Tribhuvan University
Faculty of Humanities and Social Science

A Project Report on
"Data Structure and Algorithm - Lab Report"

Submitted to:

Department of ComputerApplication
Padmashree International College

Padmaश्री
INTERNATIONAL COLLEGE

(In partial fulfillment of the requirements for the Bachelor in Computer Application)

Submitted by:

Kabita Shakha

(Batch-2019)

Tribhuvan University
Faculty of Humanities and Social Science
Padmashree International College

LETTER OF APPROVAL

This lab report binds with the entitle "Data Structure and Algorithm-Lab Report" prepared by Kabita ShakhaBatch of 2019 of Padmashree International College, Kathmandu.Registration No:.....As a partial fulfillment for requirement for the degree of Bachelor in Computer Application has been evaluated. In our view, this project includes all the quality elements for the required degree.

.....
Mr. Suresh Kumar MahatoExternal Examiner

.....

Internal Examiner

Bachelor in Computer Application

CERTIFICATE

This is to certificate that the Practical Work entitled "**Data Structure and Algorithm**" is a bonafide work carried out by **Kabita Shakha** in partial fulfillment of 3rd semester **Bachelor In Computer Application of Tribhuvan University**, during the year 2019-2021. The practical report has been approved as it satisfies the academic requirements.

GUIDE

Mr. Suresh Kr. Mahato

Asst. Professor

28-5-21

.....

Signature with Date

HOD

Mr. Raj Kumar Koirala

(Dept. Of BCA)

.....

Signature with Date

PREFACE

This lab report titled as "Data Structure and Algorithm- Lab Report" is prepared and designed as per the direction to fulfill the requirement for the BCA degree conducted by Tribhuvan University.

This report covers all the lab requirement prescribed by TU and each practical topic are included in this project. This lab report may help upcoming batch students as a source of references. I have tried to minimize all the errors presents here as far as possible. Although, every effort has been made to make this lab report as clear and accurate, however suggestion for the improvement needed for the further report are most welcome and will be highly appreciated.

Acknowledgment

To make any lab report, essential guidance and references is mostly required without which proper report cannot be prepared. So, I would like express my gratitude towards Mr. Suresh Mahato, who enable me to complete this lab report successfully in all the aspects. I am also very thankful to Mr. Raj Kumar Koirala, BCA coordinator for his encouragement and importance guidance regarding this project. Similarly, I would like to express thank to all the teaching and non-teaching staff of PadmaShree International College for providing importance resources to complete this report.

Lastly, I am very grateful to Tribhuvan University (FOHSS), Dean Office for providing this opportunity and addressing this lab report in BCA curriculum so that it will help in building great career through the practical knowledge.

Kabita Shakha

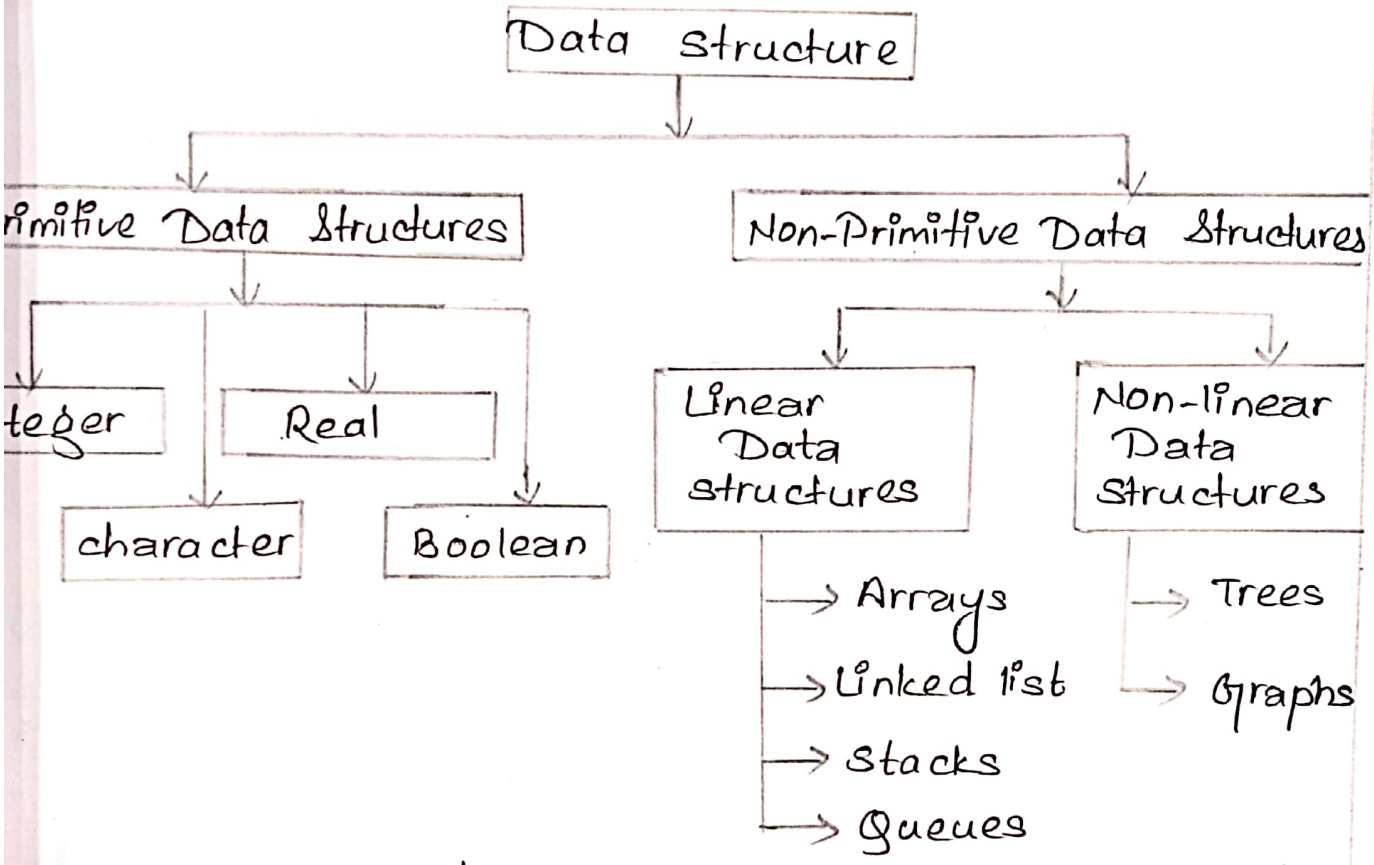
(Batch of 2019)

