Savitribai Phule Pune University

Section-I

S. Y. B. C. A. (Science)

SEMESTER III BCA – 234

Lab Course – I

Data Structure

Name:____

Roll No:_______________Seat No_________________

Academic Year:20_____-20___

From the Chairman's Desk

It gives me a great pleasure to present this workbook prepared by the Board of studies in Computer Applications.

The workbook has been prepared with the objectives of bringing uniformity in implementation of lab assignments across all affiliated colleges, act as a ready reference for both fast and slow learners and facilitate continuous assessment using clearly defined rubrics.

The workbook provides, for each of the assignments, the aims, pre-requisites, related theoretical concepts with suitable examples wherever necessary, guidelines for the faculty/lab administrator, instructions for the students to perform assignments and a set of exercises divided into three sets.

I am thankful to the Chairman of this course and the entire team of editors. I am also thankful to the reviewers and members of BOS, Mr. Rahul Patil and Mr. Arun Gangarde. I thank all members of BOS and everyone who have contributed directly or indirectly for the preparation of the workbook.

Constructive criticism is welcome and to be communicated to the Chairman of the Course and overall coordinator Mr. Rahul Patil. Affiliated colleges are requested to collect feedbacks from the students for the further improvements.

I am thankful to Hon. Vice Chancellor of Savitribai Phule Pune University Prof. Dr. Nitin Karmalkar and the Dean of Faculty of Science and Technology Prof. Dr. M G Chaskar for their support and guidance.

Prof. Dr. S S Sane
Chairman, BOS in Computer Applications
SPPU, Pune

Editors:		
Assign. No.	**Teacher Name**	**College**
1	Prof. Deepali Jagdale	New Arts, Commerce and science college, Ahmednagar
2	Prof. Kajal Davange	New Arts, Commerce and science college, Ahmednagar
3	Prof. Mayuri Dasri	Abeda Inamdar Senior College, Pune
4	Prof. Smita J. Ghorpade	K.T.H.M college,,Nashik
5	Prof. Kamil Khan	Abeda Inamdar Senior College, Pune
6	Prof. Kamil Khan	Abeda Inamdar Senior College, Pune
7	Prof. Mayuri Dasri & Smita J. Ghorpade	Abeda Inamdar Senior College, Pune & K.T.H.M college,,Nashik
8	Prof. Satish Mulgi	Dr. D. Y. Patil College ,Pune

Compiled By

Mr. Kamil Khan Ajmal Khan (Chairman, Data Structure laboratory)

Abeda Inamdar Senior College, Pune

Reviewed By:

1. **Prof. Arun Gangarde**
   New Arts, Commerce and science college, Ahmednagar
   BOS,BCA(Science)

2. **Prof. Rahul Patil**
   K.T.H.M college,,Nashik
   BOS,BCA(Science)

Introduction

1. About the Workbook:

This workbook is intended to be used by SYBCA (Science) students for the Data StructureAssignmentsinSemester–III.This workbook isdesignedbyconsidering all the practical concepts / topics mentioned in syllabus.

2. The objectives of this Workbook are:

1) Defining the scope of the course.

2) To bring the uniformity in the practical conduction and implementation in all colleges affiliated to SPPU.

3) To have continuous assessment of the course and students.

4) Providing ready reference for the students during practical implementation.

5) Provide more options to students so that they can have good practice before facing the examination.

6) Catering to the demand of slow and fast learners and accordingly providing the practice assignments to them.

3. How to use this Workbook:

The workbook is divided into two sections. Section-I is related to DS assignments

The Section-I (DS) is divided into eight assignments. Each DS assignment has several SET. It is mandatory for students to complete all the SET in given slot.

4. Instructions to the students:

Please read the following instructions carefully and follow them.

- Students are expected to carry this workbook everytime they come to the lab for practical.
- Students should prepare for the assignment by reading the relevant material which is mentioned in ready reference.
- Instructor will specify which problems to solve in the lab during the allotted slot and student should complete them and get verified by the instructor. However, student should spend additional hours in Lab and at home to cover all workbook assignments if needed.
- Students will be assessed for each assignment on a scale from 0 to5

Not done	0
Incomplete	1
Late Complete	2
Needs improvement	3
Complete	4
Well Done	5

5. Guidelines for Instructors:

- Make sure that students should follow above instructions.
- Explain the assignment and related concepts using white board if required or by demonstrating the software.
- Give specific input to fill the blanks in queries which can vary from student to student.
- Evaluate each assignment carried out by a student on a scale of 5 as specified above by ticking appropriate box.
- The value should also be entered on assignment completion page of the respective Lab course.

6. Guidelines for Lab administrator:

You have to ensure appropriate hardware and software is made available to each student.

The operating system and software requirements on server side and also client side areas given below:

- Server and Client Side-(Operating System) Fedora Core Linux/Windows
- TurboC
- DevC

0

Table of Contents

Assignment Completion Sheet

Assign. No.	Topics for the Assignments	Marks (5)	Teacher Signature
1	Non-Recursive Sorting Techniques		
2	Recursive Sorting Techniques		
3	Searching Techniques		
4	Linkedlist		
5	Stack		
6	Queue		
7	Binary Search Tree (Dynamic)		
8	Graph		
	Total Marks(Out of 40)		
	Total Marks(Out of 15)		

This is to certify that Mr. /Ms.______________________________________has successfully completed the course work for BCA-234: Data Structures Laboratory and has scored________________Marks out of 15.

Instructor **H.O.D /Coordinator**

Internal Examiner **External Examiner**

INTRODUCTION

SORTING

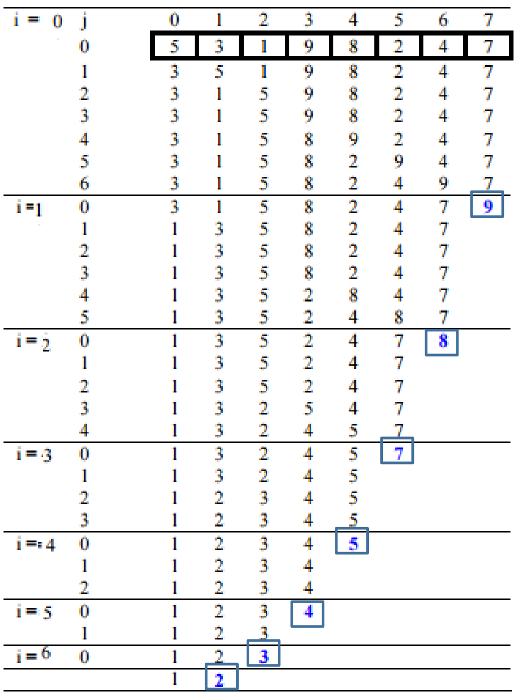
- Sorting is a process of ordering a list of elements in either ascending or descending order.
- List is a collection of record each contains one or more fields. The field which contains unique value for each record is called key field.
- Definition: - Sorting is the operation of arranging the records of a table according to the key value of each record e.g. consider a telephone directory which consists of 4 field phone number, name, address, pincode. Soalargedataismaintainedintheformofrecords.Ifwewanttosearchaphonenoandnameitshould be alphabetically sorted then we can search easily. It would be very difficult if records were unsorted.

