DATA STRUCTURES

2 - MARKS

1. Define Data Structure.
Data structures are specialized formats or arrangements used to store and organize data in a computer's memory or storage system. They enable efficient access, modification, and retrieval of data.

Data structure = Organised data + Allowed Operation.

Or

A data structure is logical way of storing and organizing data either in computer's memory or on the disk storage so that it can be used efficiently.

2. Define the classification of data structures.
There are two types of data structures:

- Linear Data Structure
- Non-linear data structure

3. Define linear data structure?
Linear data structures are data structures having a linear relationship between its adjacent elements.

4. Define Non-Linear data structure?
Non-Linear data structures are data structures having hierarchical relationship between its adjacent elements.

5. State the main advantage of binary search over linear search.
The main advantage of binary search over linear search is its efficiency in terms of time complexity. Binary search divides the element and works on sorted arrays or lists. The key advantage is that it significantly reduces the number of comparisons needed to find a target element compared to linear search.

6. Define ADT and give an example (or) Write short notes on ADT.
An Abstract Data Type (ADT) is a set of elements with a collection of well-defined operations. The operations can take as operands not only instances of the ADT but other types of operands or instances of other ADTs.
Examples of ADTs include list, stack, queue, set, tree, graph, etc.

9. What do asymptotic notation means?
Asymptotic notations are terminology that is introduced to enable us to make meaningful statements about the time and space complexity of an algorithm. The different notations are

Big – Oh notation (O)
Omega notation(Ω)
Theta notation(Θ).

10. Differentiate stack and queue ADT.

#	STACK	QUEUE
1	Objects are inserted and removed at the same end.	Objects are inserted and removed from different ends.
2	In stacks only one pointer is used. It points to the top of the stack.	In queues, two different pointers are used for front and rear ends.
3	In stacks, the last inserted object is first to come out.	In queues, the object inserted first is first deleted.
4	Stacks follow Last In First Out (LIFO) order.	Queues following First In First Out (FIFO) order.
5	Stack operations are called push and pop.	Queue operations are called enqueue and dequeue.
6	Stacks are visualized as vertical collections.	Queues are visualized as horizontal collections.

11. Convert the given infix to postfix and prefix?

A ^ B / C - (E * F) + G

Postfix : A B ^ C / E F * - G +

Prefix : + - / ^ A B C * E F G

12.What is linked list? Give its types.

In a linked list, each element is called a node. Every node in the list points to the next node in the list. Therefore, in a linked list every node contains two types of information:

- The data stored in the node
- A pointer or link to the next node in the list

The various types of linked lists are:

- singly linked lists
- circularly linked lists
- doubly-linked lists

13. Write the advantages of circular linked list.

(i) Any node can be a starting point.

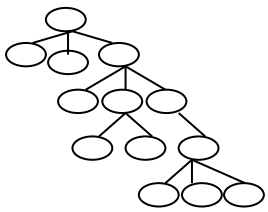
(ii) We can traverse the whole list by starting from any point. We just need to stop when the first visited node is visited again.

(iii) It includes improved efficiency, enhanced traversal flexibility, optimized memory utilization and suitability for scenarios involving cyclic operations or relationships.

14. Define tree.

Tree is defined as a finite set of one or more nodes such that

- there is one specially designated node called ROOT and
- the remaining nodes are partitioned into a collection of sub-trees of the root each of which is also a tree.



15. What is binary search tree?

Binary search tree is a special type of binary tree that follows the following condition:

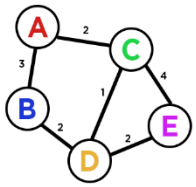
(i) Left child is smaller than its parent node

(ii) Right child is greater than its parent node

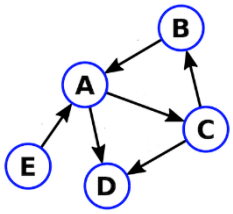
16. Among the sorting techniques which sorting is considered as the best.

Among, the sorting techniques, "Heap Sort" is often considered as one of the best. It has a consistent O(n log n) time complexity, providing reliable performance for large datasets. Unlike bubble and insertion sorts with quadratic complexities, Heap Sort is more scalable. Radix Sort is efficient for fixed-length integer keys.