Table of contents

page no.

INTRODUCTION TO DATA STRUCTURES

Data structure is defined as the way in which data is organised in the memory location. There are 2 types of data structures:



Linear Data Structure

In linear data structure all the data are stored linearly or continuously in the memory. All the data are saved in continuously memory locations and hence all the elements are saved in one boundary. A linear data structure is a one in which we can reach directly only one element from another while travelling sequentially. The main advantage, we find the first element, and then it's easy to find the next data elements. The disadvantage, the size of the array must be known before allocating the memory.

The different types of linear data structures are :-

Array
Stack
Queue
Linked list.

Non-Linear Data Structure :-

Every data item is attached to several other data items in a way that is specific for reflecting relationships. The data items are not arranged in a sequential structure.

The various types of non-linear data structures are :-

Trees
Graphs

Experiment 1:

Design, develop and implement a menu driven Program in C for the following operations on STACK of integers (Array Implementation of stack with maximum size MAX)

a) Push an element on to stack.
b) POP an element from stack.
c) Demonstrate overflow and underflow situations on stack.
d) Display the status of stack.
e) Exit.

Support the program with appropriate functions for each of the above operations.

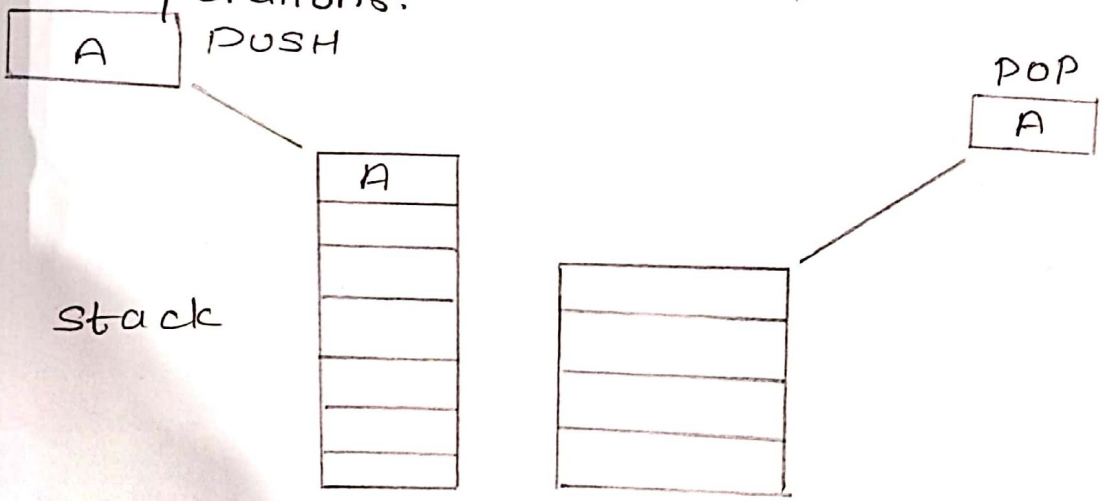
LAB NO: 1.

IMPLEMENTATION OF DIFFERENT OPERATION'S RELATED TO STACK.

Theory:

A stack is an abstract data type (ADT), commonly used in most programming languages. This feature makes it LIFO (last-in-fist-out) data structure. In stack terminology, insertion operation is called PUSH operation and removal operation is called POP operation.

Below given diagram tries to depict a stack and its operations:



A stack can be implemented by means of array, structure, pointer and linked list. Stack can either be fixed size one or it may have a sense of dynamic resizing. Here, we are going to implement stack using array() which makes it a fixed size stack implementations.

Basic operations:

push() - pushing (storing) an element on the stack.

pop() - removing (accessing) an element from the stack.