BUBBLE SORT

- This is one of the simplest and most popular sorting methods. The basic idea is to pass through the file sequentially several times.
- In each pass we compare successive pairs of elements(x[i] with x[i+1]) and interchange the two if they are not in the required order.
- One element is placed in its correct position in each pass.
- In first pass, the largest element will sink to the bottom, second largest in the second pass and so on. Thus a total of n-1 passes are required to sort n keys
- Efficiency of bubble sort

  a) Worst and Average Case Time Complexity: O(n*n). Worst case occurs when array is reverse sorted.
  b) Best Case Time Complexity: O(n). Best case occurs when array is already sorted.
  c) Auxiliary Space:O(1)
  d) Boundary Cases: Bubble sort takes minimum time (Order of n) when elements are already sorted.
  e) Sorting In Place:Yes
  f) Stable:Yes

Example : Array Elements :5,3,1,9,8,2,4,7



i = 0	j	0	1	2	3	4	5	6	7
	0	5	3	1	9	8	2	4	7
	1	3	5	1	9	8	2	4	7
	2	3	1	5	9	8	2	4	7
	3	3	1	5	9	8	2	4	7
	4	3	1	5	8	9	2	4	7
	5	3	1	5	8	2	9	4	7
	6	3	1	5	8	2	4	9	7
i =1	0	3	1	5	8	2	4	7	$\boxed{9}$
	1	1	3	5	8	2	4	7	
	2	1	3	5	8	2	4	7	
	3	1	3	5	8	2	4	7	
	4	1	3	5	2	8	4	7	
	5	1	3	5	2	4	8	7	
i = 2	0	1	3	5	2	4	7	$\boxed{8}$	
	1	1	3	5	2	4	7		
	2	1	3	5	2	4	7		
	3	1	3	2	5	4	7		
	4	1	3	2	4	5	7		
i = 3	0	1	3	2	4	5	$\boxed{7}$		
	1	1	3	2	4	5			
	2	1	2	3	4	5			
	3	1	2	3	4	5			
i = 4	0	1	2	3	4	$\boxed{5}$			
	1	1	2	3	4				
	2	1	2	3	4				
i = 5	0	1	2	3	$\boxed{4}$				
	1	1	2	3					
i = 6	0	1	2	$\boxed{3}$					
		1	$\boxed{2}$						

Illustration of Array Elements->

INSERTION SORT

- Insertion sort inserts each item into its proper place in the final list
- In this the first iteration starts with comparison of 1st element with0th
- In second iteration 2nd element is compared with the 0th and 1st element and soon.
- In every iteration an element is compared with all elements
- The basic idea of this method is to place an unsorted element into its correct position in a growing sorted list of data. We select one element from the unsorted data at a time and insert it into its correct position in the sorted set.
- E.g. in order to arrange playing cards we pick one card at a time and insert this card hold in the hand.

Algorithm

To sort an array of size n in ascending order:

1: Iterate from arr[1] to arr[n] over the array.

2: Compare the current element (key) to it spredecessor.

3: If the key element is smaller than its predecessor, compare it to the elements before. Move the greater elements one position up to make space for the swapped element.

a) Time Complexity: O(n*2)
b) Auxiliary Space:O(1)
c) Boundary Cases: Insertion sort takes maximum time to sort if elements are sorted in reverse order. And it takes minimum time (Order of n) when elements are already sorted.
d) Algorithmic Paradigm: Incremental Approach
e) Sorting In Place:Yes
f) Stable: Yes

Example: Array Elements: 5,1,6,2,4,3

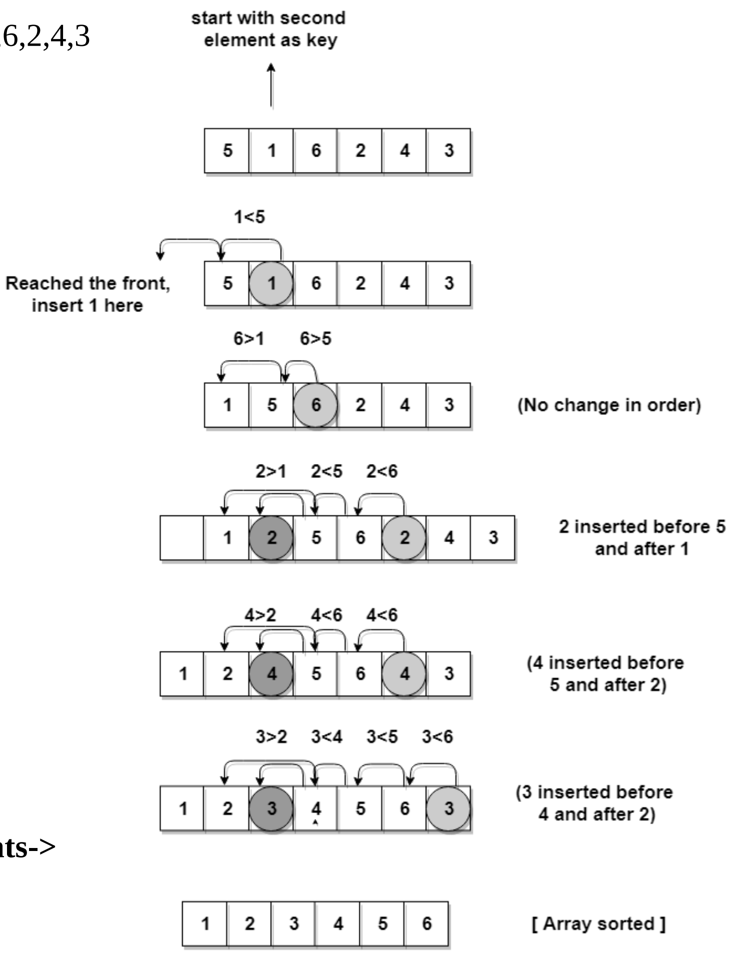

start with second element as key
1<5
Reached the front, insert 1 here
6>1 6>5
(No change in order)
2>1 2<5 2<6
2 inserted before 5 and after 1
4>2 4<6 4<6
(4 inserted before 5 and after 2)
3>2 3<4 3<5 3<6
(3 inserted before 4 and after 2)
[Array sorted]

Illustration of Array Elements->

SET A:

1) Write a C program to accept and sort n elements in ascending order by using bubble sort.

2) Write a C program to accept and sort n elements in ascending order by using insertion sort.

SET B:

1) Write a C program to read the data from the file “employee.txt” which contains empno and empname and sort the data on names alphabetically (use strcmp) using Bubble Sort.

2) Write a C program to read the data from the file “person.txt” which contains personno and personage and sort the data on age in ascending order using insertion Sort.

SET C:

1) Write a C program to sort a random array of n integers (value of n accepted from user) by using Bubble Sort algorithm in ascending order

2) Write a C program to sort a random array of n integers (value of n is accepted from user) by using Insertion Sort algorithm in ascending order.

4) Write a C program to sort the elements by initializing the array (e.g int A[5] = {10, 20, 35, 23, 12}) using bubble sort.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

INTRODUCTION

1. Quick Sort

Quick Sort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot.

Procedure –

- We will consider one element at a time (pivot) and place it in its correct position.
- The pivot is placed in a position such that all elements to the left of the pivot are less than the pivot and all elements to the right are greater.
- The array is partitioned into two parts:- left partition and right partition.
- The same method is applied for each of thepartition.
- The process continues till no more partition can be made. We shall be considering the first element of the partition as the pivot element.

