MAHARASHTRA STATE
BOARD OF TECHNICAL EDUCATION

Certificate

This is to certify that Mr. / ~~Ms.~~ Savant Omkar Vitthal
Roll No. 45, of Third Semester of Diploma in Computer Technology of Institute, MVP S's Rajarshi Shahu Maharaj Polytechnic, Nashik (Code: 1002) has completed the term work satisfactorily in course **Data Structures Using 'C' (22317)** for the academic year 20..20.. to 20..21... as prescribed in the curriculum.

Place: Nashik
Date: 9/2/21

Enrollment No: 1910020362
Exam. Seat No: 240630

Subject Teacher **Head of the Department** **Principal**

Seal of Institution

Date : / /
Page No.:

1. Program to Perform Operations on Array

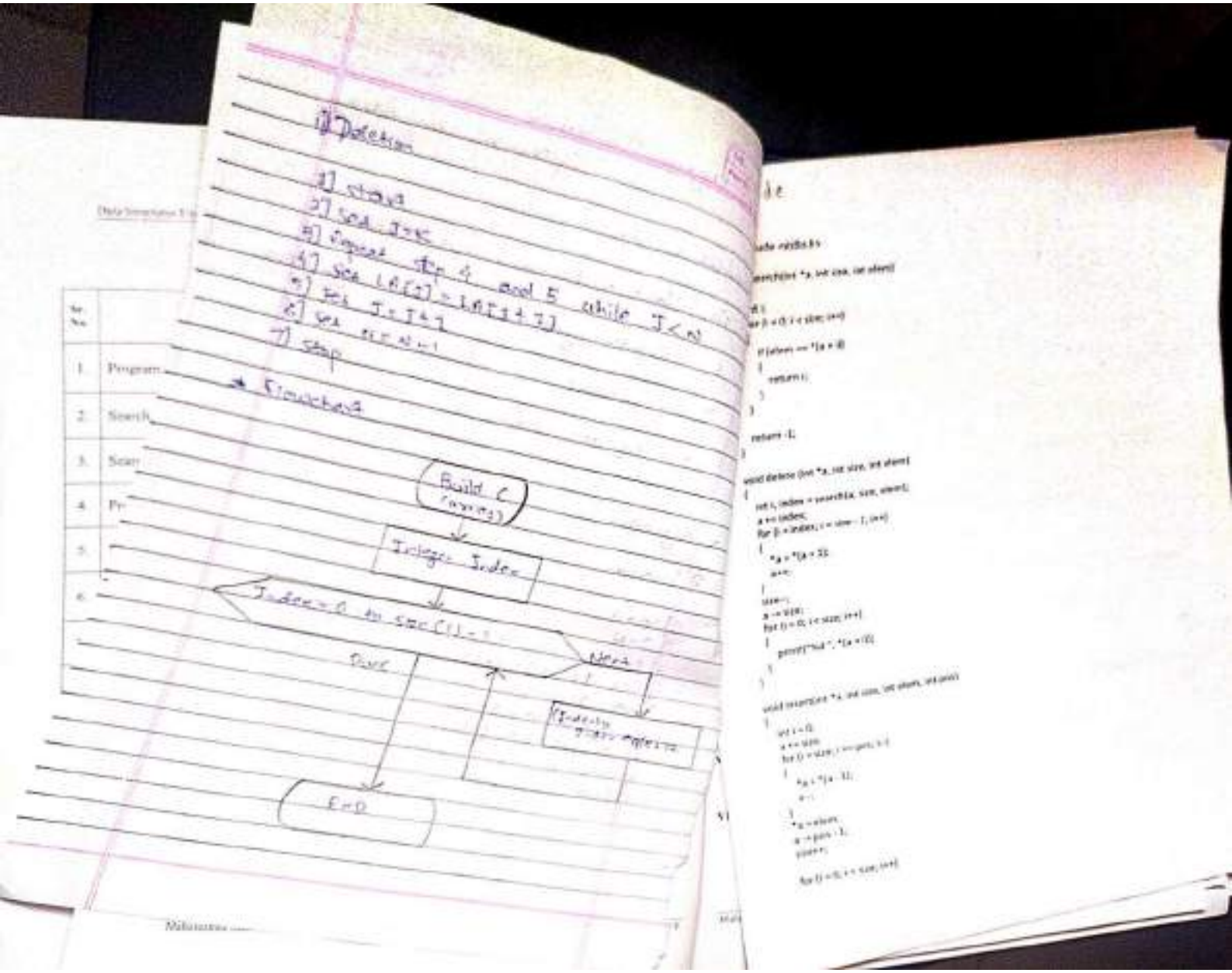
Resources Used:

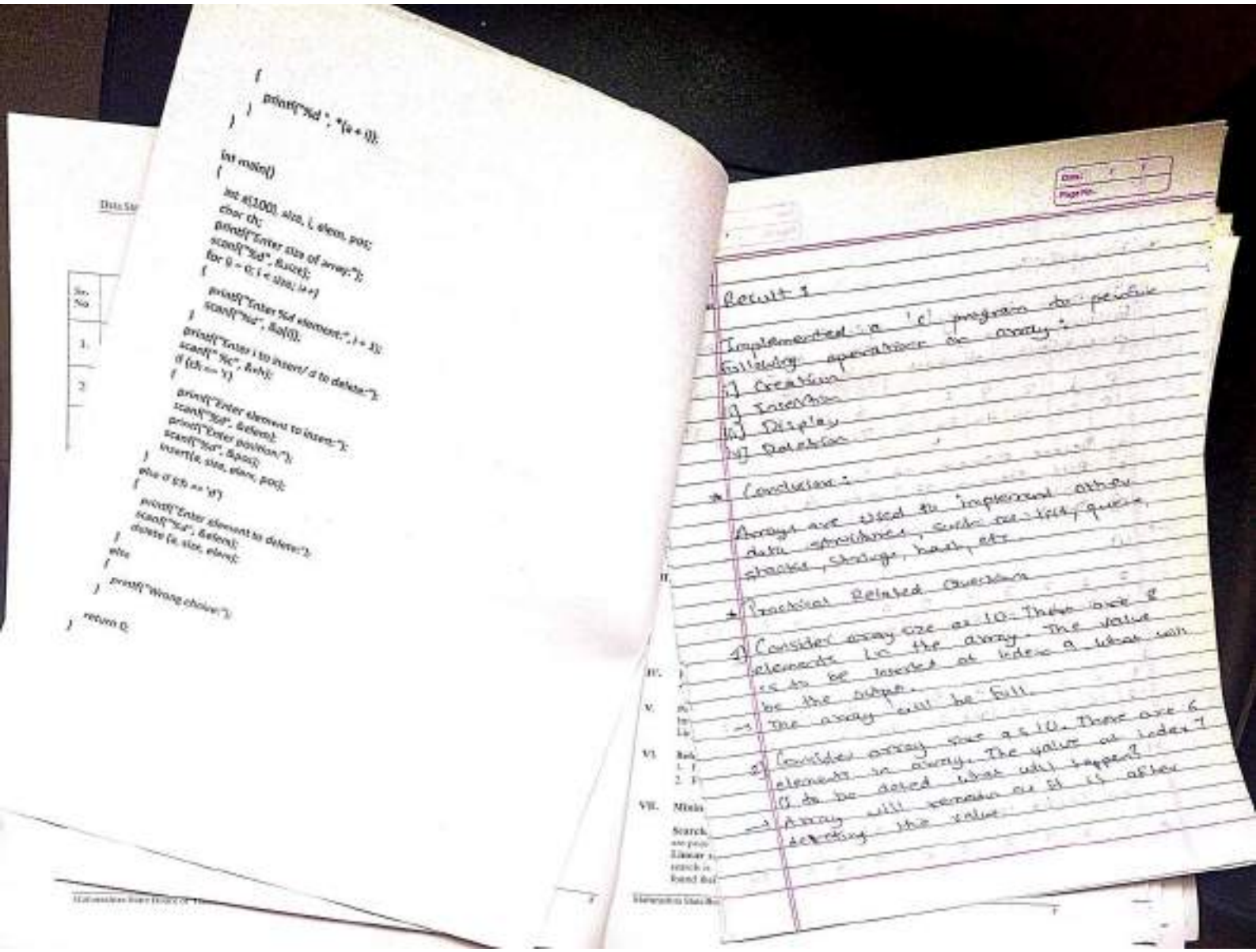
S. No.	Name of Resource	Specification
1	Computer System	Dual core, 6gb RAM
2	Software	Turbo C++
3	Any other Resource	—

* Algorithm

i) Insertion

1) Start
2) J = N
3) Repeat step 4 and 5 while J >= K
4) Set LA[J+1] = LA[J]
5) Set J = J - 1
   End of step 3 loop
6) Set LA[K] = Item
7) Reset N
8) Exit


ii) Deletion
Build C
Integer Index
Index=0 to size(1)-1
Move
E=0



```
    {
        printf("%d ", *(a + i));
    }

int main()
{
    int a[100], size, i, elem, pos;
    char ch;
    printf("Enter size of array:");
    scanf("%d", &size);
    for (i = 0; i < size; i++)
    {
        printf("Enter %d element:", i + 1);
        scanf("%d", &a[i]);
    }
    printf("Enter i to insert/ d to delete:");
    scanf(" %c", &ch);
    if (ch == 'i')
    {
        printf("Enter element to insert:");
        scanf("%d", &elem);
        printf("Enter position:");
        scanf("%d", &pos);
        insert(a, size, elem, pos);
    }
    else if (ch == 'd')
    {
        printf("Enter element to delete:");
        scanf("%d", &elem);
        delete (a, size, elem);
    }
    else
    {
        printf("Wrong choice");
    }
    return 0;
}
```