17. What is weighted and directed graph?



A graph is said to be a weighted graph if every edge in the graph is assigned some weight or value. The weight of an edge is a positive value that may be representing the distance between the vertices or the weights of the edges along the path.



A graph is said to be a directed graph if an edge between any two nodes in a graph are directionally oriented, it is also referred as digraph.

5 - MARKS

1. Explain the purpose of asymptotic notation in the analysis of algorithms.

Asymptotic notation plays a crucial role in the analysis of algorithms by providing a concise and standardized way to describe the growth rate of algorithmic functions as input size approaches infinity. The primary purposes of asymptotic notation in algorithm analysis are as follows:

Efficiency Comparison:

Asymptotic notation allows for a high-level comparison of the efficiency of algorithms by focusing on their growth rates rather than specific constants or lower-order terms. This abstraction helps identify algorithms that are more scalable for larger input sizes.

Algorithmic Complexity Analysis:

It provides a framework for expressing the time complexity and space complexity of algorithms. Time complexity, often denoted using big O notation, describes how the running time of an algorithm increases with the input size, while space complexity characterizes the space requirements.

Simplicity and Clarity:

Asymptotic notation simplifies the analysis of algorithms by capturing the essential characteristics of their performance. It provides a concise way to express the upper and lower bounds of algorithmic behavior, making it easier to communicate and understand.

Commonly used asymptotic notations include Big O (upper bound), Omega (lower bound), and Theta (tight bound), each providing different insights into the behavior of algorithms. Asymptotic analysis is a powerful tool in algorithmic design and is instrumental in understanding and comparing the efficiency of algorithms in various contexts.

2. Explain the stack operations using an array with a relevant example.

The operations on stack are

1. Push (Insertion)
2. Pop (Deletion)
3, Peek (Top most Element)
4. Overflow (Full)
5. Underflow (Empty)

Stack – Abstract Data Type

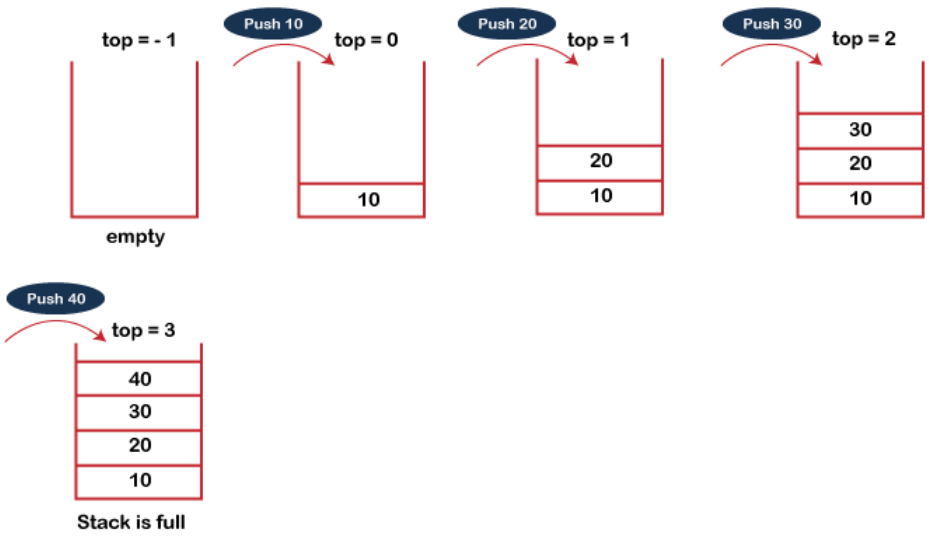
A stack 'S' is an abstract data type (ADT) supporting the following three methods:

push(n) : Inserts the item n at the top of stack
pop(n) : Removes the top element from the stack and returns that top element. An error occurs if the stack is empty.
peek(n) : Returns the top element and an error occurs if the stack is empty.

PUSH OPERATION

Push operation inserts an element onto the stack.
Let's represented the stack by means of array.
An attempt to push an element onto the stack, when the array is full, causes an overflow.