Algorithm:

```
Step 1: start.
Step 2: A is an array of n element.
Step 3:lb=0                lb = lower bound
       ub=n-1              ub = upper bound.
Step 4: if(lb<ub)
          i.e. if the array can be partitioned
          j=partition(A,lb,ub)    // j is the pivot position

          quicksort(A,lb,j-1);
          quicksort(A,j+1,ub);
```

- Now, we must write the function to partition the array. There are many methods to do the partitioning depending upon which element is chosen as the pivot.
- We will be selecting the first element as the pivot element and do the partitioning accordingly.
- We shall choose the first element of the sub- array as the pivot and find its correct position in the sub-array.
- We will be using two variables down and up for moving down and up array.

Algorithm for partitioning

```
Step 1: down=lb+1
Step 2: up=ub
Step 3:pivot=A[lb]
Step 4: perform step 5 to 7 as long as down<up else go to step 8.
Step 5:  while (A[down]<=pivot&&down<ub)              down++;
Step 6:while(A[up]>pivot)                             up--;
Step 7: if (down<up) interchange A[down] and A[up]
Step 8:  interchange A[up]and pivot,          j=up,   i.e. pivot position=up
Step 9: return up
Step 10: stop.
```

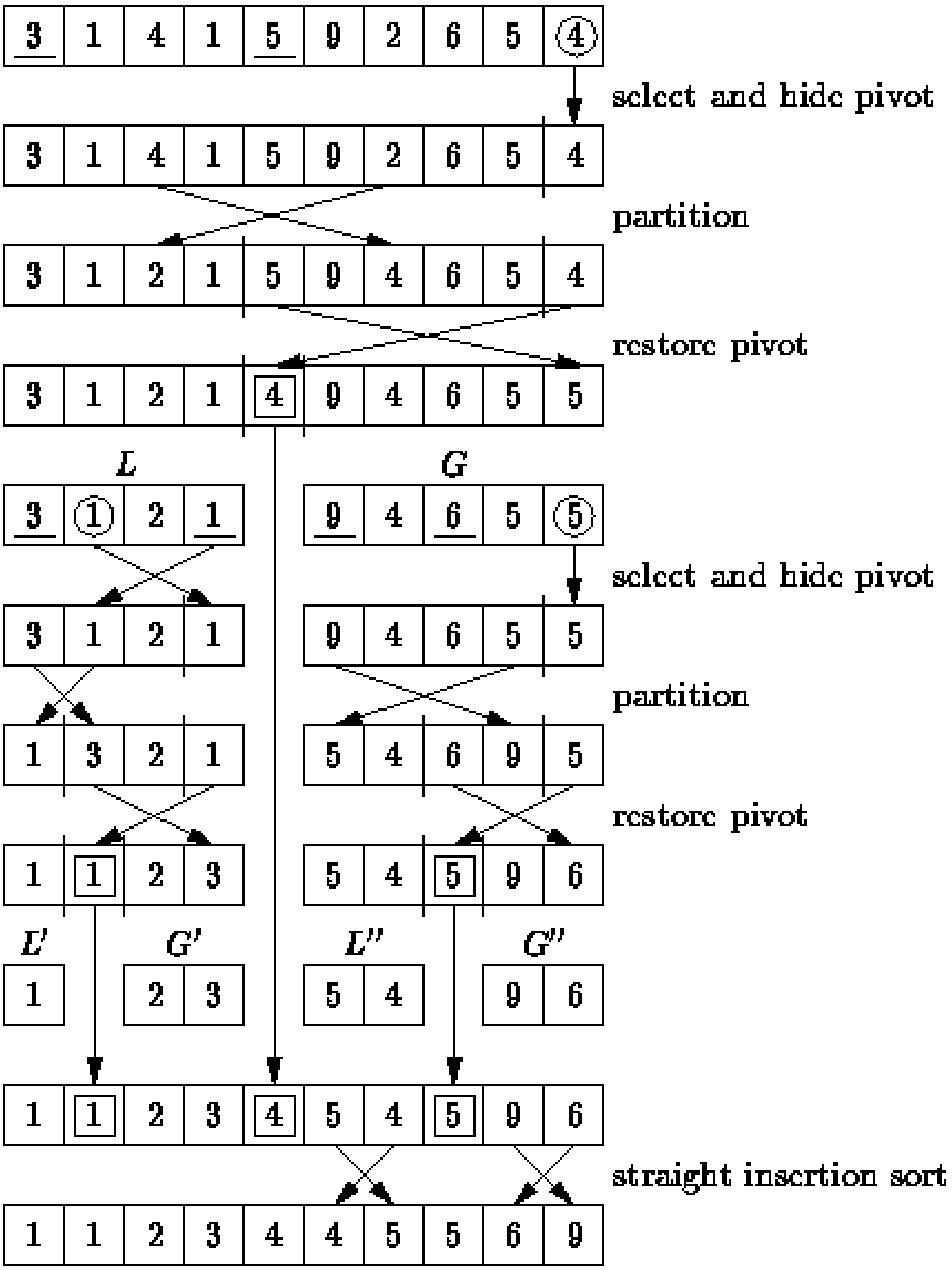
- In this algorithm we want to find the position of pivot i.e.A[lb].
- We use two pointers up and down initialized to the first and last elements respectively.
- We repeatedly increase down as long as the element is <pivot.
- We repeatedly decrease up as long as the element is >pivot.

- If up and down cross each other i.e. up<=down, the correct position of the pivot is up and A[up and pivot are inter changed.
- If up and down do not cross A[up] and A[down] are interchanged and process is repeated till they do not cross or coincide.
- Efficiency of quicksort.

 Best case = average case = O(nlogn)