Result :-

Implemented a 'c' program to perform following operations on array :-

1] Creation
2] Insertion
3] Display
4] Deletion

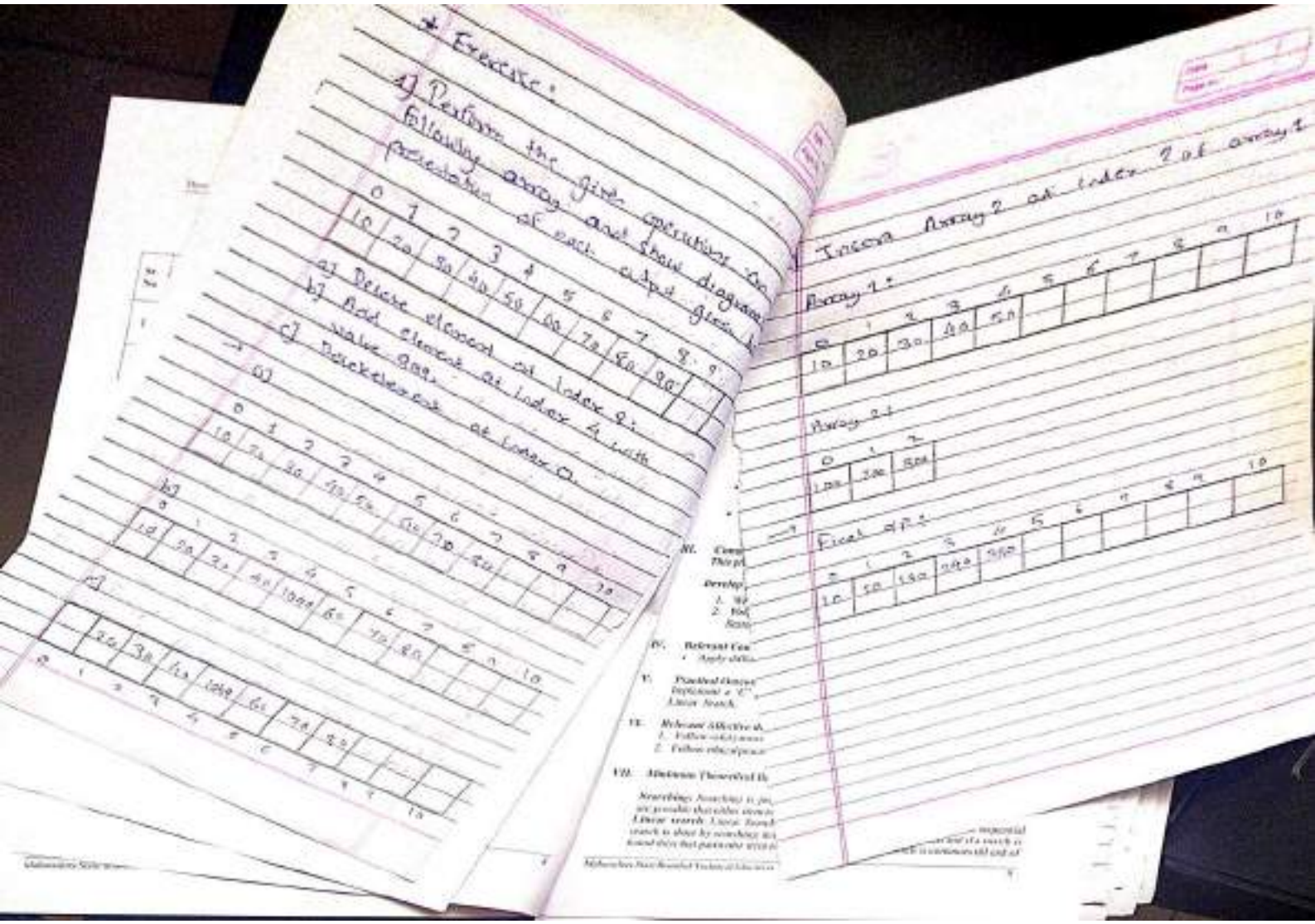
* Conclusion :-

Arrays are used to implement other data structures, such as list, queue, stacks, strings, heap, etc.

* Practical Related Questions

1] Consider array size as 10. There are 8 elements i.e. the array. The value is to be inserted at index 9 what will be the output.
→ The array will be full.

2] Consider array size as 10. There are 6 elements in array. The value at index 7 is to be deted what will happen?
→ Array will remain as it is after deleting the value.


* Exercise :
1) Perform the give operations on following array and show diagram
Traverse Array 2 at index 2 of array 1
Array 1 :
Array 2 :
Final array :

Date: / /
Page No.:

Search a Data using Linear Search.

Resources Required

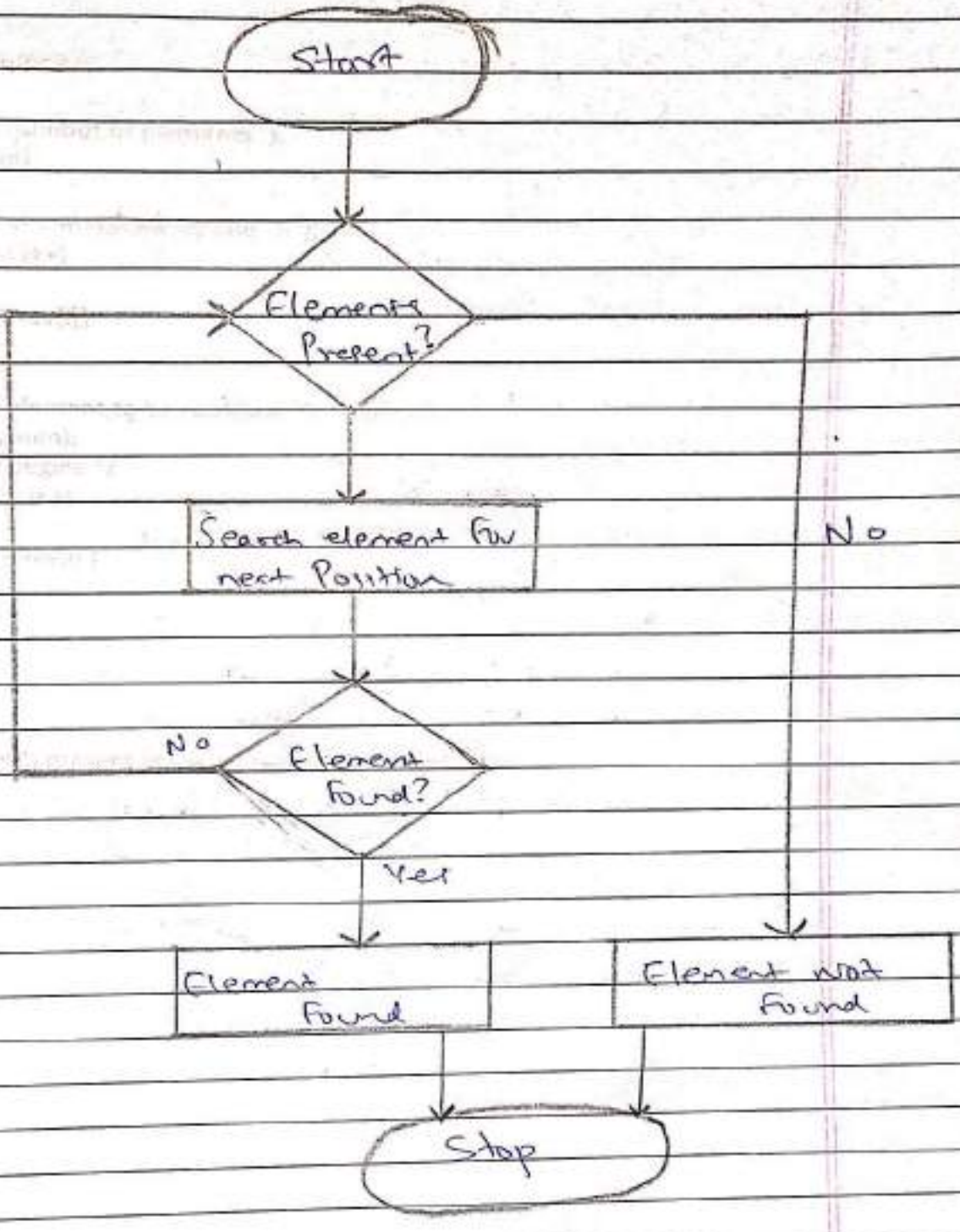
S. No.	Name of Resource	Specification
1	Computer with Specification	Dual-core, 6 GB RAM
2	Software	Turbo C
3	Any other Resource	—

* Algorithm

1] Start
2] set i to 1
3] if i > n then go to step 8
4] if A[i] = x then goto step 7
5] set i = i+1
6] Go to step 2
7] Print element x found at index i and go to step 9
8] Print element Not Found
9] Stop

Date : / /
Page No.:

Flowchart



```
<stdio.h>

ain()
num;

t i,  keynum, found = 0;

printf("Enter the number of elements ");
scanf("%d", &num);
int array[num];
printf("Enter the elements one by one \n");
for (i = 0; i < num; i++)
{
   scanf("%d", &array[i]);
}

printf("Enter the element to be searched ");
scanf("%d", &keynum);
/*  Linear search begins */
for (i = 0; i < num ; i++)
{
   if (keynum == array[i] )
   {
      found = 1;
      break;
   }
 }
 if (found == 1)
    printf("Element is present in the array at position %d",i+1);
  else
    printf("Element is not present in the array\n");
}
```

Date: / /
Page No.:

Result:

Implemented a 'C' program to search a particular data from given Array using linear search.