To use a stack efficiently we need to check status of stack as well. For the same purpose, the following functionally is added to stack:

PROGRAM CODE:

```
#include <stdio.h>
#include <conio.h>

#define MAX 4

int stack[MAX];item;
int ch, top=-1, count=0, status=0;

/* PUSH FUNCTION */
void push(int stack[], int item)
{
    if (top==(MAX-1))
      printf("\n\n stack is overflow");
    else
     {
        stack[++top]=item;
        status++;
     }
}
/* POP FUNCTION */
int pop(int stack[])
{
      int ret;
      if (top==-1)
       printf("\n\n Stack is underflow");
```

```
est else
{
    ret = stack [top--];
    status--;
    Printf ("\n Popped element is %d", ret);
}
 return ret;
/*FUNCTION TO CHECK STACK IS PALINDROME OR NOT*/
void palindrome (int stack[])
{
  int i, temp;
  temp = status;
  for (i=0; i<temp; i++)
  {
    if (stack[i] == pop(stack))
    count ++;
  }
  if (temp== count)
   Printf ("\n stack contents are Palindrome");
  else
  Printf ("\n stack contents are not palindrome");
}
/* FUNCTION TO DISPLAY STACK*/
void display (int stack[])
{
  int i;
 Printf ("\n The stack contents are :");
  if (top== -1)
    Printf ("\n stack is Empty");
```

```
else {
        for (i= top; i>= 0; i--)
        printf ("\n----- \n| %d |", stack[i]);
        printf ("\n");
    }
}
/* MAIN PROGRAM */
void main( )
{
  clrscr( );
  do {
       printf ("\n\n---- MAIN MENU ----\n");
       printf ("\n1. PUSH (Insert) in the stack");
      printf ("\n2. POP (Delete) from the stack");
      printf ("\n3. PALINDROME check using stack");
      printf ("\4. Exit (End the Execution)");
      printf ("\n Enter your choice:");
       scanf ("%d", &ch);

      switch (ch) {
      cas 1: printf ("enter a element to be pushed:");
             scanf ("%d", &item);
             push (stack, item);
             display (stack);
             break;
```

```
case 2: item = pop(stack);
        display(stack);
        break;

case 3:
        palindrome(stack);
        break;

case 4:
        exit(0);
        break;

default:
        printf("\n END OF EXECUTION");
  } //end switch
} while(ch!=4);
  getch();
}
```

OUTPUT:

...MAIN MENU...
1. PUSH (Insert) in the stack
2. POP (Delete) from the stack.
3. PALINDROME check using stack
4. Exit (End the execution)
Enter your choice:
Enter on element to be pushed:
The stack contents are:
...
111
...MAIN MENU...
1. PUSH (Insert) in the stack
2. POP (Delete) from the stack
3. PALINDROME check using stack
4. Exit (End the Execution)
Enter your choice: 1
Enter on element to be pushed: 1
The stack contents are:
...
121
(...AFTER THE 4 TIMES PUSH OPERATION...)
...MAIN MENU...
1. PUSH (Insert) in the stack
2. POP (Delete) from the stack
3. PALINDROME check using stack
4. Exit (End the execution)

Enter your choice: 1
Enter an element to be pushed: 9
Stack is Overflow
The stack contents are:
...
111
...
121
.....1

```
21
....1
11
...MAIN MENU...
1. PUSH (Insert) in the stack
2. POP ( Delete) from the stack
3. PALINDROME check using stack
4. Exit ( End the Execution)
Enter Your Choice: 2
Popped element is: 1
The stack contents are:
...
121
...1
21
11
(... AFTER THE 4 TIMES POP OPERATION ...)
... MAIN MENU...
1. PUSH ( Insert) in the stack
2. POP ( Delete) from the stack
3. PALINDROME check using stack
4. Exit ( End the execution)
Enter Your choice: 2
Stack is underflow.
The stack contents are:
Stack is Empty
(... CHECKING FOR PALINDROME OR NOT USING
    STACK...)
... MAIN MENU...
1. PUSH ( Insert ) in the stack
2. POP (Delete) from the stack
3. PALINDROME check using stack
4. Exit ( End the @execution)
```

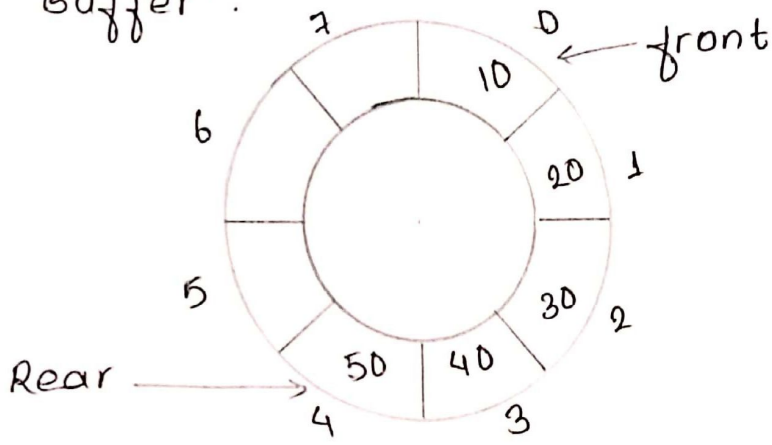
```
Enter Your choice: 1.
Enter on element to be pushed : 1.
The stack contents are:
...
|1|
.. MAIN MENU...
. PUSH (Insert) in the stack
. POP (Delete) from the stack
. PALINDROME check using stack
. Exit (End the execution)

Enter Your choice : 1.
Enter an element to be pushed : 2
The stacks contents are:
...
|2|
...|
|1|
.. MAIN MENU...
1. PUSH (Insert) in the stack
. POP (Delete) from the stack
. PALINDROME check using stack
1. Exit (End the execution)
et Enter Your choice : 1.
nter a element to be pushed : 2
The stack elements are:
...
|1|
2'...|
 2|
...|
 1|
.. MAIN MENU...
. PUSH (Insert) in the stack
 POP (Delete) from the stack
. PALINDROME check using stack
1. Exit (end the execution)
```