 Worst case = **$O(n^2)$**

Example: Array Elements:3,1,4,1,5,9,2,6,5,4



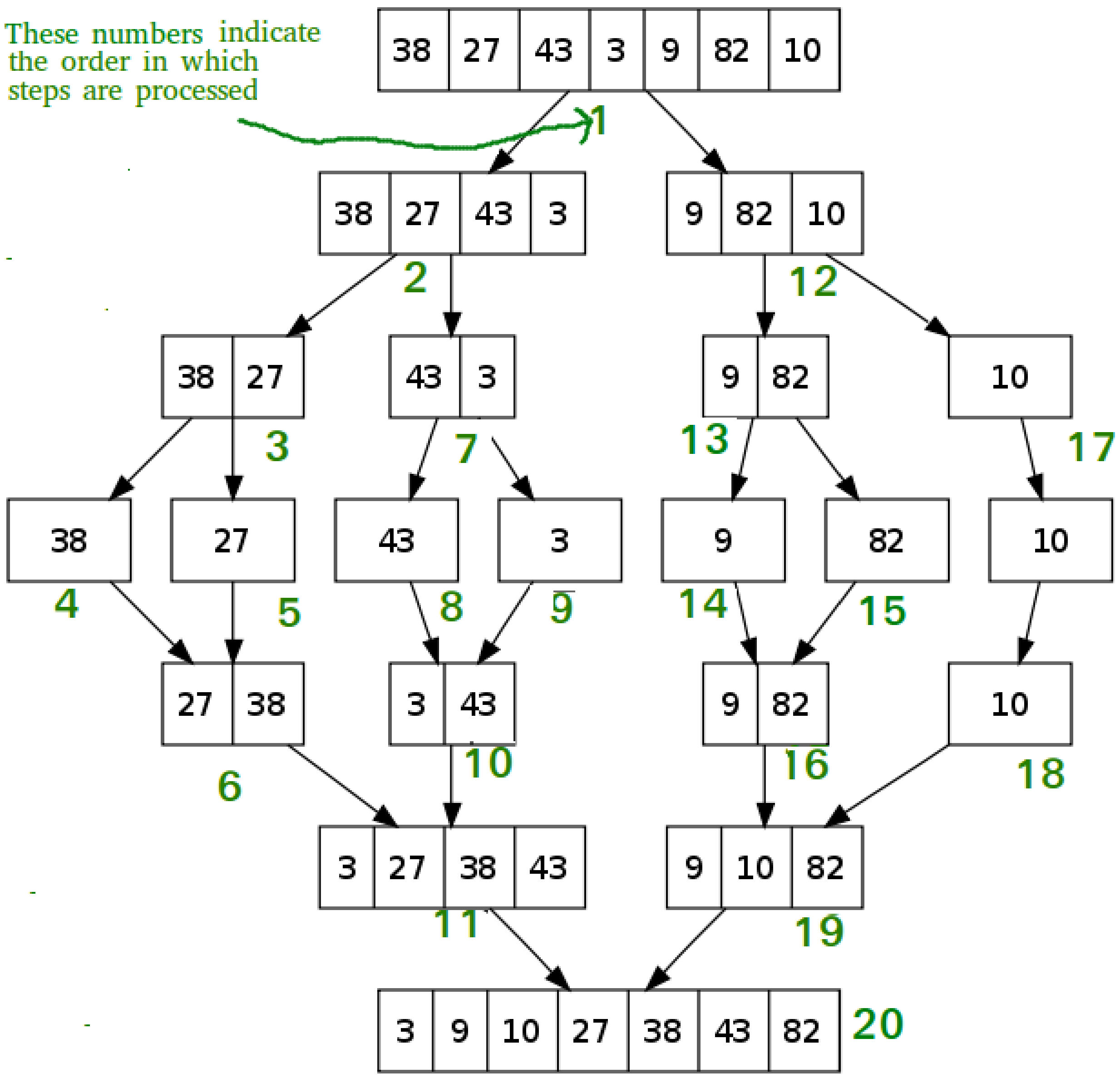
2. Merge Sort

- Merging is the process of combining two or more sorted data lists into a third list such that it is also sorted.
- Merge sort follows Divide and Conquer strategy.
  - Divide :- divide an n element sequence into n/2subsequence.
  - Conquer :- sort the two sequences recursively.
  - Combine :- merge the two sorted sequence into a single sequence.
- In this two list are compared and the smallest element is stored in the third array.

Algorithm:-

Step 1: start
Step 2: initially the data is considered as a single array of n element.
Step 3: divide the array into n/2 sub-array each of length 2^i (I is 0 for 0th iteration). i.e. array is divided into n sub-arrays each of 1 element.
Step 4: merge two consecutive pairs of sub-arrays such that the resulting sub-array is also sorted.
Step 5: The sub-array having no pairs is carried a sit is
Step 6: step 3 and 4 are repeated till there is only one sub-array remaining of size n.
Step 7: stop.

Example: Array Elements: 38, 27, 43,3,9,82,10



SET A:

1) Write a program in C to accept 5 numbers from the user and sort the numbers in ascending order by using Quicksort.
2) Write a C program to sort a random array of n integers by using Quick Sort algorithm in ascending order.
3) Write a C program to accept and sort n elements in ascending order by using Quicksort.

SET B:

1) Write a program in C to accept 5 numbers from the user and sort the numbers in ascending order by using Merge sort.
2) Write a C program to sort a random array of n integers by using Merge Sort algorithm in ascending order.

SET C:

1) Write a C program to sort the elements by initializing the array (e.g. int A[5] = {11, 12,15, 16, 17}) using Merge sort.
2) Write a C program to sort the elements by initializing the array (e.g. int A[5] = {11, 12,15, 16, 17}) using Quick Sort.
3) Add the code in SETA(Q1) and SETB(Q1) to Print Time complexity for Quick sort and Merge sort.

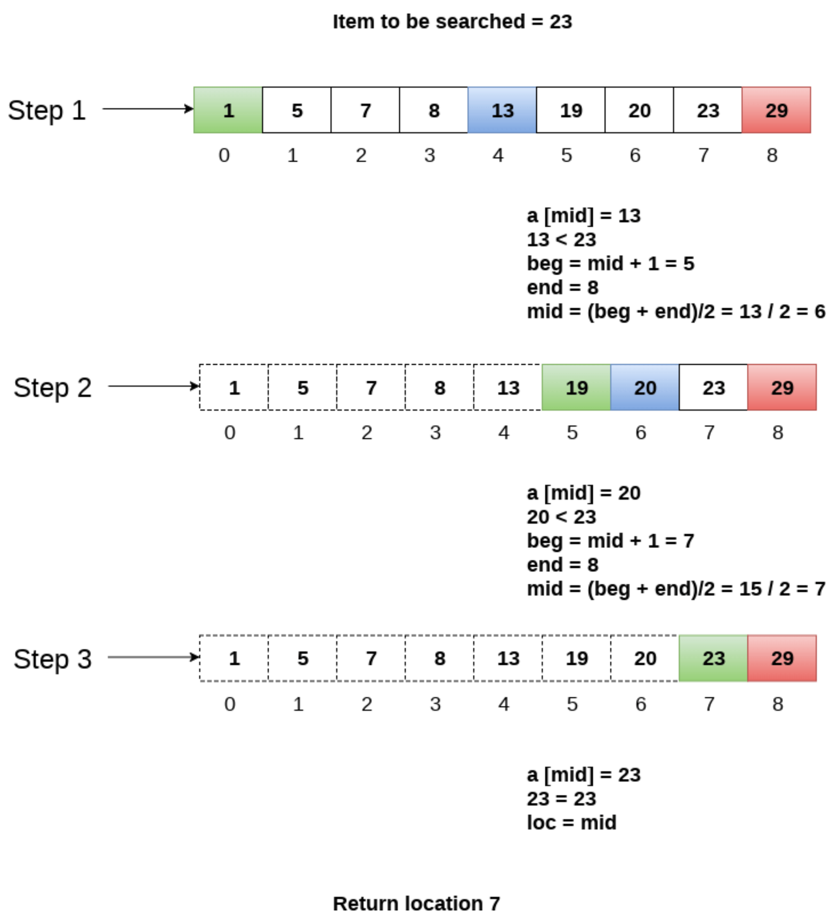
Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

INTRODUCTION

1. **Linear search**
   Linearsearchisaverysimplesearchalgorithm.Inthistypeofsearch,asequentialsearchismadeover allitemsonebyone.Everyitemischeckedandifamatchisfoundthenthatparticularitemisreturned, otherwise the search continues till the end of the data collection. Time complexity of linear search is O(n).
   Example 1:
   Input: arr[] = {10,130,170,20,80,110,30, 60, 50,100}
   Search element x = 110;
   Output: The element is present at 6th index position.
   Example 2:
   Input: arr[] = {10, 20, 80, 30, 60, 50,110, 100, 130, 170}
   Search element x = 175;
   Output: Element x is not present in arr[]

2. **BinarySearch:**Search a sorted array by repeatedly dividing the search interval in half.Begin with an interval covering the whole array. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise narrow it to the upper half. Repeatedly check until the value is found or the interval is empty.Time complexity of binary search is O(Log$_2$n).
   Example:



SET A:

1. Write a C program to accept the following array and find 'x=26' is whether present in array or not.
   A[7] = {11, 5, 45, 26, 12,34,19}
2. WriteaCprogramtoacceptasearchelementformuserandusingbinarysearchmethodfindwhether given element is present or not in the following array.
   A[10] = {1,5,7,12,13,16,17,22,24}

SET B:

1. WriteaCprogramtocreateanarrayofintegers.Acceptavaluefromuseranduselinearsearchmethod to check whether the given number is present in array or not. Display proper message in output.
2. Write a C program to accept n elements from user store it in an array. Accept a value from the user and use Non- recursive binary search method to check whether the value is present in array or not. Display proper message in output.