* Conclusion:

In order to perform some operation on specific data element, the data has to be searched in data collection, and System requires a searching method. linear search is one of them.

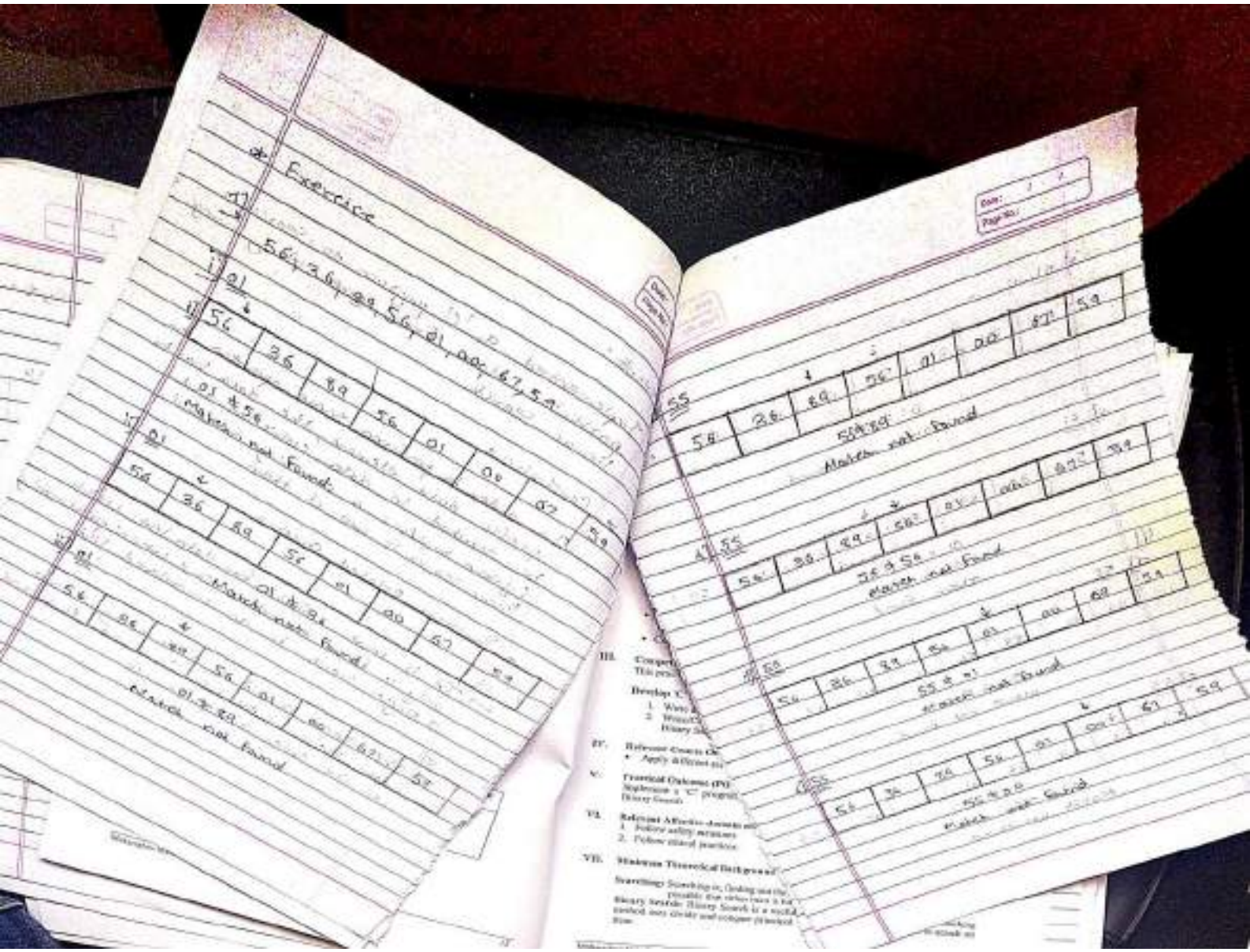
* Practical Related Question.

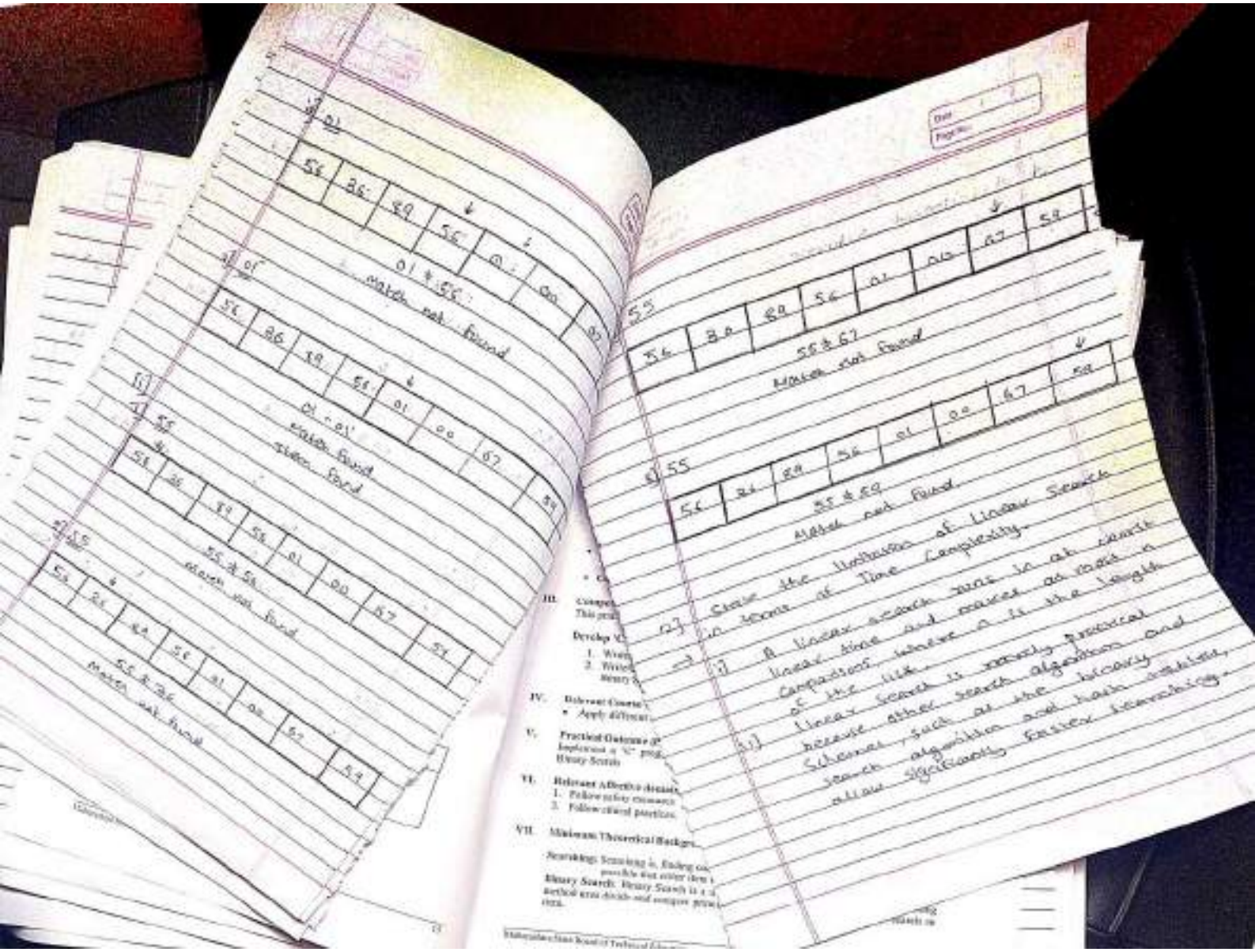
1)

→ It is most suitable for data list with only a few elements and when performing a single search in an unordered list.

2)

→ The output if list of item having same data item is to be searched is based on its order and preference.





