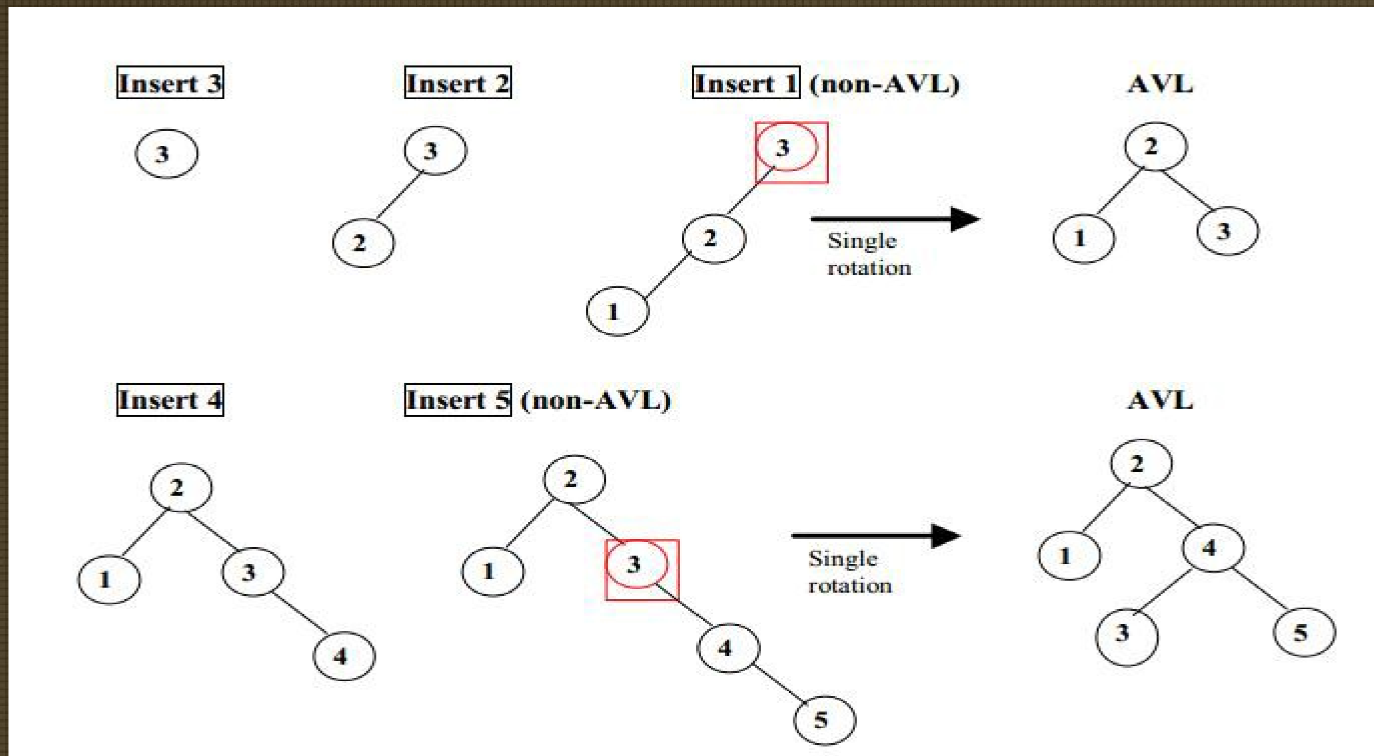
c) Right Right Case



d) Right Left Case



Insertion in AVL Tree



Deletion in AVL Tree

Let w be the node to be deleted

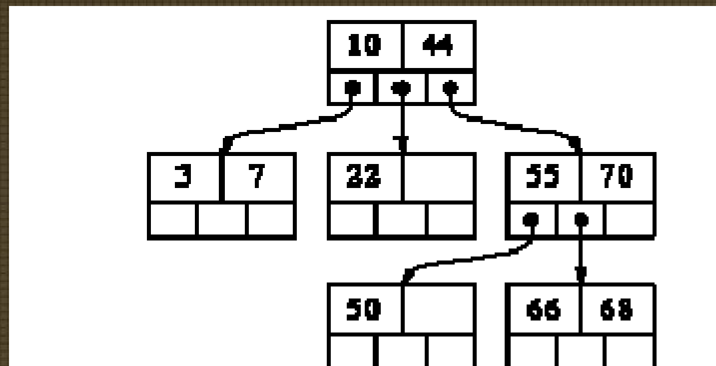
1) Perform standard BST delete for w.

2) Starting from w, travel up and find the first unbalanced node. Let z be the first unbalanced node, y be the larger height child of z, and x be the larger height child of y. Note that the definitions of x and y are different from insertion here.

3) Re-balance the tree by performing appropriate rotations on the subtree rooted with z as explained above.