SET C:

1) Write a C program to read the data from file 'cities.txt' containing names of 10 cities and their STD codes.Acceptanameofthecityfromuseranduselinearsearchalgorithmtocheckwhetherthename is present in the file and output the STD code, otherwise output "city not in thelist".
2) Write a C program to accept n elements from user store it in an array. Accept a value from the user and use recursive binary search method to check whether the value is present in array ornot.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

INTRODUCTION

1. Linkedlist:-

A linked list is a linear data structure. It is collection of items, which are not stored at contiguous memory locations. The elements in a linked list are linked using pointers.

IMPLEMENTATION OF LINKED LIST

A linked list may be implemented in two ways

1) Static representation

2) Dynamic representation.

1) Static Representation:-

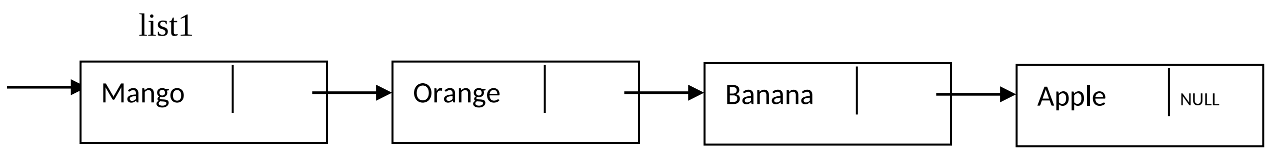
An array is used to store the elements of the list. Two arrays are used : first array to store data and second array to store link.

Data array	Link array
Data[0]=Mango	Link[0] = 3
Data[3] =Banana	Link[3] = 1
Data[1]=Orange	Link[1] =6
Data[6]=Grapes	Link[6] = -1 list end

2) Dynamic Representation:-

• A linked list is a dynamic data structure when a new data is added, the size should increase and when elements are deleted, its size should decrease.

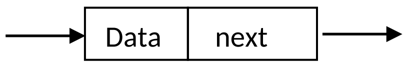
• To store a list in memory, dynamically memory is allocated for each node and linking them by means of pointers since each node will be at random memory location. We will need a pointer to store the address of the first node(head).



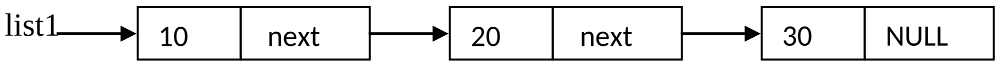
TYPES OF LINKED LIST

- Singly Linkedlist
- Singly Circular Linkedlist
- Doubly Linkedlist
- Doubly Circular Linkedlist

1) Singly Linked list- Each node has two parts : data and next. In ‘data’ we can store any type of data and ‘next’ is a pointer pointing to nextnode.

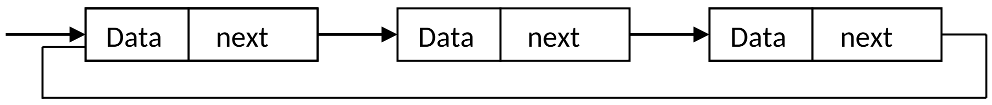


Ex.

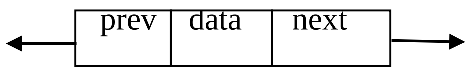


2) Singly Circular Linked list - In this list, the last node points back to the first node i.e. last node contains the address of the first node.

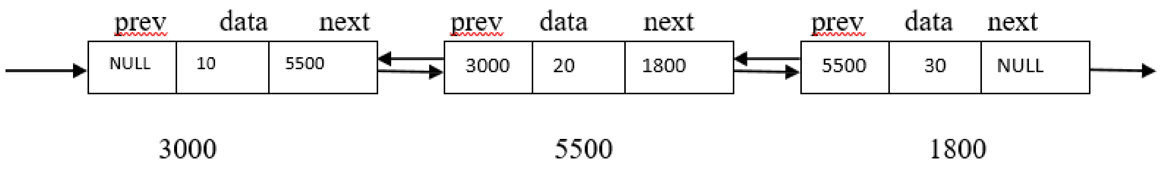
Singly Circular linked list



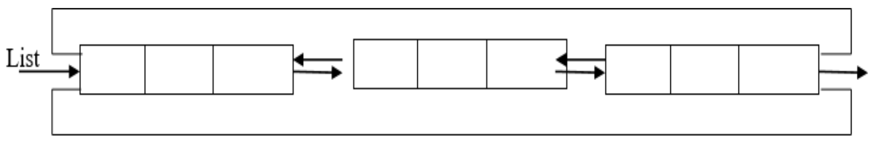
3) Doubly Linked list - Each node in this contains two pointers, one pointing to the previous node and the other pointing to the next node. This list is used when traversing in both directions is required.



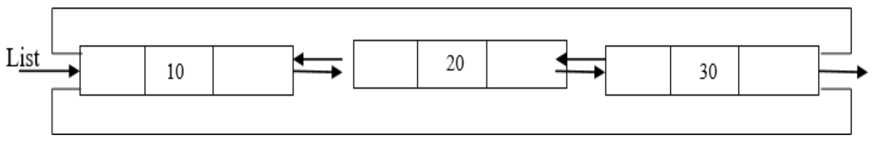
Ex.



4) Doubly Circular Linked list -In this list, the last node does not contain a NULL pointer but points back to the first node i.e. it contains the address of the first node.



Example



OPERATIONS ON A LIST -The following are some of the basic list operation
Some basic Operations on Linked List -

1) **Traversing a list** -Visiting each node of the list is called traversal
2) **Insertion** -A node can be inserted at the beginning, end or in between two nodes of the list.
3) **Deletion** -A node can be deleted from a list either position-wise or element-wise
4) **Display** -Display each element of the list.
5) **Search** -Searches for a specific element in the list.
6) **Reversing** -This process reverses the order of nodes in the list
7) **Concatenation** -This operation joins two lists.
8) **Computation of length** – Count total no. of nodes in a list
9) **Merging of two linked list** – Join two list in sorted manner
10) Union, Intersection, Difference of two linked list

SET A

1) Write a C program to implement a Singly linked list with Create and Display operation.
2) Write a C program to implement a Singly linked list with following operations
   create() , display(), insert(),delete()
3) Write a C program to implement a Singly Circular linked list with following operations
   create(), display(), search(),length()

SET B

1) Write a C program to implement a Doubly linked list with Create and Display operation.
2) Write a C program to implement a Doubly linked list with following operations
   create() , display(), insert(),delete()
3) Write a C program to implement a Doubly Circular linked list with following operations
   create() and display()
4) Write a C program to implement a Doubly Circular linked list with following operations
   create() and display(), append(),delete()

SET C

1) Write a C program to find addition of two polynomials.
2) Write a C program to concatenate two linkedlist.
3) Write a C program to find intersection of two linkedlist.
4) Write a C program to reverse a singly linkedlist.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

INTRODUCTION

- A stack is an ordered collection of items into which items may be inserted and deleted from one end called the top of the stack.
- The stack operates in a LIFO (last in first out) manner. i.e. the element which is put in last is the first to come out. That means it is possible to remove elements from stack in reverse order from the insertion of elements into the stack.
- The real life e.g. of the stack are stack of coins, stack of dishes etc. only the topmost plate can be taken and any new plates has to be put at the top.