Date: / /
Page No.:

5. Search a data using binary search

Resources Required

S.No.	Name of Resource	Specification
1	Computer System	Dual-core, 6.GB RAM
2	Software	Turbo C
3	Any other resource	—

* Algorithm

Binary_Search (A, lower_bound, upper_bound, VAL)

1] [Initialize] Set BEG = Lower_bound
END = upper_bound, POS = -1
2] Repeat step 3 and 4 while BEG <= END
3] Set MID = (BEG + END) / 2
4] If A[MID] = VAL
Set POS = MID
Print POS
Goto step 6
else if A[MID] > VAL
Set END = MID - 1
else
set beg = MID + 1

5] If pos = -1
Print "Value is not present in array"

6] Exit

* Flowchart


```
<stdio.h>
ch_search();
i, n, item, loc, beg, mid, end, i;
n()
printf("\nEnter size of an array: ");
canf("%d", &n);
printf("\nEnter elements of an array in sorted form:\n");
for(i=0; i<n; i++)
  scanf("%d", &a[i]);
printf("\nEnter ITEM to be searched: ");
scanf("%d", &item);
binary_search();
getch();
}
void binary_search()
{
  beg = 0;
  end = n-1;
  mid = (beg + end) / 2;
  while ((beg<=end) && (a[mid]!=item)
  {
    if (item < a[mid])
      end = mid - 1;
    else
      beg = mid + 1;
    mid = (beg + end) / 2;
  }
  if (a[mid] == item)
    printf("\n\nITEM found at location %d", mid+1);
  else
    printf("\n\nITEM doesn't exist");
  }
```

Date: / /
Page No.:

Result:

Implemented a C program to search a particular data from the given Array using Binary Search.

* Conclusion:

In order to perform some operation on specific data element, the data has to be searched in data collection and it requires a searching method. Binary Search is most commonly used.

* Practical Related questions.

1) Binary search is most suitable for what kind of data list?

→ Binary search is most suitable for sorted data list.

2) If mid = 4.5 then what should be the value of mid?

→

mid = low + (high - low) / 2

low = mid + 1

low = 4.5 + 1 = 5.5

high = mid * 2 = 4.5 * 2 = 9

mid = 5.5 + (9 - 5.5) / 2

mid = 7.25

Exercise

1) State the benefits and limitation of Binary Search in terms of Time Complexity.

Benefits:

1] It eliminates half of the list from further searching by using the result of each comparison.
2] It indicates whether the element being searched is before or after the current position in the list.
3] It runs better than linear search.

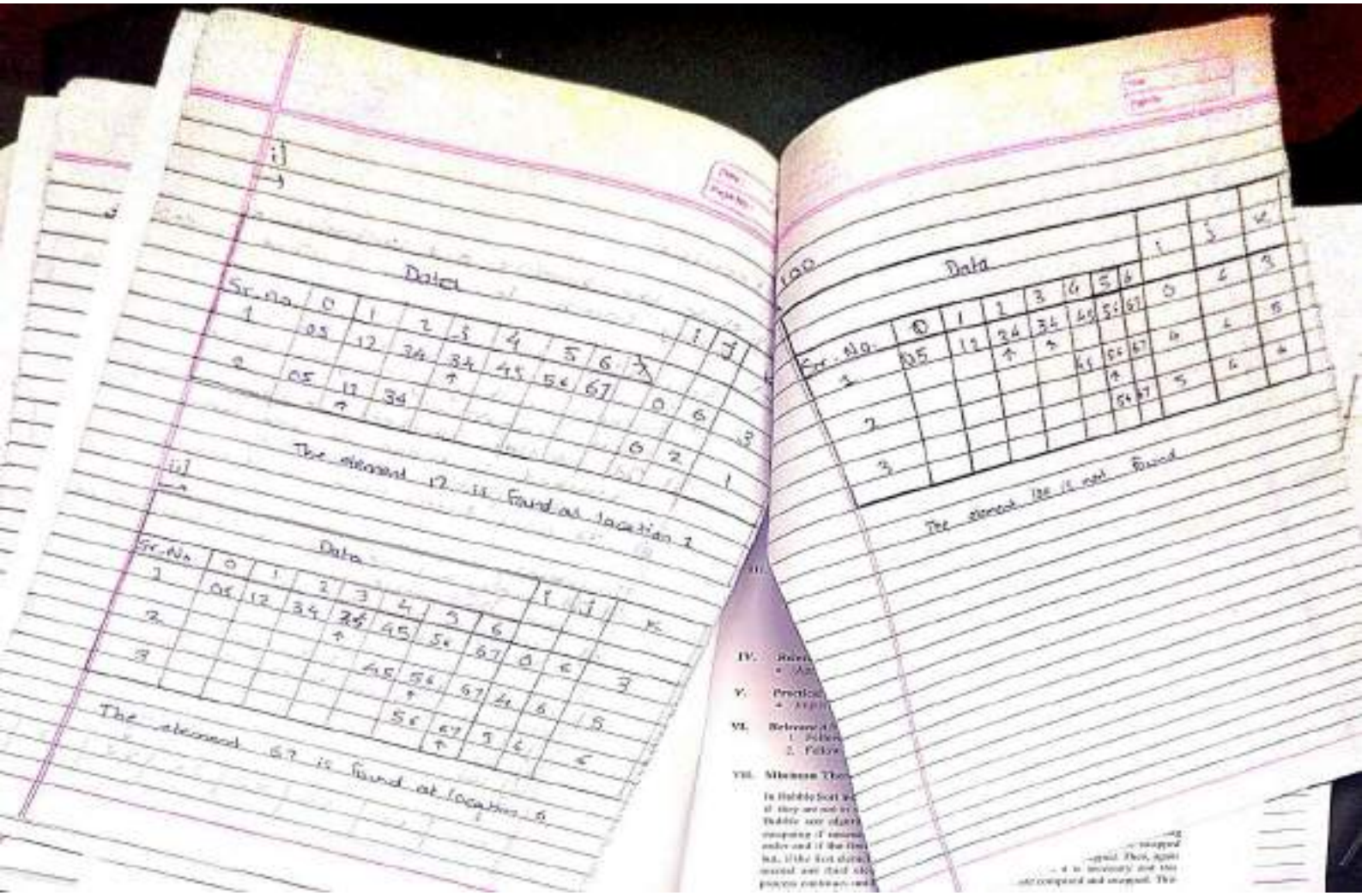
Limitation:

1] It employ recursive approach which require more stack space.
2] Caching is poor, because of its random access nature.

2] Consider following list and write down steps to perform binary search.

i] 12
ii] 67
iii] 100

Low					High	
05	12	34	34	45	56	67



Date / /
Page No.

Program to Sort an Array Using Bubble Sort.

Resources Required

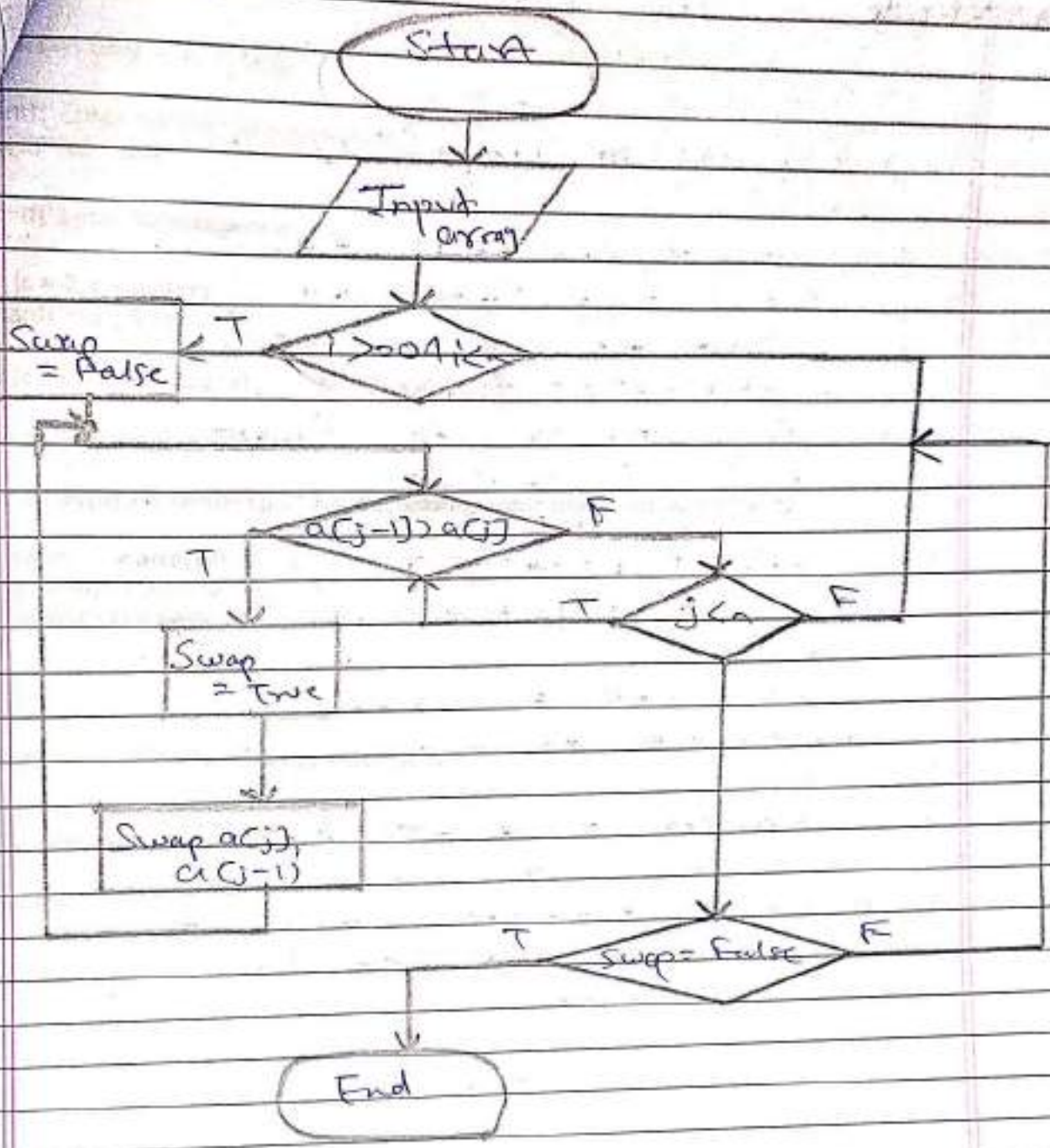
Sr. no	Name of Resource	Specifications
1	Computer System with specification	Dual - core, 6 gt RAM 500 GB HDD
2	Software	Turbo C
3	Any other	—