Enter Your choice: 3
Stack contents are palindrome.
... CHECKING FOR PALINDROME OR NOT USING STACK...)
... MAIN MENU...
. PUSH (Insert) in the stack
. POP (Delete) from the stack
3. PALINDROME (check using stack)
. Exit (End the execution)
Enter Your choice: 1.
Enter an element to be pushed: 1
The stack contents are:
...1
1
(AFTER 3 TIMES PUSH)
.. MAIN MENU...
. PUSH (Insert) in the stack
. POP (Delete) from the stack
3. PALINDROME check using stack
. Exit (End the execution)
Enter Your choice: 1
Enter an element to be pushed: 3
The stack contents are:
...
31
..1
L1
...1
11
.. MAIN MENU...
. PUSH (Insert) in the stack
. POP (Delete) from the stack
. PALINDROME check using stack
. Exit (End the execution)
Enter Your choice: 3
Stack contents are not palindrome.

Introduction:

Circular Queue is a linear data structure in which the operations are performed based on FIFO (first in first Out) principle and the last position is connected back to the first position to make a circle. It is also called 'Ring Buffer'.



It is very similar to queue. The ~~not~~ only difference is that that ~~to~~ last node is connected back to the first node. Thus, it is called circular queue.

Operations on circular queue:

- Enqueue: Adding an element in the queue if there is space in the queue.
- Dequeue: Removing elements from the queue if there are any elements in the queue.
- Front: get the first item from the queue.
- Rear: get the last item from the queue.
- is Empty/is full: check if queue is empty or full.

Peak( ) - get the top data element of the stack, without removing it.

isFULL( ) - check if stack is full.

isEmpty( ) - check if stack is empty.

LAB NO: 2

```
#include <stdio.h>
#include <conio.h>
#define MAX 4
int ch, front = 0, rear = -1, count = 0;
char q [MAX] item;
void insert ( );
{
  if (count == MAX)
  Printf ("\n Queue is Full");
  else
  {
    rear = (rear + 1) % MAX;
    q [rear] = item;
    count ++;
  }
}
void del ( )
{
  if (count == 0)
  Printf ("\n Queue is empty");
  else
    {
      if (front > rear && rear == MAX-1)
      {
```

```
    front = 0; rear = -1; count = 0;
}

else
{
  item = q[front];
  printf("\n Deleted item is : %c", item);
  front = (front + 1) % MAX;
  count--;
  }
}

void display()
{
 int i, f = front, re = rear;
 if (count == 0)
printf("\n Queue is Empty");
 else
 {
   printf("\n Contents of Queue is : \n");
   for (i=f; i<=r; i++)
   {
    printf("%c \t", q[i]);
     f = (f+1)%MAX;
   }
 }
 void main()
```

```
{
 do
 {
  Printf("\n1. Insert \n2. Delete \n3. Display
        \n4. Exit");
  Printf("\n Enter the choice:");
   scanf ("%d", &ch);
   switch (ch)
   {
    case 1: printf("\n Enter the character:");
            scanf ("%c", &item);
            insert();
            break;
     case 2: del();
             break;

     case 3: Display();
             break;

      case 4:
              break;

     }
  } while (ch!=4);
    getch();
}
```

OUTPUT:

1. Insert 2. Delete 3. Display 4. Exit
Enter the choice: 1
Enter the character /item to be inserted: A

1. Insert 2. Delete 3. Display 4. Exit
Enter the choice : 1

Enter the character /item to be inserted : B

1. Insert 2. Delete 3. Display 4. Exit
Enter the Choice: 1
Enter the character /item to be displayed: C

1. Insert 2. Delete 3. Display 4. Exit
Enter the choice: 1
Enter the character/item to be inserted: D
Contents of Queue is:-
A B C D

1. Insert 2. Delete 3. Display 4. Exit
Enter the choice: 1
Enter the character/item to be inserted : F
Queue is full.

1. Insert 2. Delete 3. Display 4. Exit
Enter the choice: 2
Deleted item is:- A

1. Insert 2. Delete 3. Display 4. Exit
Enter the choice:- 2
Deleted item is :- B

1. Insert 2. Delete 3. Display 4. Exit
Enter the choice: 3
Contents of Queue is:-
C D

1. Insert 2. Delete 3. Display 4. Exit
Enter the choice:- 1
Enter the character /item to be inserted: K

1. Insert 2. Delete 3. Display 4. Exit
Enter the choice: 3
Contents of Queue is:
C D K

1. Insert 2. Delete 3. Display 4. Exit
Enter the choice: 4

Introduction:

Recursion is a process by which a function calls itself repeatedly, until some specified condition has been satisfied. The process is used for repetative computations in which each action is stated in terms of a previous result. In order to solve a recursively two condition must be satisfied:

- Each time a function calls itself at it must be closer to solution.
- The problem statement must include a stopping condition.

Tower of Hanoi Problem:

Initial State:

- There are 3 poles named as origin, Intermediate and destination.
- n number of different-sized disks having hole at the center is stacked around the origin pole in decreasing order.
- The disks are numbered as: 1,2,3,4...n.