M-WAY Search Trees

- A binary search tree has *one* value in each node and *two* subtrees. This notion easily generalizes to an M-way search tree, which has (M-1) values per node and M subtrees. M is called the *degree* of the tree. A binary search tree, therefore, has degree 2. For example, here is a 3-way search tree:



B-trees

A B-tree is an M-way search tree with two special properties:

1. It is perfectly balanced: every leaf node is at the same depth.
2. Every node, except perhaps the root, is at least half-full, i.e. contains M/2 or more values

(of course, it cannot contain more than M-1 values). The root may have any number of values (1 to M-1).

Insertion into a B-Tree

To insert value X into a B-tree, there are 3 steps:

1. Using the SEARCH procedure for M-way trees (described above) find the leaf node to which X should be added.

2. Add X to this node in the appropriate place among the values already there. Being a leaf node there are no subtrees to worry about.

3. If there are M-1 or fewer values in the node after adding X, then we are finished.

B+ Tree

- A B+ tree is an n-ary tree with a variable but often large number of children per node. A B+ tree consists of a root, internal nodes and leaves.
- The root may be either a leaf or a node with two or more children. A B+ tree can be viewed as a B-tree in which each node contains only keys (not key-value pairs), and to which an additional level is added at the bottom with linked leaves.

HASHING

Hash Function

A hash function is a function that:

1. When applied to an Object, returns a number
2. When applied to equal Objects, returns the same number for each
3. When applied to unequal Objects, is very unlikely to return the same number for each.

Collision Resolution Techniques

When two values hash to the same array location, this is called a collision. There are two broad ways of collision resolution:

1. Separate Chaining:: An array of linked list implementation.

2. Open Addressing: Array-based implementation

   (i) Linear probing (linear search)

   (ii) Quadratic probing (nonlinear search)

   (iii) Double hashing (uses two hash functions)

STORAGE MANAGEMENT

- **Garbage collection** (GC) is a form of automatic memory management. The garbage collector attempts to reclaim garbage, or memory occupied by objects that are no longer in use by the program.
- Garbage collection is the opposite of manual memory management, which requires the programmer to specify which objects to de-allocate and return to the memory system.
- Like other memory management techniques, garbage collection may take a significant proportion of total processing time in a program and can thus have significant influence on performance.

Compaction

- The process of moving all marked nodes to one end of memory and all available memory to other end is called compaction. Algorithm which performs compaction is called compacting algorithm.
- After repeated allocation and de allocation of blocks, the memory becomes fragmented.
- Compaction is a technique that joins the non contiguous free memory blocks to form one large block so that the total free memory becomes contiguous.

ASSIGNMENT QUESTIONS

Q1. What are the factors to be considered during the selection process of sorting technique?

Q2. Sort the following list in ascending order using bubble sorting and write an algorithm for the same.

56,92,38,44,90,61,73,23,2

Q3. Sort the list using quick sorting and explain with the help of algorithm.

10,40,8,53,2,13,25,14

Q4. Differentiate between 1. Internal and external search 2. Primary key and secondary key.

Q 5. Write a c program to find a desired element in an array using sequential searching technique.

ASSIGNMENT QUESTIONS

Q6. How do we resolve the collision?

Q7. What are the factors to be considered during the selection process of sorting technique?

Q8. Create a B Tree of order 5 on the following data:

10, 15, 19, 3, 5, 50, 79, 47, 33, 20

Q9. Differentiate between linear search and binary search.

Q10. What is the prerequisite for the binary search?

Q11. What is called as hashing?

Q12. Draw the 11 item hash table resulting from hashing the keys: 12, 44, 13, 88, 23, 94, 11, 39, 20, 16 and 5 using the hash function h(i) = (2i+5) mod 11.

Q13. Sort the following list using Heap Sort technique, displaying each step. 20, 12, 25 6, 10, 15, 13.

Q14. When will you sort an array of pointers to list elements, rather than sorting the elements themselves?

TUTORIAL QUESTIONS

Q1. The element being searched for is not found in an array of 100 elements. What is the average number of comparisons needed in a sequential search to determine that the element is not there, if the elements are completely unordered?

Q2. Show the result of inserting the keys.

F, S, Q, K, C, L, H, T, V, W, M, R, N , P, A, B, X, Y, D, Z, E in the order to an empty B-tree of degree-3.

Q3.What do you mean by hash clash? Explain in detail any one method to resolve hash collisions.

Q4. Define Hashing. How do collisions happen during hashing? Explain the different techniques resolving of collision.

Q5. Write an algorithm for quick sort. Trace your algorithm on the

Following data to sort the list:

2,15,4,21,56,7,85,51,8,1,59,42,10,9.

Outcomes

After reading above topics students will be able to:

- Understand various searching and sorting techniques.

References

1)Data Structures and Algorithms Made Easy by *Narasimha Karumanchi*

2)Lipschutz, “Data Structures” Schaum’s Outline Series, Tata McGraw-hill Education (India) Pvt. Ltd.

3) Thareja, “Data Structure Using C” Oxford Higher Education.

4) AK Sharma, “Data Structure Using C”, Pearson Education India.

5)Rajesh K. Shukla, “Data Structure Using C and C++” Wiley Dreamtech Publication.