* Algorithm

1] Start
2] begin BubbleSort (list)
3] for all elements of list
4] if list[i] > list[i+1]
Swap (list[i], list[i+1])
5] End if
6] End for
7] return list
8] end BubbleSort
9] Stop

Date / /
Page No.:

Flowchart



```
#include <stdio.h>

main()

{
  int array[100], n, c, d, swap;

  printf("Enter number of elements\n");
  scanf("%d", &n);

  printf("Enter %d integers\n", n);

  for (c = 0; c < n; c++)
    scanf("%d", &array[c]);

  for (c = 0 ; c < n - 1; c++)
  {
    for (d = 0 ; d < n - c - 1; d++)
    {
      if (array[d] > array[d+1]) /* For decreasing order use '<' instead of '>' */
      {
        swap       = array[d];
        array[d]   = array[d+1];
        array[d+1] = swap;
      }
    }
  }

  printf("Sorted list in ascending order:\n");

  for (c = 0; c < n; c++)
    printf("%d\n", array[c]);

  return 0;
}
```

Date: / /
Page No.:

Result :

Implemented a 'C' program to sort an array using Bubble Sort method.

* Conclusion :

In order to perform some operations on specific data element, the data may to be sorted in data collection, and system requires a sorting method. One of most commonly used method is Bubble Sort.

* Practical Related Questions :

1] What is output of following code ?

```
For (i=0 ; i<=n-1, i++)
{
  for (j=0 , j<=n-1-i ; j++)
    {
      if (a[j] > a[j+1])
        {
          temp = a[j]
          a[j] = a[j+1]
          a[j+1] = temp;
        }
    }
}
```

→ Its the code for Bubble sort to perform actual Sorting

2] Find error(s) (if any) from the code:

```
for(i=0; i<n-1; i++)
{
    for(j=1; j<n-1-i; j++)
    {
        if(a[i] > a[i])
        {
            temp = a[i];
            a[j] = a[i+1];
            a[j+1] = temp;
        }
    }
}
```

→ No error's found.

* Exercise

1] Find the number of comparison required in Bubble sort of the following list having 5 number's.

→ 10 20 30 40 50

0 Comparisons required, list is already Sorted.

Date: / /
Page No:

Sort the given array in ascending order using Bubble sort and show steps.

1000, -20, 300, 150, 50

1]
1000 -20 300 150 50
-20 1000 300 150 50
-20 300 1000 150 50
-20 300 150 1000 50
-20 300 150 50 1000

2]
-20 300 150 50 1000
-20 300 150 50 1000
-20 150 300 50 1000
-20 150 50 300 1000

3]
-20 150 50 300 1000
-20 150 50 300 1000
-20 50 150 300 1000

4]
-20 50 150 300 1000

List has been sorted.

8] Sort the given array using ascending order using bubble sort and show steps.
500, 120, 2000, -210, 89

→

I]
500 120 2000 -210 89
120 500 2000 -210 89
120 500 -210 2000 89
120 500 -210 89 2000
120 500 -210 89 2000

II]
120 500 -210 89 2000
120 -210 500 89 2000
120 -210 89 500 2000
120 -210 89 500 2000

III]
~~-210 120~~ 120 -210 89 500 2000
-210 120 89 500 2000
-210 89 120 500 2000
-210 89 120 500 2000

The list is sorted.

Program to Sort an array Using Selection Sort.

Resources Used:

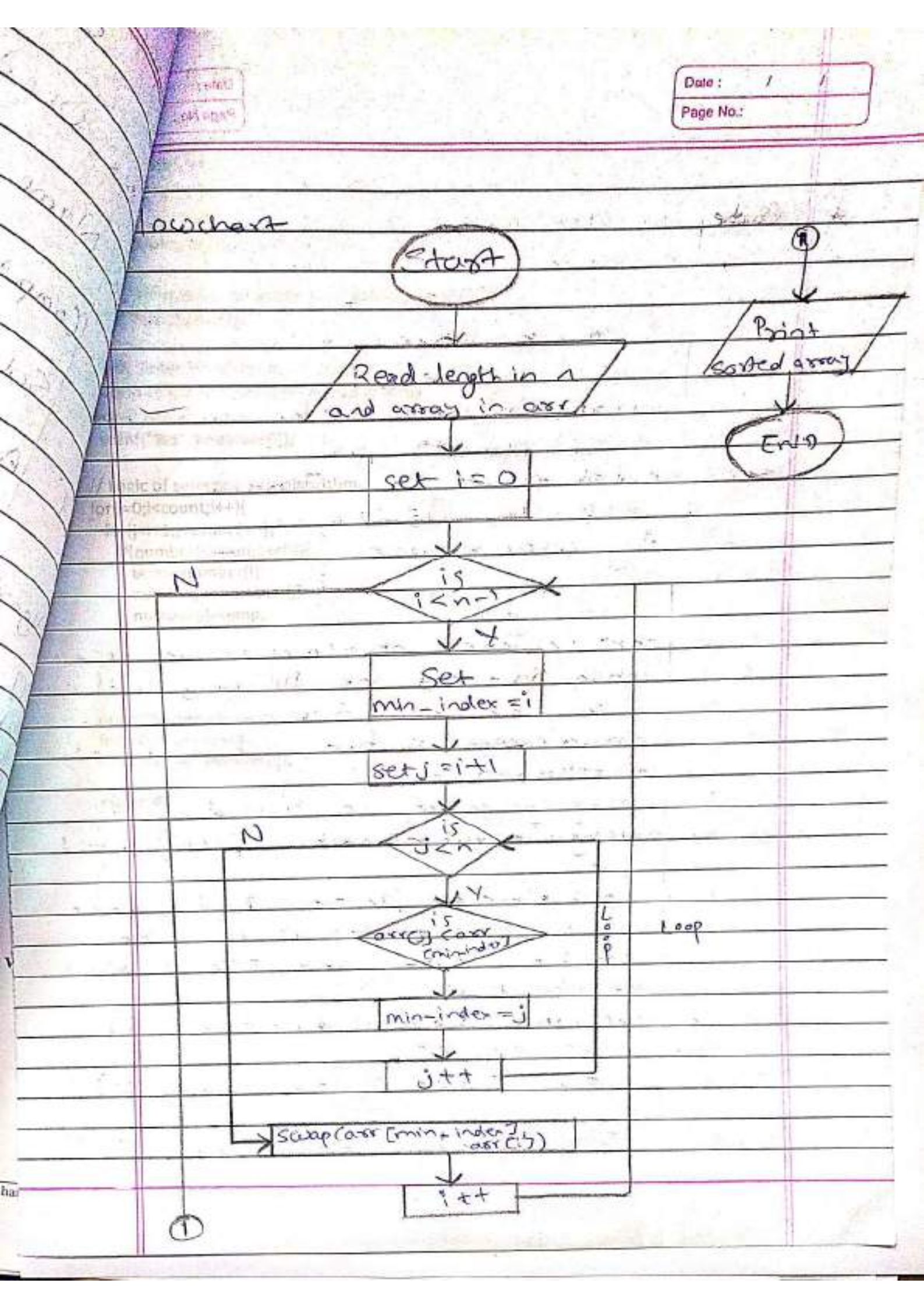
Sr. No.	Name of Resource	Specification
1	Computer System	Dual-core, 6GB RAM
2	Software	Turbo C
3	Any other resource	—

* Algorithm:

1] Start
2] Set MIN to location 0
3] Search the minimum element in the list.
4] Swap with value at location MIN
5] Increment MIN to point to next element.
6] Repeat until list is sorted
7] Stop

Date : / /
Page No.:

owchart



```
dio.h>


ount, temp, number[25];

"How many numbers u are going to enter?: ");
("%d",&count);

ntf("Enter %d elements: ", count);
Loop to get the elements stored in array
r(i=0;i<count;i++)
scanf("%d",&number[i]);

// Logic of selection sort algorithm
for(i=0;i<count;i++){
  for(j=i+1;j<count;j++){
    if(number[i]>number[j]){
      temp=number[i];
      number[i]=number[j];
      number[j]=temp;
    }
  }
}

  printf("Sorted elements: ");
  for(i=0;i<count;i++)
    printf(" %d",number[i]);

    return 0;
  }
```

Date
Page No.

Result :

Implemented a 'C' program to sort an array using Selection sort method.

★ Conclusion :

In order to perform some operation on specific data element, the data needs to be sorted. One of the most commonly used method is Selection sort.

★ Practical Related Questions :

1] A Selection Sort compares adjacent elements, and swap them if they are in wrong order.
a. True c. Depend on element
b. False d. None of these.
→ a. True.