PRIMITIVE OPERATIONS ON A STACK.

1) Create
2) Push
3) Pop
4) Isempty
5) Isfull
6) Peek

STACK IMPLEMENTATION:-

The stack implementation can be done in two ways:-

1) Static implementation:-

- It can be achieved using arrays. Though it is very simple method it has few limitations.
- Once a size of an array is declared, its size cannot be modified during program execution.
- The vacant space of stack also occupies memory space.
- In both cases, if we store less argument than declared, memory is wasted and if we want to store more elements than declared array can not be expanded .It is suitable only when we exactly know the number of elements to be stored.

Operations on static stack:-

1) Declaring of a stack:-

A stack to be implemented using an array will require.
- An array of a fixed size.
- An integer called top which stored the index or position of the topmost element. We can use a structure for the above purpose.

2) Creating a stack:-
This declaration only specifies the template. The actual stack can be declared as- STACK s1;
3) initialize a stack:-
Whenastackvariableisdeclaredtheintegertophastobeinitializedtoindicateanemptystack.Since we are using an array the first element will occupy position 0. Hence to indicate an empty stack top has to be initialized to -1
4) Checking whether stack isempty:-
An empty stack can be tested from the value contained in top. If top contains -1 it indicates an empty stack.
5) Checking whether stack isfull:-
Ifthevalueoftopreachesthemaximumarrayindexi.e.MAX-1nomoreelementscanbepushedinto the stack.

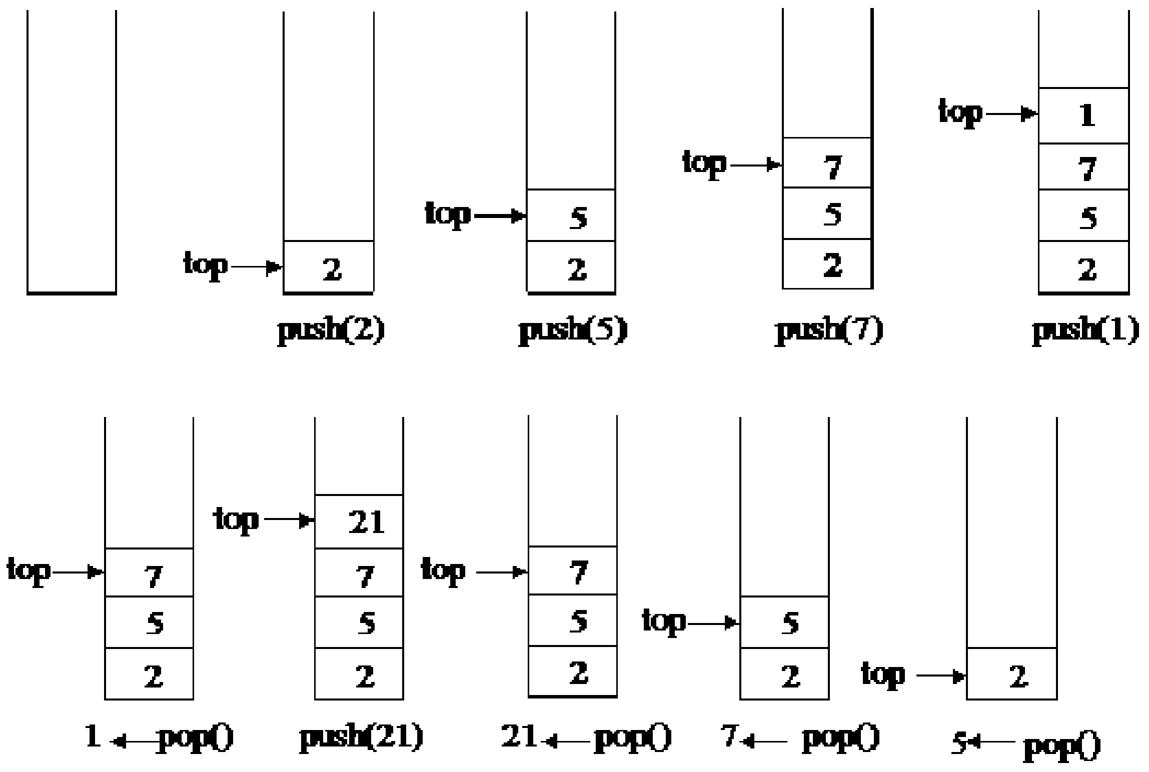
6) The push operation:-
The element can be pushed into the stack only if it is not full. In such case the top has to be incremented first and the element has to be put in this position.
7) The pop operation:

An element can be removed from the stack if it is not empty. The topmost element can be removed after which top has to decremented
8) The peek operation:

It displays the top most element of the stack without decrementing the top.
2) Dynamic implementation:-

- Pointers are used for implementation of stack. The linked list is an e.g. of this implementation.
- The limitations noticed in static implementation can be removed using dynamic implementation. The dynamic implementation is achieved using pointers.
- Using pointer implementation at runtime there is no restriction on the no. of elements. The stack may be expandable.
- The memory is efficiently utilized with pointers.
- Memory is allocated only after element is pushed to the stack
- In static representation there is a limitation on the size of the array if less elements are stored, memory will be wasted. To overcome the program the stack can be implemented using linked list.
- In the linked organization
    - The stack can grow to any size.
    - We need not have prior knowledge of the number of elements.
- When an element is popped the memory can be freed. Thus memory is not unnecessary occupied.
- Since random access to any element is not required in a stack, the linked representation is preferred over the sequential organization.

Example:



SET A :

1) Write a C program to implement Static implementation of stack of integers with following operation:

a) Initialize()
b) push()
c) pop()
d) isempty()
e) isfull()
f) display()
g) peek()

2) Write a C program to implement Dynamic implementation of stack of integers with following operation:

a) Initialize()
b) push()
c) pop()
d) isempty()
e) isfull()
f) display()
g) peek()

SET B :

1) Write a C program to reverse the given string by using static and dynamic implementation of stack.

2) Write a C program to reverse the given number by using static and dynamic implementation of stack.

SET C:

1) Write a C program to reverse the stack by using recursive function.

2) Write a C program to convert infix expression to postfix expression

3) Write a C program to evaluate postfix expression

Assignment Evaluation

0: Not Done[]	1: Incomplete[ ]	2: Late Complete []
3: Needs Improvement[ ]	4: Complete[]	5: Well Done []

INTRODUCTION

A queue is an ordered collection of items from which items may be deleted from (or removed from) one end called the front and into which items may be inserted at the other end called rear.

BASIC OPERATIONS ON QUEUE

1) Create:

Creates a new queue. This operation creates an empty queue.

2) Add orinsert:

Adds an element to the queue. A new element can be added to the queue at the rear.

3) Delete:
Remove an element from the queue. This operation removes the elements, which is at the front of the queue. This operation can only be performed if the queue is not empty.
The result of an illegal attempt to remove an element from an empty queue is called underflow.

4) isempty:

Checks whether a queue is empty. The operation return true if the queue isempty and false otherwise

5) isfull:

Checks whether a queue is full. The operation return true if the queue isfull and false otherwise

REPRESENTATION OF LINEAR QUEUES.