Push operation involves :
- Check whether the array is full before attempting to push another element. If so, halt the execution.
- Otherwise increment the top pointer.
- Push the element onto the top of the stack.



Overflow

The steps involved in the PUSH operation :

- Before inserting an element in a stack, check whether the stack is full.
- If we try to insert the element in a stack, and the stack is full, then the overflow condition occurs.
- When initialize the stack, set the value of top as -1 to check that the stack is empty.
- When the new element is pushed in a stack.
- The value of the top gets incremented, i.e., top = top + 1, and the element will be placed at the new position of the top.
- The elements will be inserted until we reach the max size of the stack.

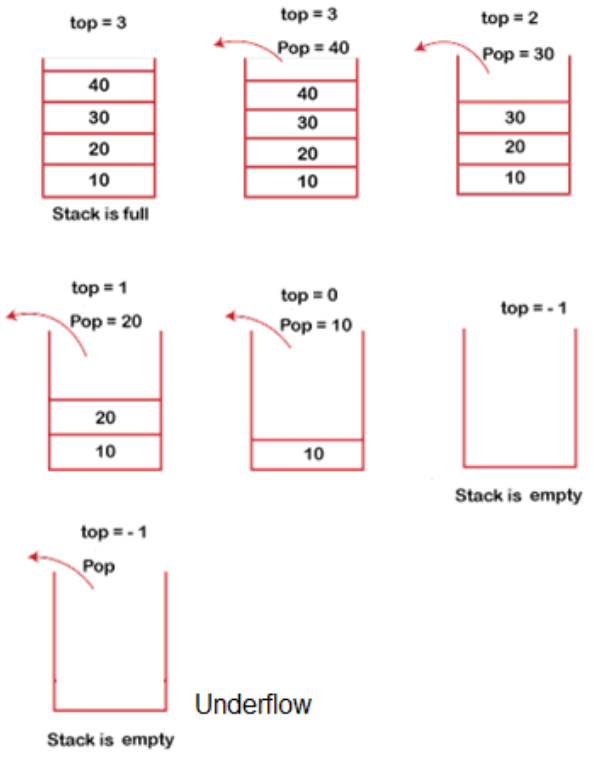
POP OPERATION

POP operation removes an element from the stack.
An attempt to pop an element from the stack, when the array is empty, causes an underflow.

Pop operation involves :
- Check whether the array is empty before attempting to pop another element. If so, halt execution.
- Decrement the top pointer.
- Pop the element from the top of the stack.

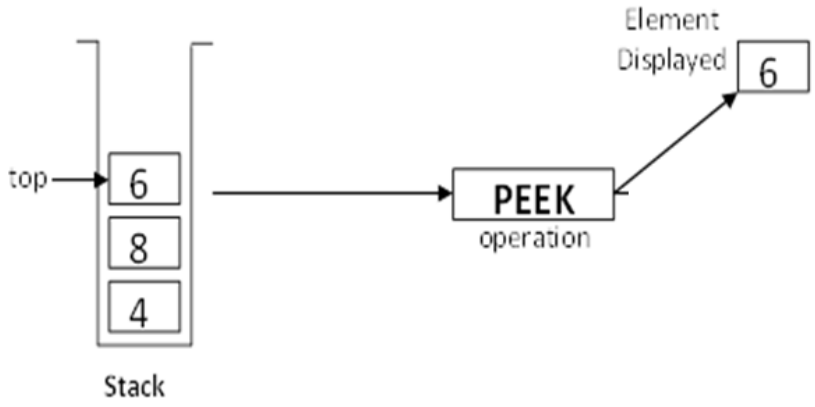
The steps involved in the POP operation :

- Before deleting the element from the stack, we check whether the stack is empty.
- If we try to delete the element from the empty stack, then the underflow condition occurs.
- If the stack is not empty, we first access the element which is pointed by the top
- Once the pop operation is performed, the top is decremented by 1, i.e., top=top-1.



Peek Operation:

- Returns the item at the top of the stack but does not delete it.
- This can also result in underflow if the stack is empty.