2] For each from 1 to n-1, there are exchanges for Selection Sort.
a. 1 c. n
b. n-1 d. none of these
→ a. 1

3] What is the output of selection sort 2nd iteration given numbers:
20 12 100 15 2

a. 2 15 12 10 20
b. 2 10 12 15 20
c. 2 12 10 15 20
d. None of above
→ c. 2 12 10 15 20

* Exercise

1] Find the number of comparisons in Selection Sort of the following list
10 20 30 40 50
→ 14 Comparisons

2] Sort the given array in ascending order using Insertion Sort.
→ 1000, -20, 300, 140, 50

1000 -20 300 140 50
-20 1000 300 140 50 I]
-20 50 300 140 1000 II]
-20 50 140 300 1000 III]

Array has been Sorted.

Sort the given array in ascending order using Insertion Sort.

→ 500 120 2000 -210 30
(I) 500 120 2000 -210 30
(II) -210 120 2000 500 30
(III) -210 30 2000 500 120
(IV) -210 30 120 500 2000
The array has been sorted.

Date: / /
Page No.:

5. Program to sort an array using insertion sort.

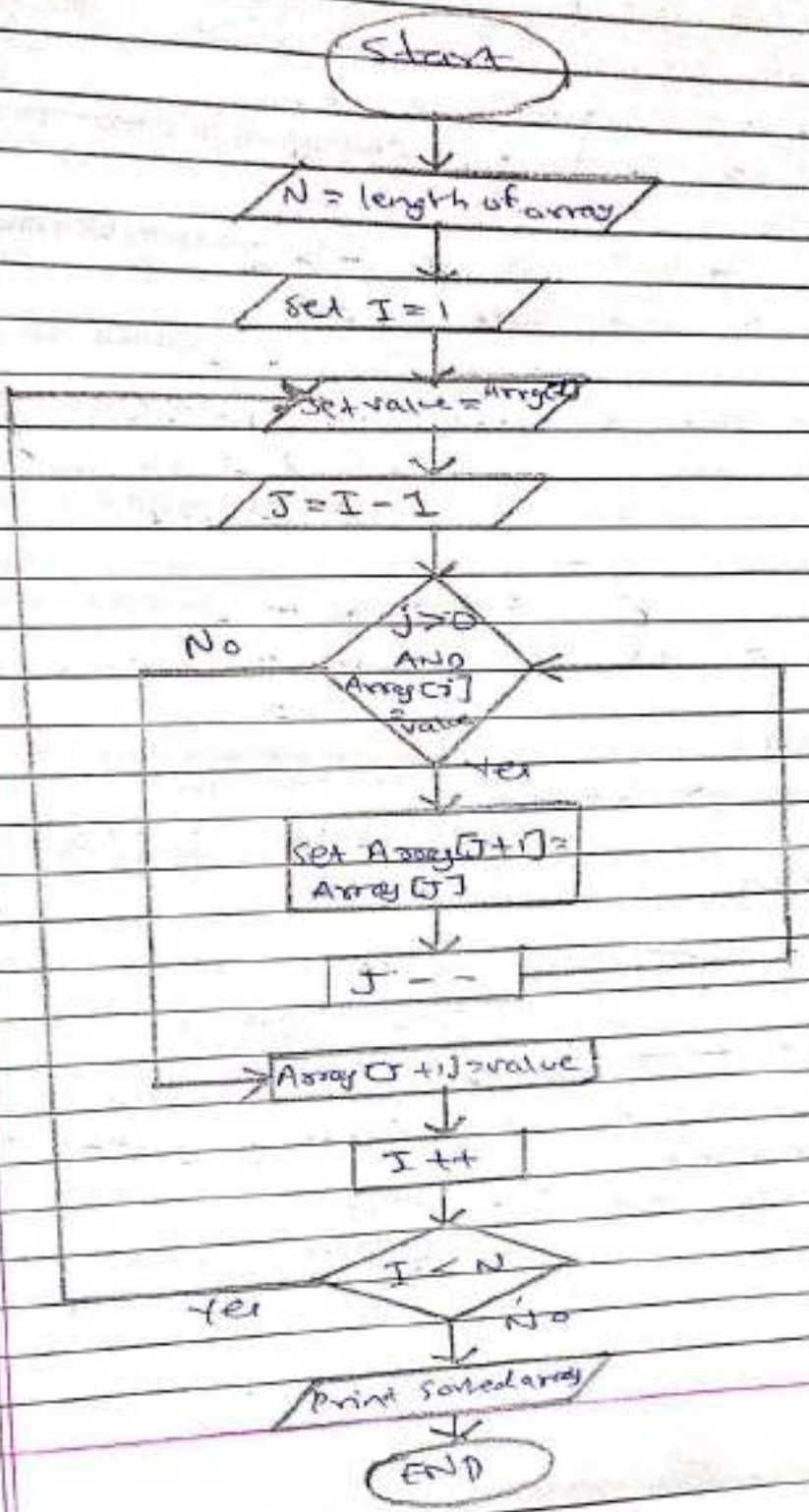
Resources Used:

Sr. No.	Name of Resource	Specification
1	Computer System	Dual-core, 6gb RAM
2	Software	Turbo C
3	Any other resource	—

* Algorithm:

1] Start
2] If the element is first, and is already sorted, return 1;
3] Pick next element
4] Compare with all elements in the sorted sub-list.
5] Shift all elements in the sorted sub-list that is greater than the value to be sorted.
6] Insert the value
7] Repeat until list is sorted
8] Stop

Flowchart


Start
N = length of array
Set I = 1
Set value = Array[I]
J = I - 1
j >= 0 AND Array[j] > value
No
Yes
Set Array[J+1] = Array[J]
J - -
Array[J+1] = value
I ++
I < N
Yes
No
Print sorted array
END

```
de <stdio.h>
ain()

nt n, i, j, temp;
nt arr[64];

printf("Enter number of elements\n");
scanf("%d", &n);

printf("Enter %d integers\n", n);
for (i = 0; i < n; i++)
{
   scanf("%d", &arr[i]);
}
for (i = 1 ; i <= n - 1; i++)
{
            j = i;
      while ( j > 0 && arr[j-1] > arr[j])
      {
         temp    = arr[j];
         arr[j]  = arr[j-1];
         arr[j-1] = temp;
         j--;
      }
}
printf("Sorted list in ascending order:\n");
for (i = 0; i <= n - 1; i++)
{
   printf("%d\n", arr[i]);
}
return 0;
}
```

Result:

Implemented a 'C' program to sort an array using Insertion Sort method.

* Conclusion:

In order to perform some operation on data the data has to be sorted. One of the most commonly used method is Insertion Sort for sorting data in list.

* Practical Related Questions:

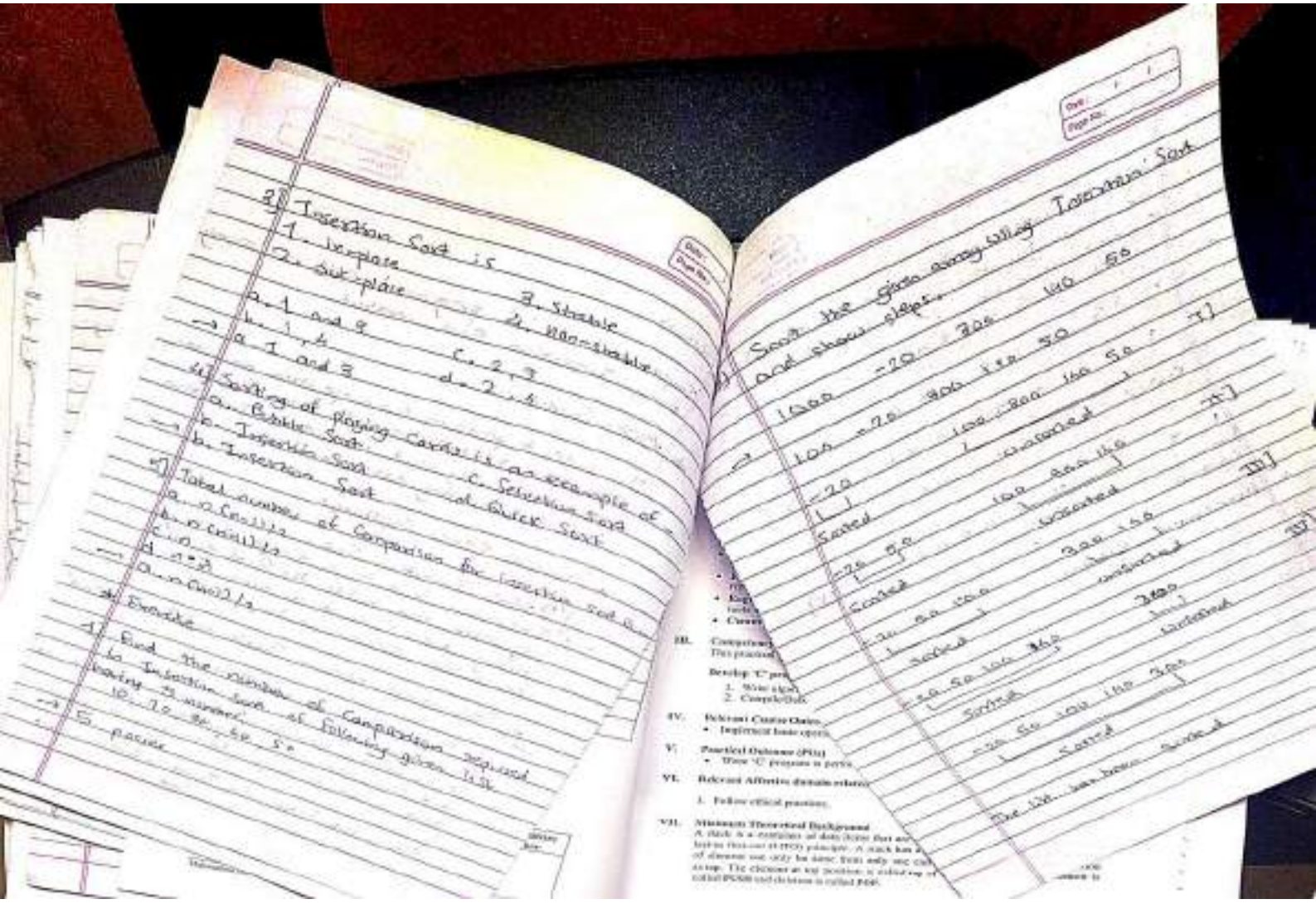
1) Define Insertion Sort.