There are two ways to represent a queue in memory.
1) Static (using anarray)
2) Dynamic (using an linkedlist)

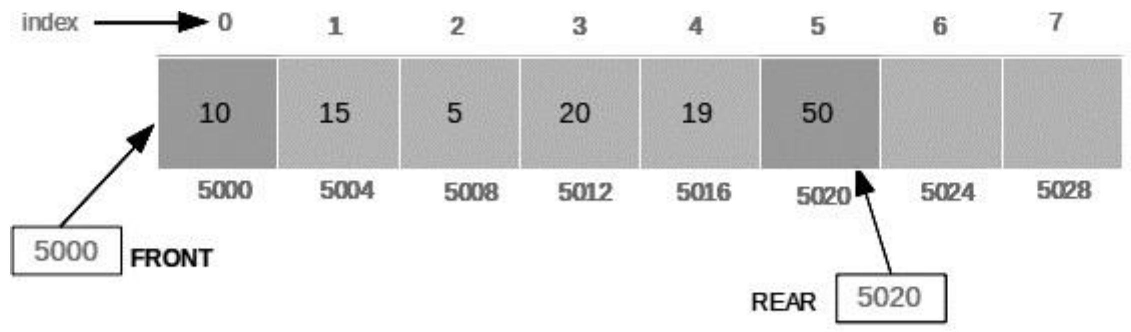
1) Static implementation of queue

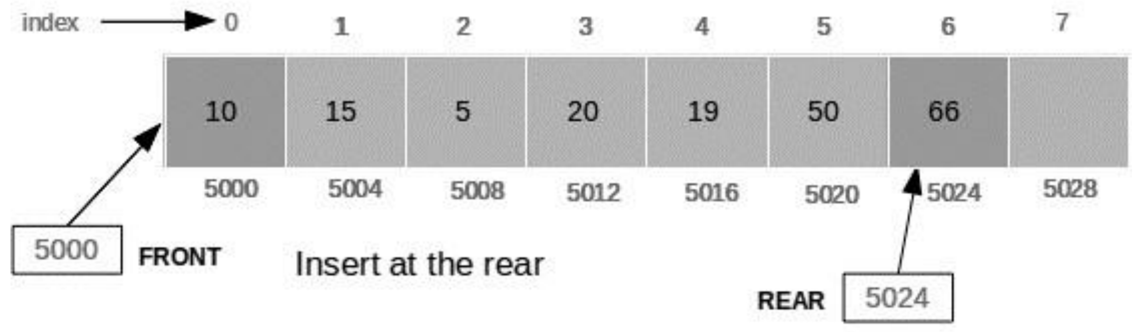
Static implementation or array representation of queue requires three entities-
- An array to hold queue elements.
- A variable to hold the index of the front element.
- A variable to hold the index of the rear element.

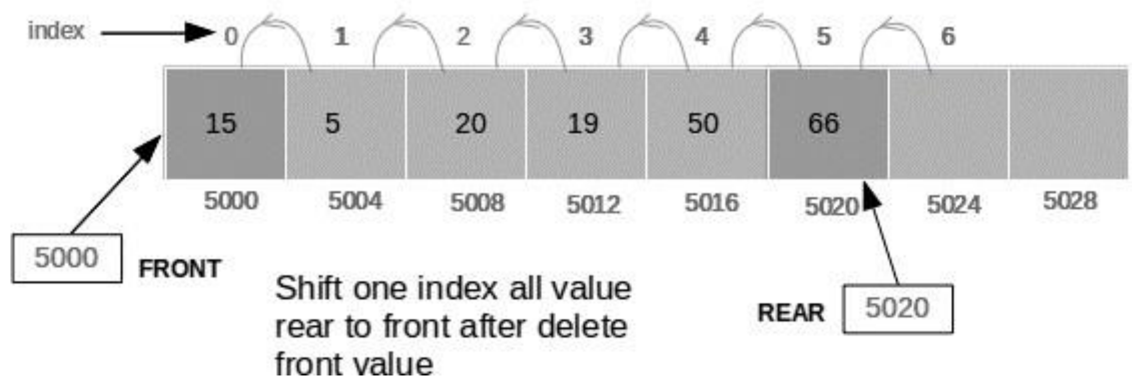
The implementation of a queue using sequential representation is done by using some size MAX and twointegervariablefrontandrear.Initiallyfrontandrearissetto-1.Whenevernewelementisadded it is added from the rear and whenever an element is to be removed from the front. The queue full condition is when rear reaches to MAX - 1. Queue empty condition is when front is equal to rear.

2) Dynamic implementation of linear queue (using a linkedlist)

A queue can be considered as a list in which all insertions are made at one end called the rear and all deletions from the other end from front.
A queue can be easily represented using a linked list. The front and rear will be two pointers pointing to the first and last node respectively.
Example:







SET A:
1) Write a C program to Implement Static implementation of Queue of integers with following operation:

a) Initialize()
b) insert()
c) delete()
d) isempty()
e) isfull()
f) display()
g) peek()

SET B:
1) Write a C program to Implement Dynamic implementation of Queue of integers with following operation:

a) Initialize()
b) insert()
c) delete()
d) isempty()
e) display()
f) peek()

SET C:
1) Write a C program to Implement Static implementation of circular queue of integers with followingoperation:

a) Initialize()
b) insert()
c) delete()
d) isempty()
e) isfull()
f) display()
g) peek()

2) Write a C program to Implement Dynamic implementation of circular queue of integers with followingoperation:

a)Initialize()
b) insert()
c)delete()
d) isempty()
e)display()
f) peek()

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

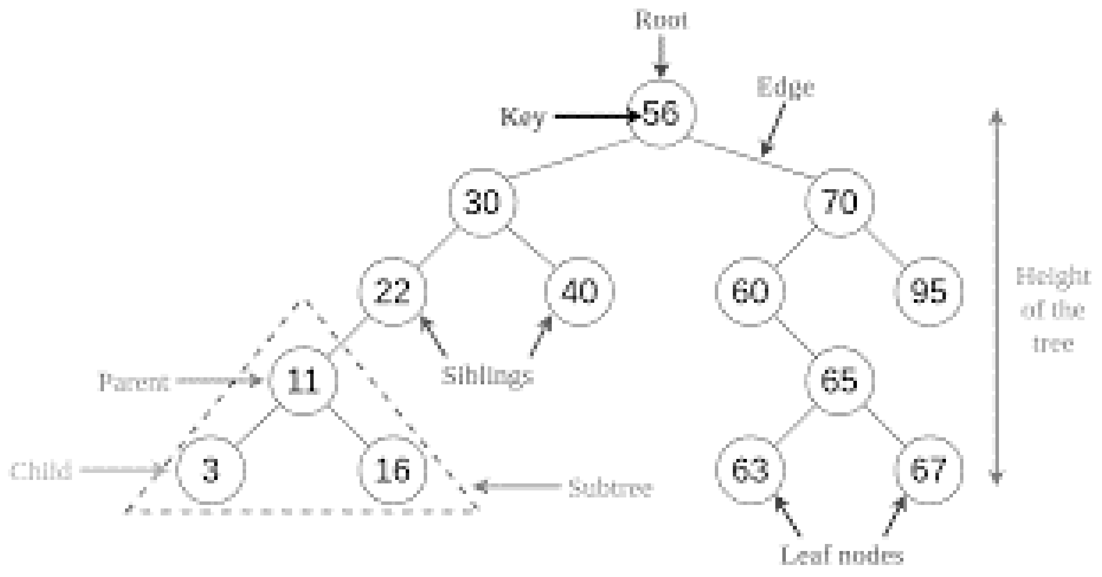
Assignment No: 7 Binary Search Tree (Dynamic)

INTRODUCTION

Binary Search Tree (BST) is a tree in which all the nodes follow the below-mentioned properties-

- The value of the key of the left sub-tree is less than the value of its parent (root) node's key.
- The value of the key of the right sub-tree is greater than or equal to the value of its parent (root) node's key.
- The left and right sub tree each must also be a binary search tree.