3. Discuss the insertion and deletion operations in singly linked list.

Insertion : This operation is used to add an element to the linked list.

Insertion of a node of a linked list can be on three positions

- Insertion at the beginning
- Insertion at the end and
- Insertion in the middle of the list.

Deletion : Deletion operations are used to remove an element from the beginning of the linked list.

Deletion of a node in the linked list can be done in three ways

- Deletion at the beginning
- Deletion at the end and
- Deleting at a specific position.

INSERTION IN A LIST:

Inserting a new item, say 'x' into the list has three situations:

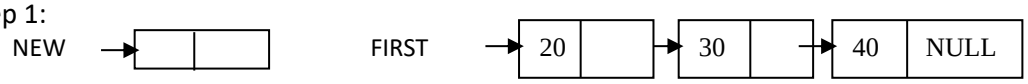
1. Insertion at the beginning or front of the list
2. Insertion at the end of the list
3. Insertion in the middle of the list

(i) INSERTION AT THE BEGINNING :

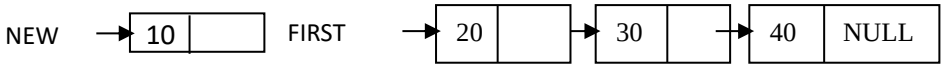
Steps for placing the new item at the beginning of a linked list:

1. Obtain space for new node
2. Assign data to the item field of new node
3. Set the next field of the new node to point to the start of the list
4. Change the head pointer to point to the new node.

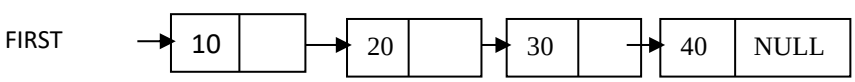
Step 1:



Step 2:



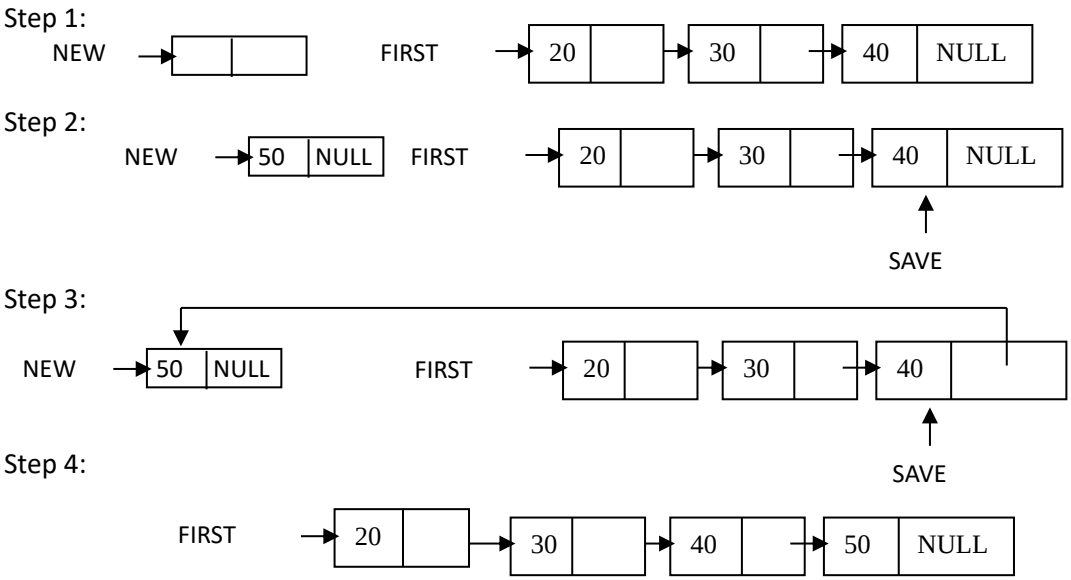
Step 3:



(ii) INSERTION AT THE END :

Steps for inserting an item at the end of the list:

1.Set space for new node x
2.Assign value to the item field of x
3.Set the next field of x to NULL
4.Set the next field of N2 to point to x