→ Insertion Sort is a simple sorting algorithm that builds the final sorted array one item at a time. It is much less efficient on large lists.

2) Differentiate between Bubble and Insertion Sort.

→

Bubble Sort	Insertion Sort
Check neighbour elements and swaps them	Transfers an element at a time to sort
More number of Swaps	Less number of Swaps
Slower than insertion Sort	Twices as fast as bubble Sort



8] What is output of Insertion sort after the 2nd iteration given the following sequence of numbers:

7 3 5 1 9 8 4 6

→

7 3 5 1 9 8 4 6

1 7 3 5 9 8 4 6 I]

sorted unsorted

1 3 7 5 9 8 4 6 II]

sorted unsorted

* Assesment Scheme

1. Perform Push and Pop Operation on stack

Date / / Page No.

* Resources Used :

Sr no.	Name of Resource	Specifications
1	Computer System	Dual core, 6gb RAM
2	Software	Turbo C
3	Any other	—

* Algorithm

i) PUSH

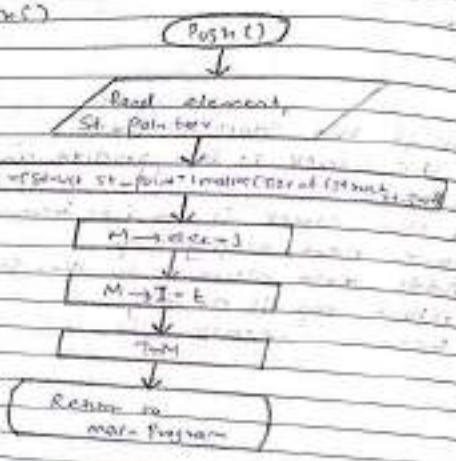
1] Check if the stack is full
2] If the stack is full, produces an error & exit.
3] If the stack is not full, increment top to point next empty step.
4] Adds data element to the stack location where top is pointing
5] Return success

1] Pop

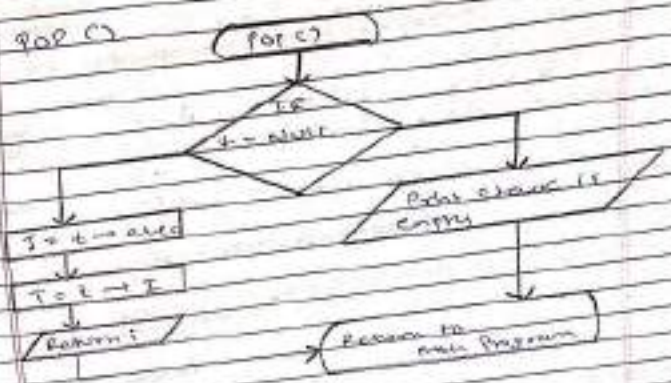
1] Check if stack is empty
2] If the stack is empty, produce an exit.
3] If the stack is not empty, access the data element at which top is pointing
4] Decrease the value of top by 1
5] Return success

* Flowchart

1] Push()



2] Pop()



CodC

```
ude<stdio.h>
lude<process.h>
clude<stdlib.h>

define MAX 5        //Maximum number of elements that can be stored

int top=-1,stack[MAX];
void push();
void pop();
void display();

void main()
{
        int ch;

        while(1)  //Infinite loop, will end when choice will be 4
        {
                printf("\n*** Stack Menu ***");
                printf("\n\n1.Push\n2.Pop\n3.Display\n4.Exit");
                printf("\n\nEnter your choice(1-4):");
                scanf("%d",&ch);

                switch(ch)
                {
                        case 1: push();
                                break;
                        case 2: pop();
                                break;
                        case 3: display();
                                break;
                        case 4: exit(0);
                        default: printf("\nWrong Choice!!");
                }
        }
}

void push()
{
        int val;

        if(top==MAX-1)
        {
                printf("\nStack is full!!");
        }
        else
        {
```

```
        printf("\nEnter element to push: ");
        scanf("%d",&val);
        top=top+1;
        stack[top]=val;
    }
}
void pop()
{
    if(top==-1)
    {
        printf("\nStack is empty\n");
    }
    else
    {
        printf("\nDeleted element is %d",stack[top]);
        top=top-1;
    }
}
void display()
{
    int i;
    if(top==-1)
    {
        printf("\nStack is empty\n");
    }
    else
    {
        printf("\nStack is...\n");
        for(i=top;i>=0;i--)
            printf("%d\n",stack[i]);
    }
}
```

Date:
Page No:

Result :-

Implemented basic operations to create 'C' program to perform PUSH and POP operations on stack array.

→ Conclusion :-

In order to perform insertion and deletion of data item from given data list based on Last In First Out (LIFO) suitable data structure is required. So Stack is structure that follows LIFO order.

→ Practical Related Questions

1) Draw stack after POP operation using given diagram.

→ POP ()

Ans :-

2] Draw stack after Push operation given stack

Push(45)

Ans :

30
60
15

45
30
60
15

2] Write a Top value after Push operation.

Push(45)

Ans :

30
60
15

45	← Top
30	
60	
15	

Exercise Related Questions

7-1

Date: / /
Page No:

2] Implement a 'C' program for stack size 8 for
push(10), push(20), pop, push(10), push(20), pop, pop, pop, push(20), pop and draw final output.

→

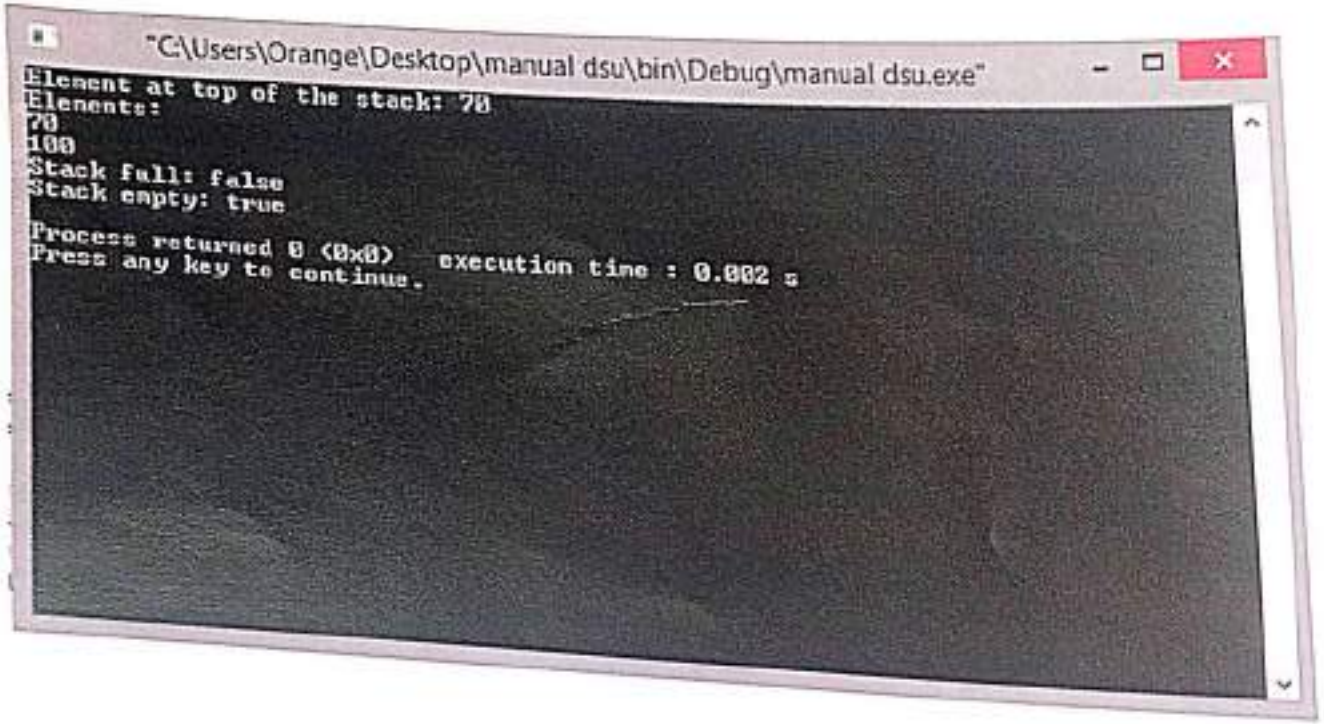
```
#include <stdio.h>
int MAXSIZE = 8;
int stack[8];
int top = -1;

int isempty() {
   if(top == -1)
      return 1;
   else
      return 0;
}

int isfull() {
   if(top == MAXSIZE)
      return 1;
   else
      return 0;
}
int peek() {
   return stack[top];
}
```

```
int pop(){
    int data;

    if(!isempty()){
        data = stack[top];
        top = top-1;
        return data;
    }else{
        printf("stack is empty\n");
    }
}
int push(int data){

    if(!isfull()){
        top = top+1;
        stack[top]=data;
    }else{
        printf("can't since is full\n");
    }
}

int main(){
    push(10);
    push(20);
    pop();
    push(30);
    push(40);
    pop();
    pop();
    pop();
```

```
    push(20);
    pop();
    printf("Element at top of stack : %d\n",
            peek());
    printf("Elements:\n");
    while(!isempty()){
        int data= pop();
        printf("%d\n",data);
    }
    return 0;
}
```