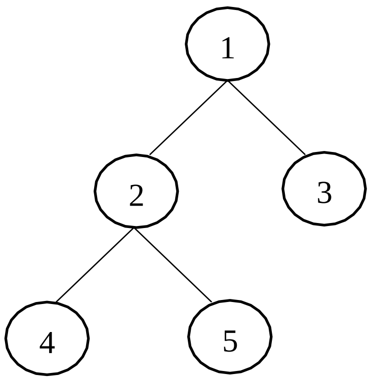
Thus, BST divides all its sub-trees into two segments; the left sub-tree and the right sub-tree and can be defined as – left_subtree (keys) < node (key) ≤ right_subtree (keys)



Basic Operations

Following are the basic operations of a tree –

- **init (T)** – creates an empty Binary search tree by initializing T to NULL
- **Search(T,x)** − Searches an x element in a tree.
- **Insert(T,x)** − Inserts an element in a tree.
- **Pre-order Traversal** (Root, Left, Right)− Traverses a tree in a pre-order manner.
- **In-order Traversal** (Left, Root, Right) − Traverses a tree in an in-order manner.
- **Post-order Traversal** (Left, Right, Root) − Traverses a tree in a post-order manner.



BST-1

Results of BST-1

Pre-Order Traversal : 1, 2, 4, 5, 3

In-Order Traversal : 4, 2, 5, 1, 3

Post-Order Traversal : 4, 5, 2, 3, 1

SET A

1) Write C programs to create and display the elements using Inorder traversal.
2) Write a C program to create binary search tree of integers and perform following operations:
   - Preordertraversal
   - Post ordertraversal

SET B

1. Write a C program to implement binary search tree of integers with following operations:
   - Add a function to insert a new element in the tree
   - Add a function to count non-leaf nodes.
   - Add a function to count leaf nodes.
   - Add a function to count total number of nodes.

SET C

1. Write a C program to implement binary search tree of integers with following operations:
   - Inserting a new element
   - Searching an element
   - Creating mirror of the tree.
   - Find largest and smallest value from a binary search tree

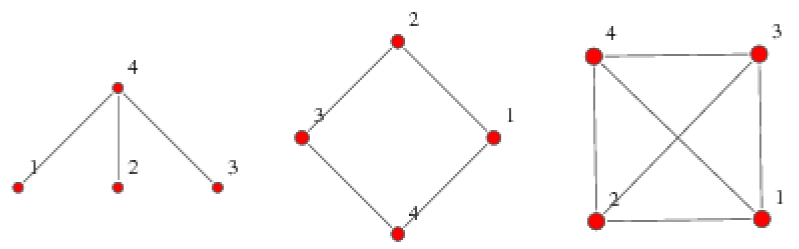
Assignment Evaluation

0: Not Done[]	1: Incomplete[ ]	2: Late Complete []
3: Needs Improvement[ ]	4: Complete[]	5: Well Done []

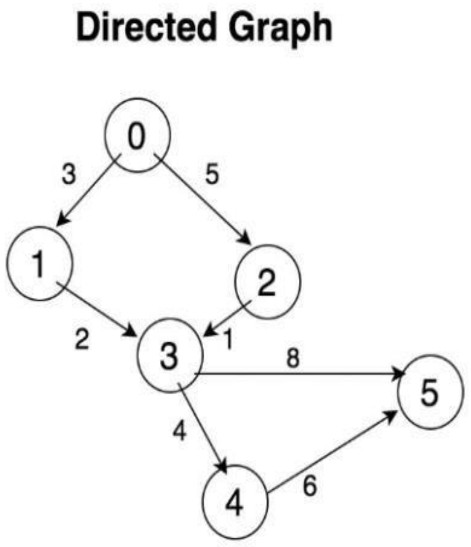
Assignment No: 8 Graph

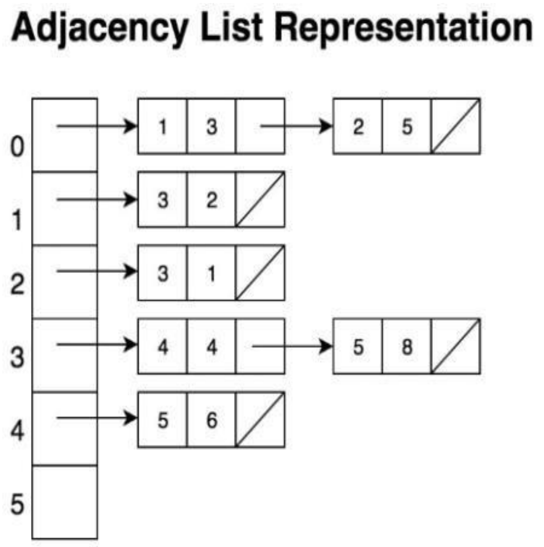
INTRODUCTION

- A graph is a collection of nodes which are called Vertices V, connected in pairs by line segments, called as Edges E. Sets of vertices are represented as V(G) and sets of edges are represented as E(G). So we can represent a graph as G = (V,E). There are two types of graphs : Undirected graph and Directed graph .
- An undirected graph (represented as G = (V,E)) is one where in each edge E is an unordered pair of vertices. Thus, pairs (v1,v2) and (v2,v1) represent the same edge.
- Adirected graph is one,where each edge is represented by a specificd irection or by directed pairs <v1, v2>, where v1 is a tail and v2 is the head of the edge. Hence <v1, v2> and <v2, v1> represent two different edge.
- The two main ways of representing graphs are adjacency matrix representation and adjacency list representation. In adjacency matrix representation of a Graph with n vertices and e edges, a two dimensionalnXnarray,sayA,isused,withthepropertythatA[i,j]equals1ifthereisanedgefrom itojandA[i,j]equals0 ifthereisnoedgefromitoj. Inadjacencylistrepresentationof agraphwith n vertices and e edges, there are n linked lists, one list for each vertex in thegraph.
  Example of Adjacency Matrix



$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$





The usual operations on graph are:

Indegree(i) – returns the indegree (the number of edges ending on) of the ith vertex

Outdegree(i) – returns the outdegree(the number of edges moving out) of the ith vertex

displayAdjMatrix – displays the adjacency matrix for the graph

SET A:

1) Write a C program to read a graph as adjacency matrix and display the adjacency matrix.
2) Add a function in Q1 (above question) to count total degree, indegree and outdegree of the graph.

SET B:

1) Write a C program to convert adjacency matrix into adjacency list. Display the adjacency list.
2) Write a C program to read a graph as adjacency matrix and Traverse using BFS

SET C:

1) Write a C program to read a graph as adjacency matrix and Traverse using DFS
2) Implement a program for simple transpose of the matrix.
3) Write a C program that accepts the graph as an adjacency matrix and checks if the graph is undirected.

   The matrix for undirected graph is symmetric

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]