Objective:

- Transfer all disks from origin pole to destination pole using intermediate pole for temporary register.

Conditions:

- Move only one disk at a time.
- Each disk must always be placed around one of the pole.
- Never place larger disk on top of smaller disk.

Algorithm:

To move a tower of n disks from source to destination (where n is positive integer)

1. If n == 1.
   → Move single disk from source to destination.

2. If n > 1.
   → let temp be the remaining pole other than source and destination.
   → Move a tower of (n-1) disks from source to temp.
   → Move a single disk from source to destination.
   → Move to a tower of (n-1) disks from temp to destination.

3. Stop.

LAB NO: 3

SOLUTION OF FIBONACCI SERIES USING RECURSION

```
#include <stdio.h>
#include <conio.h>
int main ( )
{
  int n, i;
  int fibo(int);
  printf("enter n: ");
  scanf ("%d", &n);
  printf(" Fibonacci number up to %d terms: \n",n);
  for (i=1; i<=n; i++)
  printf( "%d \n," fibo(i));
  getch( );
}
int fibo(int k)
{
  if(k==1 || k==2)
  return 1;
  else
  return fibo(k-1) + fibo(k-2);
```

OUTPUT:

```
Enter n:
fibonacci number up to 5 terms:
  1
  2
  3
  4
  5
```

LAB 8:

SOLUTION OF TOH USING RECURSION

```
#include <stdio.h>
#include <conio.h>
void TOH (int, char, char, char);
void main ( )
{
 int n;
printf ("Enter number of disks:");
scanf ("%d", &n);
TOH (n, 'O', 'D', 'I');
getch ( );
}
void TOH (int n, char A, char B, char C)
{
 if (n >0)
{
TOH (n-1), A, C, B);
Printf ("Move disk %d from %c to %c\n",
                    A, B);
 TOH (n-1, C, B, A);
 }
}
```

OUTPUT:

```
Enter number of disks: 3
Move disk 1 from O to D
Move disk 2 from O to A
Move disk 1 from D to I
Move disk 3 from O to D
Move disk 1 from I to D
Move disk 2 from I to D
Move disk 1 from O to D.
```

Double linked list:

A linked list in which all nodes are linked together by multiple numbers of links i.e. each node contains three fields (two pointer fields and one data field) rather is called doubly linked list. It provides bidirectional traversal. It can be conceptualized as two singly linked lists formed from the same data items, but in opposite sequence orders.

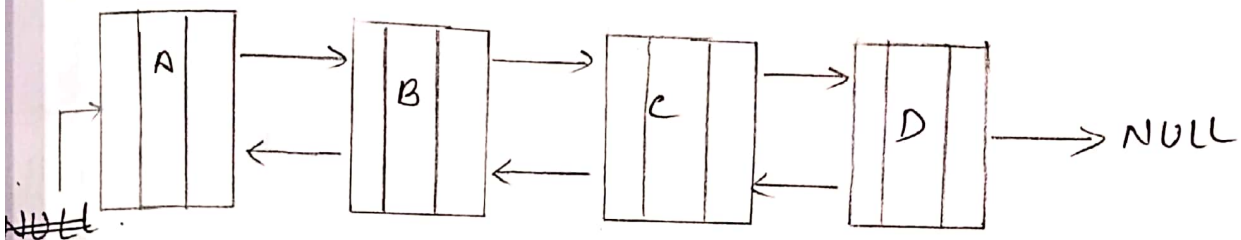


fig: Double linked list with three nodes

n doubly circular linked list, the previous link of the first node points to last node and next link of the last node points to first node. In doubly circular linked list, each node contains two fields called links used to represent reference to the previous and the next node in the sequence of nodes.

LAB NO: 4.