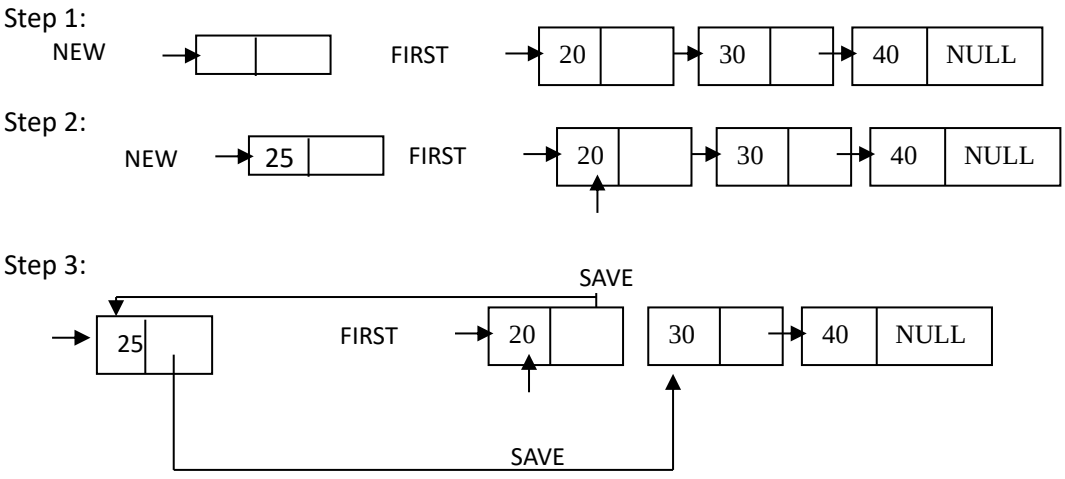


Note that no data is physically moved, as we had seen in array implementation. Only the pointers are readjusted.

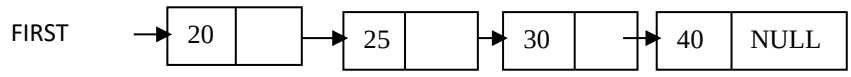
(iii) INSERTION AT THE MIDDLE :

Steps for inserting the new node x between the two existing nodes, say N1 and N2(or in order):

1. Set space for new node x
2. Assign value to the item field of x
3. Search for predecessor node n1 of x
4.Set the next field of x to point to link of node n1(node N2)
5.Set the next field of N1 to point to x.



Step 4:



DELETING AN ITEM FROM A LIST:

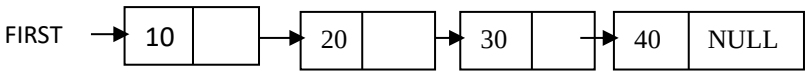
Deleting a node from the list requires only one pointer value to be changed, there we have situations:

1.Deleting the first item
2.Deleting the last item
3.Deleting between two nodes in the middle of the list.

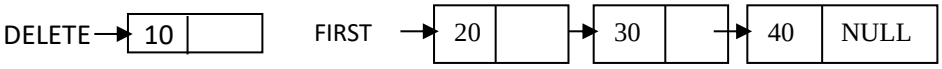
(i) DELETION AT THE BEGINNING :

Steps for deleting the first item:
1.If the element x to be deleted is at first store next field of x in some other variable y.
2.Free the space occupied by x
3.Change the head pointer to point to the address in y.

Step 1: Delete 10



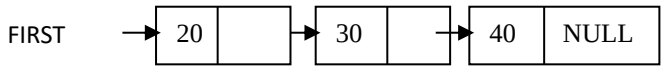
Step 2:



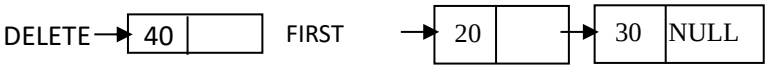
(ii) DELETION AT THE END :

Steps for deleting the last item:
1.Set the next field of the node previous to the node x which is to be deleted as NULL
2.Free the space occupied by x