Implementation of double linked list:

```
#include <stdio.h>
#include <stdlib.h>

/* Declaring a structure to create a node */
struct node
{
  struct node * prev;
  int data;
  struct node * next;
};

struct node * struct *, nt;
// Inserting nodes into the list.
// function to insert values from beginning
   of double linked list.

void insertbeg (void)
{
  int a;
struct node * nn, * temp;
// Allocating implicit Memory to node
 nn = (struct node *) malloc (size of (struct node));
Printf ("Enter data:");
 scanf (" %d ", && nn -> data);
 a = nn -> data;
 if (strart == NULL) /* Checking if list is empty*/
{
  nn -> prev = nn -> next = NULL;
```

```
    start = nn;
  }
  else
  {
  nn->next = start;
  nn->prev = NULL;
  start->prev = nn;
  }
  printf("%d succ inserted \n", a);
}
/* Function to insert values from the end of
   linked list */
void inserted(void)
{
  int b;
  struct node *nn, *ip;
  nn = (struct node*) malloc (size of (struct node));
  Printf("Enter data:");
  scanf("%d", && nn->data);
  b = nn->data;
  if (start == NULL)
  {
    nn->prev = nn->next = NULL;
    start = nn;
  }
  else
  {
    ip = start;
    while (ip->next != NULL)
    {
      IP = IP->next;
```

```
    }
    nn->prev = lp;
    lp->next = nn;
    nn->next = NULL;
    }
    printf("%d successfully Inserted \n", b);
    }
void insertmid(void)
{
    struct node *nn, *temp, *ptemp;
    int x, c;
    if (start = NULL)
    {
    printf("All is empty \n");
      return;
    }
    printf("Enter data before which nn is to be
     inserted \n");
     scanf("%d", &x);
     if (x = start->data)
    {
      insertbeg();
    }
     ptemp = start;
      temp = start->next;
      while (temp->next != NULL && temp->data!x)
    {
     ptemp = temp;
      temp = temp->next;
    }
```

```
if (temp == NULL)
{
printf ("%d doesnot exit \n",n);
}
else
{
nn = (struct node *) malloc (size of (struct node));
printf = ("Enter data :");
scanf ("%d", &nn -> data);
{ c = nn -> data;
  nn -> data;
  nn -> prev = ptemp;
 ptemp -> next = nn;
temp -> prev = n;
printf ("%d successfully inserted \n", c);
}

}

void deletion ( )
{
 struct node * pt, * t;
 int n;
t = pt = start;
 if (start == NULL)
{
 printf ("dll is empty \n");
}
 printf ("Enter data to be deleted :");
 scanf ("%d", &n);
 if (n == start -> data)
{
    t = start;
```

```
    t = t -> next;
    free (start);
     start = t;
    start = pt;
}
else
{
while (t = next != NULL && t -> data != x)
{
  pt = t;
  t = t -> next;
}
if (t -> next == NULL && t -> data == x)
{
  free (t);
   pt -> next = NULL;
}
else
{
 if (t -> next == NULL && t -> data != x)
printf ("data not found ");
else
{
 pt -> next = t -> next;
 free (t);
}
}
Printf ("%d is successfully deleted \n",
                                  x);
}
}
```

```
void display ( )
{
struct node * temp ;
if (start == NULL)
printf (" stack is empty");
temp = start;
while (temp -> next != NULL)
{
printf ("%d \n", temp -> data);
 temp = temp -> next;
}
printf ("The content is %d \n", temp -> data);
}
int main ( )
{
 int c, a;
start = NULL;
do
{
printf ("\n 1. Insert \n 2. Delete \n 3. Display
      \n 4. Exit \n Enter choice: ");
scanf ("%d ", & c);
 switch ( )
{
case 1: printf ("1. Insert beg \n 2. Inserted \n 3.
        Insert mid \n Enter choice: ");
 switch ( a)
 {
  case 1: insert beg ( );
          break;
```

```
case 2: insert end ( );
        break;
case 3: insert mid ( );
        break;
}
break;
case 2: deletion ( );
        break;
case 3: display ( );
        break;
case 4: printf ("program ends\n");
        break;
default : printf ("Wrong choice\n");
          break;
  }
}
while (c!=4);
relation 0;
}
```

Output

1. Insert
2. Delete
3. Display
4. exit

Enter choice: 1

1. Insertbeg
2. Insertmid
3. Insertend

Enter choice : 1
Enter data : 8

8 successfully inserted

1. Insert
2. Delete
3. Display
4. Exit

Enter choice: 1

1. Insertbeg
2. Insert mid
3. Insert end

Enter choice: 2
Enter data : 9

9 successfully inserted.

Insert
Delete
Display
Exit

Enter choice : 1

Insert beg
Insertmid
Insertend

Enter choice : 3
Enter data : 12

12 successfully inserted

1. Insert
2. Delete
3. Display
4. Exit

Enter choice : 2
Enter data to be deleted : 12

2 successfully deleted.

Insert
Delete
Display
Exit

Enter choice : 4
Program ends.

Merge sort:

Merge sort is an efficient sorting algorithm which involves merging two or more sorted files into a third sorted file. Merging is the process of combining two or more sorted files into a third sorted file. the ~~mea~~ merge sort algorithm is based on divide and conquer method.

The process of merge sort can be formalized into three basic operations:

- Divide the array into two sub arrays.
- Recursively sort the two sub arrays.
- Merge the newly sorted sub arrays.

LAB NO: 6.

Implementation of Merge sort program

```
#include <stdio.h>
#include <conio.h>
void merge (int a[ ], int l, int m, int r)
{
 int x = l;
 int k = l-1;
 int b[10];
 int y = m;
 while (x<m && y<=r)
 {
  if (a[x]<a[y])
  {
   b[k] = a[x];
   k++;
   x++;
  }
  else {
   b[k]=a[y];
   k++;
   y++;
  }
 }
while (x<m)
{
 b[k] = a[x];
 k++;
 x++;
}
while (y<=r)
{
 b[k] = a[y];
 y++;
 k++;
```

```
}
for(i=l ; i<=r; i++)
{
a[i]=b[i];
}

}
void merge_sort (int a [ ], int l, int r)
{
int mid;
if (l<r)
{
mid = (l+r)/2;
merge_sort ( a,l, mid);
merge_sort (a, mid+1, r);
merge (a, l, mid+1, r);
}

}
void main ( )
{
int a[100], n, i, l, r;
Printf ("Enter no. of elements: \n");
scanf ("%d", &n);
Printf ( "Enter %d elements ", n);
l=0;
r=n-1;
for (i=0; i<n; i++)
{
scanf ("%d", &a[i]);
}
Printf ("elements before sort: \n");
for (i=0; i<n; i++)
{
Printf ("%d \t", a[i]);
}
```

```
merge_sort(a, l, r);
printf("\n elements after sort : \n");
for(i=0; i<n; i++)
{
printf("%d\t", a[i]);
}
getch();
}
```

OUTPUT:

Enter no. of data points

5

Enter 5 numbers

3 12 9 10 11

The data item before sorting

3 12 9 10 11

The data item after sorting

3 9 10 11 12

Searching:

Searching is an operation or a technique that helps find the place of a given element or value in the list. Any search is said to be successful or unsuccessful depending upon whether the element that is being searched is found or not. Some of the standard searching technique that is being followed in data structure is listed below:

- Sequential Search
- Binary Search

. Sequential Search:

The main agenda of this search is to find the given elements in a list of elements with O(n) time complexity; where 'n' is the total no. of elements in the list.

Algorithm:

Start
Read the search element from the user.
Compare, the search element with the first element in the list.
If both are matching then display 'Given elements found!!!" and terminate the function
If both are not matching, then compare search element with next element in the list.
Repeat steps 4 & 5 until search element is compared with last element in the list.
If the last element in list also ~~des~~ doesn't match, then display "Element not found!!!" and terminate function.
Stop.

LAB No:

```
#include<stdio.h>
int main( )
{
 int a[100], key, i, n;
 printf("Enter the number of elements in array
                \n");
 scanf("%d", &n);
 printf("Enter %d numbers \n", n);
 for (i=0; i<n; i++)
 {
  scanf("%d", &a[i]);
 }
 printf("Enter the number to search \n");
 scanf ("%d", &key);
 for (i=0; i<n; i++)
 {
  if (a[i]==key)
  {
   printf("%d is present at location %d \n", key,
          i+1);
    break;
  }
}
 if (i==n)
```

```
printf("%d is not present in array.\n", key);
  return 0;
}
```

OUTPUT:

Enter the number of elements in array.

4

Enter a number

4 5 14 15

Enter the number to search

4

4 is present at location 1.

Binary search:

The main agenda of this search is also to find the given element in a list of elements but with O(log n) time complexity where 'n' is the total no. of elements in list, which means it can be only used with list of elements which are already arranged in an order.

Algorithm:

start

Read the search element from the user.

. find the middle element in sorted list.

. compare the search element with the middle element in the sorted list.

. If both are matching, then display "Given elements found!!!" and terminate the function.

. If both are not matching, then check whether the search element is similar or larger than middle element.

. If search element is similar than middle element, then repeat steps 2,3,4 and 5 for the left sub list of the middle element.

. If search element is larger than ~~mid~~ middle element, the repeat steps 2,3,4 and 5 for the right sub list of the middle element.

. Repeat the same process until we find the search element in the list or until sub list contains only w. one element.

. If that element doesn't match with search element then display "element not found" & terminate function.

11. Stop.

LAB NO: 7b

```
#include <stdio.h>
#include <conio.h>
int Binarysearch (int a[100], int l, int r, int key)
int m;
int flag=0;
if (l<=r)
{
   m=(l+r)/2;
   if (key==a[m])
   flag=m;
   else if (key<a[m])
   return Binarysearch (a, l, m-1, key);
   else
   return Binarysearch (a, m+1, r, key);
   }
  else
    return flag;
}
void main ()
{
  int n, a[100], i, key, flag;
 Printf ("Enter no. of data items: \n");
  scanf ("%d", &n);
  Printf ("Enter %d data items in sorted form:
                                \n", n);
   for (i=0; i<n; i++)
    scanf ("%d", &a[i]);
```

```
printf (" Enter searched item");
scanf ("%d", & key);
flag = Binary Search (a, 0, n-1, key);
 if (flag == 0)
printf ("Search un-successful");
 else
printf ("search successful and found at
          location %d", flag+1);
}
```

OUTPUT:

Enter no. of data item: 5

Enter data item in sorted form:-

24 25 34 35 44

Enter searched item: 25

Search successful and found at location 2.

Graph Traversals:

Traversing a graph means visiting all the vertices in a graph exactly ~~ont~~ one.

Two common approaches for traversal are:.

- BFS
- DFS

BFS :

BFS is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root and explores the neighbour nodes first before moving to the next level neighbours. BFS uses a queue to rember to get the next vertex to start a search.

DFS:

DFS is an algorithm for traversing or searching tree or graph data structures. One starts at the root and explores as far as possible along each branch ~~d~~ before back tracking.

In DFS, there is possible of three coloring stages:

- while : unvisited;
- gray : vertex in progress;
- blacl : finished processing the vertex;

Breadth-first search(BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root & explores the neighbour nodes first before moving to the next level neighbours.

Depth-first search(DFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the root (selecting some arbitrary node as the root in the case of graph) & explores as far as possible along each other branch before backtracking.

ALGORITHM:-

Start 1:- Start.

Step 2:- Input the value of N nodes of the graph.

Step 3:- Create a graph of N nodes using adjacency matrix representation.

Step 4:- Print the nodes reachable from the starting node using DFS.

Step 5:- Check whether graph is connected or not using DFS.

Step 6:- Stop.