Step 1: DELETE 40



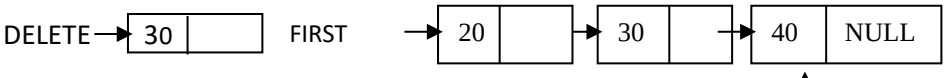
Step 2:



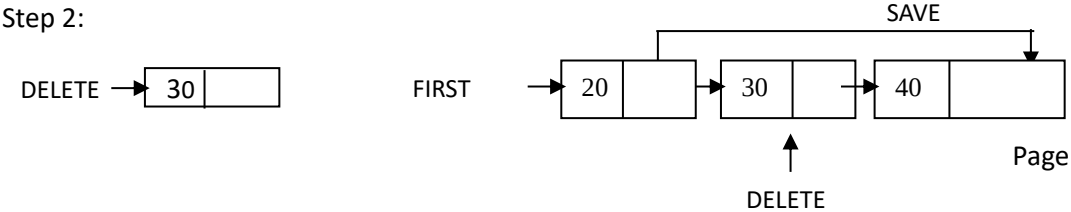
(iii) DELETION AT THE MIDDLE :

Steps for deleting x between two nodes N1 and N2 in the middle of the list:
1.Set the next field of the node N1 previous to x to point to the successor field N2 of the node x.
2.Free the space occupied by x.

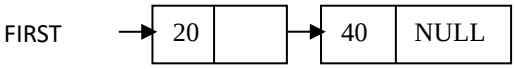
Step 1:



Step 2:

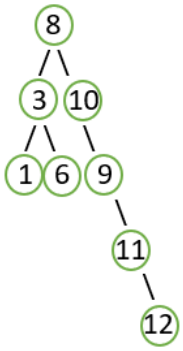


Step 3:



4. Construct a binary search tree for the following list of elements.
8, 3, 10, 1, 6, 9, 11 and 12.
Then give the relevant steps to insert an element 5 and delete an element 12 in the constructed binary search tree.

To construct a binary search tree (BST) with the given values (8, 3, 10, 1, 6, 9, 11, 12), we follow the rules of a binary search tree:



Start with the first element as the root.
Insert each subsequent element by comparing it with the current node:
If the element is smaller, go to the left subtree.
If the element is larger, go to the right subtree.
Repeat until an empty spot is found, then insert the element there.

5. Sort the list of elements using Radix sort.
351, 220, 6, 58, 0, 175, 267, 311, 512, 9, 24, 578

Radix sort is a non-comparative sorting algorithm that works by distributing elements into buckets based on their individual digits. The process is repeated for each digit position from ones place, tens place, hundredth place and so on until all elements are sorted.

Let's sort the given elements using radix sort:

Original List:
351, 220, 6, 58, 0, 175, 267, 311, 512, 9, 24, 578

First Pass (Sort by Units place):

Bucketing:

Create 10 buckets (0 to 9).
Distribute the elements into buckets based on their units place (rightmost digit).
Elements in each bucket maintain their original order.

Bucket 0 for units place number 0 : 220, 0
Bucket 1 for units place number 1 : 351, 311
Bucket 2 for units place number 2 : 512
Bucket 3 for units place number 3 : 24
Bucket 4 for units place number 4 :
Bucket 5 for units place number 5 : 175
Bucket 6 for units place number 6 : 6
Bucket 7 for units place number 7 : 267
Bucket 8 for units place number 8 : 58, 578
Bucket 9 for units place number 9 : 9

Now, concatenate the elements from each bucket in the order of buckets (0 to 9).
220, 0, 351, 311, 512, 24, 175, 6, 267, 58, 578, 9

Second Pass (Sort by Tens place):

List of elements after the first Pass
220, 0, 351, 311, 512, 24, 175, 6, 267, 58, 578, 9

Bucketing:

Create 10 buckets (0 to 9).
Distribute the elements after the first pass into buckets based on their tens place (second number from rightmost digit).
Elements in each bucket maintain their original order.