LAB NO: 08

PROGRAM CODE:

```
#include <stdio.h>
#include <conio.h>
int a[10][10], n, m, i, j, source, s[10], b[10];
int visited[10];
void create( )
{
printf("\n Enter the number of vertices of the diagraph:");
scanf("%d", &n);
printf("\n Enter the adjacency matrix of the graph:\n");
for (i=1; i<=n; i++)
for (j=1; j<=n; j++)
scanf("%d", &a[i][j]);
}
void bfs( )
{
int q[10], u, front=0, rear=-1;
printf("\n Enter the source vertex to find other
        nodes reachable or not:");
scanf("%d", &source);
q[++rear]=source;
visited[source]=1;
printf("\n the reachable vertices are:");
while(front<=rear)
```

```
{
  u = q[front++];
  for (i=1; i<=n; i++)
  {
    if (a[u][i]==1 && visited[i]==0)
    {
      q[++rear]=i;
      visited[i]=1;
      printf("\n %d", i);
    }
  }
}
void dfs(int source)
{
  int v, top=-1;
  s[++top]=1;
  b[source]=1;
  for(v=1; v<=n; v++)
  {
    if (a[source][v]==1 && b.[v]==0)
    {
      printf("\n %d -> %d", source, v);
      dfs(v);
    }
  }
}
```

```
void main ( )
{
 int ch;
while (1)
{
  printf("\n 1. create Graph \n2. BFS \n3. check
        graphs connected or not (DFS) \n4. exit");
  printf ("\n Enter your choice:");
   scanf ("%d", & ch);
   switch (ch)
   {
   case 1: create ( );
           break;
   case 2: BFS ( );
          for (i=1; i<=n; i++)
          if (visited [i]==0)
        printf ("\n Enter the source vertex to
           find the connectivity:");
        scanf ("%d", & source);
        m=1;
        dfs (source);
        for (i=1; i<=n; i++)
      {
        if (b[i]==0)
        m=0
      }
        if (m==1)
        Printf("\n Graph is connected");
```

```
        break;
    }
  }
}
```

OUTPUT:

Create Graph 2. BFS 3. Check graph connected or not (DFS)
1. Exit

nter Your choice: 1
nter the number of vertices of the diagraph: 4.
nter the adjacency matrix of the graph

0 0 1 1
0 0 0 0
0 1 0 0
0 1 0 0

. Create Graph 2. BFS 3. check graph connected or not.
4. Exit.

nter Your choice: 2
nter the source vertex to find other nodes reachable or not: 1.

3
4
2

. Create Graph
. BFS
. Check graph connected or not (DFS)
. Exit.

Enter Your Choice: 3

Enter the source vertex to find the connectivity:

1 → 3

3 → 2

1 → 4

Graph is connected.

1. Create Graph
2. BFS
3. Check graph connected or not
4. Exit

Enter your choice: 4

Heap sort:

A heapsort is an almost complete tree whose elements have keys that satisfy the following heap property; the keys along any path from root to leaf are descending (ie. non increasing).

In brief, a heap is an almost complete binary tree of n nodes such that the value of each node is less than or equal to the value in parent node. This type of heap is called max heap. By default, the heap is max heap. There are two types of heap:

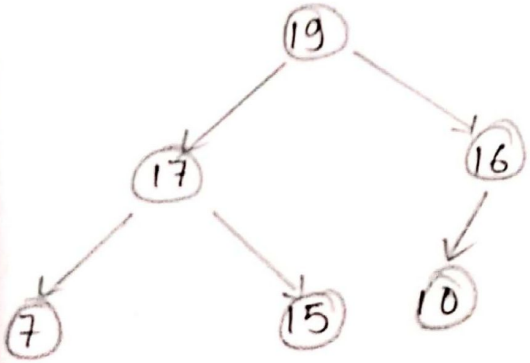
- Max heap and
- Min heap

Heap are used to implement priorities queues and to the heap sort algorithm.

To Demonstrate heap element:

we consider a following example:

[15, 19, 10, 7, 17, 16]



LAB : 10

Implementation of Heap sort

```
# include <stdio.h>
int p(int);
int left (int);
int right (int);
void heapify (int[],int,int);
void buildheap (int [],int);
void heapsort (int [],int);
void main ( )
{
int x[20],n,i;
printf ("Enter the no. of elements to be sorted
scanf ("%d", &n);
printf (" Enter the element ");
for ( i = 0; i<n; i++)
scanf ( " %d ", &x[i]);
heapsort (x,n);
printf ("sorted array is ");
for (i= 0; i<n; i++)
printf ("%d", x[i]);
}
 int p(int i)
{
 return i/2;
}
 int left (int)
```

```
{
 return 2*i+1;
}
 int right (int i)
{
 return 2*i+2;
 }
void heapify (int a [ ], int i, int n )
{
  int l, r, large, t;
  l= left (i);
  r= right (i);
  if ( l<=n-1) && (a[ ] > a [i])
  large = 1;
   else
  large =i;
  if ( (r<=n-1) && (a[r] >a [large]))
  large =r;
  if (large!=i)
 }
   t= a[i];
   a[ i] = a [large];
   a [large ] =t;
   heapify (a, large, n);
  }
}
```

```
void buildheap(int a[ ], int n)
{
  int i;
  for (i=(n-1)/2; i>=0; i--)
  heapify (a,i,n);
}
 void heapsort (int a [ ],int n)
{
   int i,m,t;
   build heap (a,n);
   m=n;
   for (i=n-1; i>=1; i--)
  {
   t=a[0];
   a[0]=a[i];
   a[i]=t;
   m=m-1;
   heapify (a,0,m);
  }
}
```

OUTPUT:

Enter the no. of the elements to be sorted: 10

Enter the ~~numbers~~ elements:

3
7
2
9
8
1
0
45
67

sorted array is:

0 1 2 3 5 7 45 867