Bucket 0 for Tens place number 0 : 0, 6, 9
Bucket 1 for Tens place number 1 : 311, 512
Bucket 2 for Tens place number 2 : 220, 24
Bucket 3 for Tens place number 3 :
Bucket 4 for Tens place number 4 :
Bucket 5 for Tens place number 5 : 351, 58
Bucket 6 for Tens place number 6 : 267
Bucket 7 for Tens place number 7 : 175, 578
Bucket 8 for Tens place number 8 :
Bucket 9 for Tens place number 9 :

Now, concatenate the elements from each bucket in the order of buckets (0 to 9).
0, 6, 9, 311, 512, 220, 24, 351, 58, 267, 175, 578

Third Pass (Sort by Hundreds place):

List of elements after the second Pass
0, 6, 9, 311, 512, 220, 24, 351, 58, 267, 175, 578

Create 10 buckets (0 to 9).
Distribute the elements after the first pass into buckets based on their tens place (second number from rightmost digit).
Elements in each bucket maintain their original order.

Bucket 0 for Hundred place number 0 : 0, 6, 9, 24,58
Bucket 1 for Hundred place number 1 : 175
Bucket 2 for Hundred place number 2 : 220, 267
Bucket 3 for Hundred place number 3 : 311, 351
Bucket 4 for Hundred place number 4 :
Bucket 5 for Hundred place number 5 : 512, 578
Bucket 6 for Hundred place number 6 :
Bucket 7 for Hundred place number 7 :
Bucket 8 for Hundred place number 8 :
Bucket 9 for Hundred place number 9 :

Final Sorted List:
0, 6, 9, 24, 58, 175, 220, 267, 311, 351, 512, 578

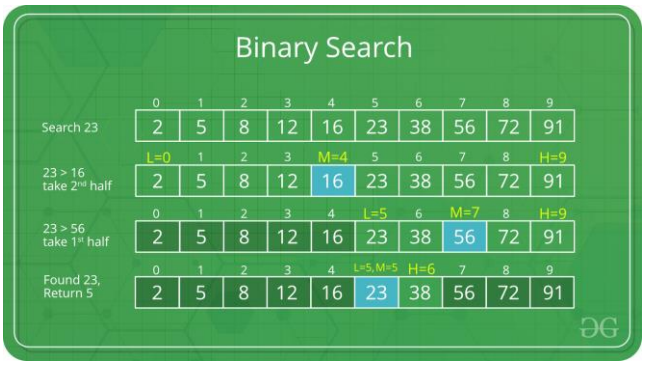
The elements are now sorted using radix sort.
The time complexity of radix sort is O(k * n), where k is the number of digits in the maximum number and n is the number of elements.
It is a linear time sorting algorithm, but it requires additional space for the bucketing process.

10 - MARKS

1. Describe binary search with an example and give its algorithm.

BINARY SEARCH

1. The binary search algorithm is one of the most efficient searching techniques, which requires the list to be sorted in ascending order.
2. To search for an element in the list, the binary search algorithm splits the list and locates the middle element of the list. It is then compared with the search element.
3. If the search element is less than the middle element, the first part of the list is searched else the second part of the list is searched.
4. The algorithm again reduces the list into two halves locate the middle element and compares with the search element. If the search element is less than the middle element the first part of the list is searched.
5. This process continues until the search element is equal to the middle element or the list consists of only one element that is not equal to the search element.



Binary Search Algorithm:

```
procedure  BINSRCH(F, n, i, k)
//Search an ordered sequential file F with records R1, ...,Rn and the keys
K1 <= K2 <= ... <= Kn for a record Ri such that Ki = K; i = 0 if there is no such record else Ki = K.
Throughout the algorithm, l is the smallest index such that Kl may be K and u the largest
index such that Ku may be K//
  l <-- 1; u <-- n
  while l <= u do
  m <-- [(l + u)/2] //compute index of middle record//
  case
  :K > Km: l <-- m + 1 //look in upper half//
  :K = Km: i <-- m; return
  :K < Km: u <-- m - 1 //look in lower half//
  end
  end i <-- 0 //no record with key K//
end BINSRCH
```

Time required by a binary search to search a list on n records is O(logn)

Best case for binary search happen if the search key is found in the middle array O(1).

Worse case for binary search happens if the search key is not in the list or the searching size equal to 1. The searching time for worst case is O(log2n).