Final output :

Empty stack



1 - 3

Page No.:

stack of

Date: / /
Page No.:

8. Perform insert and delete operations on linear queue using array.

Resources Used:

Sr.No	Name of Resource	Specifications
1	Computer System	Dual-core, 6 gb RAM
2	Software	Turbo C
3	Any other	—

* Algorithm

i) Insertion

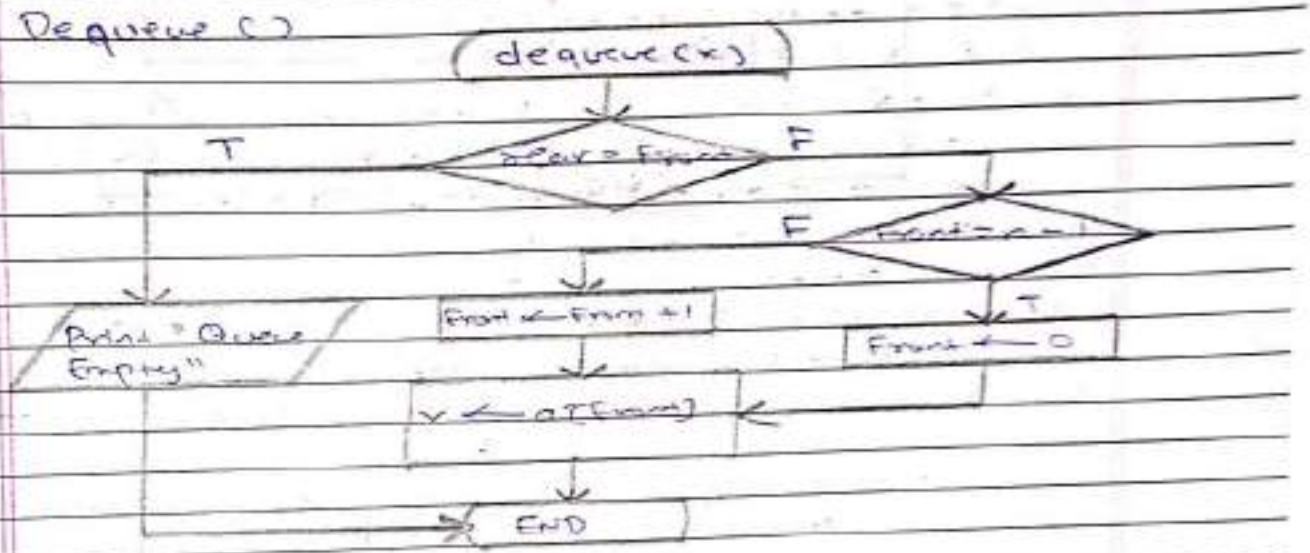
1] IF Rear = Max - 1
write Overflow
Go to step
[End of IF]
2] IF Front = -1 and Rear = -1
Set Front = Rear = 0
else
Set Rear = Rear + 1
[End of IF]
3] Set Queue [Rear] = Num
4] Exit

3] Deletion

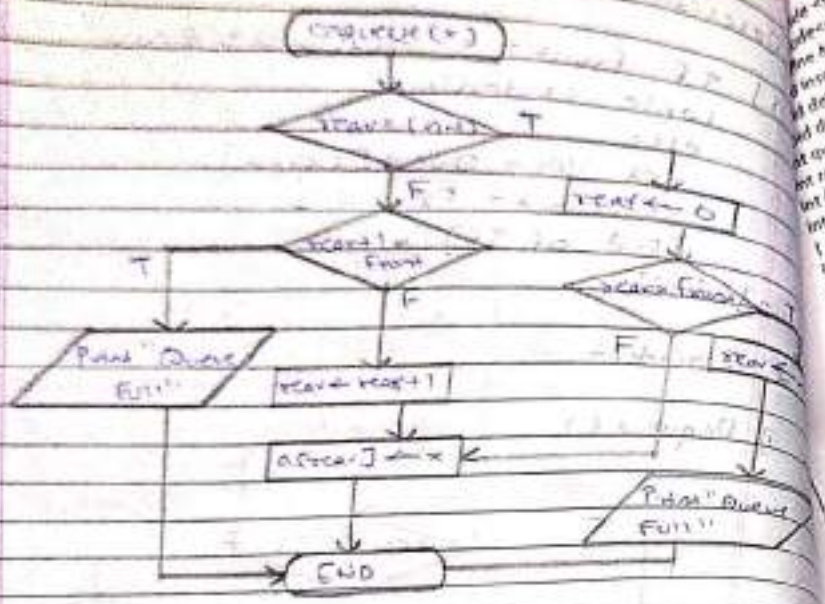
1] If front = -1 or front > Rear
Write Underflow
else
set VAL = QUEUE[front]
set front = front + 1
[End of IF]
2] Exit

* Flowchart

1] Dequeue ()



2] Enqueue()



Code

```
de <stdio.h>
de<stdlib.h>
ine MAX 50
d insert();
d delete();
id display();
t queue_array[MAX];
nt rear = - 1;
nt front = - 1;
int main()
{
 int choice;
 while (1)
 {
 printf("1.Insert element to queue n");
 printf("2.Delete element from queue n");
 printf("3.Display all elements of queue n");
 printf("4.Quit n");
 printf("Enter your choice : ");
 scanf("%d", &choice);
 switch(choice)
  {
  case 1:
  insert();
  break;
  case 2:
  delete();
  break;
  case 3:
  display();
  break;
  case 4:
  exit(1);
  default:
  printf("Wrong choice n");
  }
 }
 void insert()
 {
 int item;
 if(rear == MAX - 1)
 printf(" Queue Overflow n");
 else
 {
 if(front == - 1)
 front = 0;
```

```
        printf("Insert the element in queue : ");
        scanf("%d", &item);
        rear = rear + 1;
        queue_array[rear] = item;
    }
    void delete()
    {
    if(front == -1 || front > rear)
    {
    printf("Queue Underflow \n");
    return;
    }
    else
    {
    printf("Element deleted from queue is : %d\n", queue_array[front]);
    front = front + 1;
    }
    void display()
    {
    int i;
    if(front == -1)
    printf("Queue is empty \n");
    else
    {
    printf("Queue is : \n");
    for(i = front; i <= rear; i++)
    printf("%d ", queue_array[i]);
    printf("\n");
    }
```

Date: / /
Page No.:

Result :

Implemented basic C program to perform Insert and Delete operations on Linear queue using Array.

Conclusion :

Linear Queue is collection of elements that follows FIFO order which keep track of two indices, front and rear, one for front and another for rear delete

* Practical Related Questions

1) Write a front and rear values of linear queue.

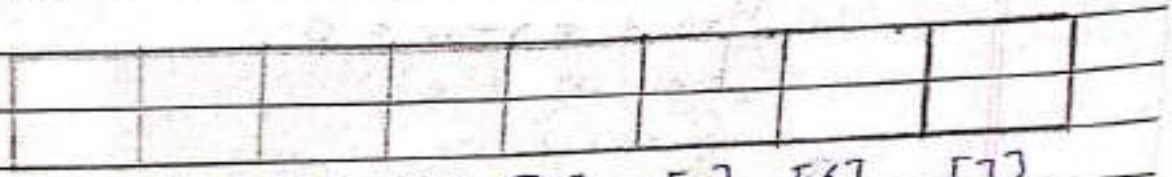
Enqueue

Dequeue

5	6	5	8	9	10	6	7

→ Front = 0
Rear = 7

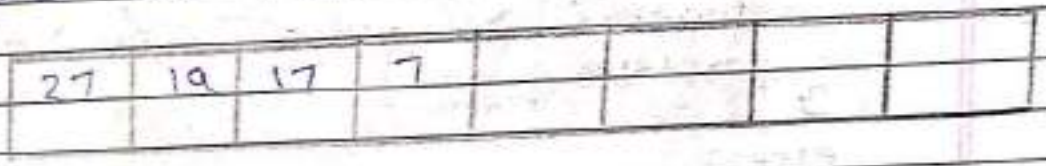
1] Sketch the front and rear in empty queue.



→

Front = -1
Rear = -1

2] Sketch the front and rear in this queue.



→

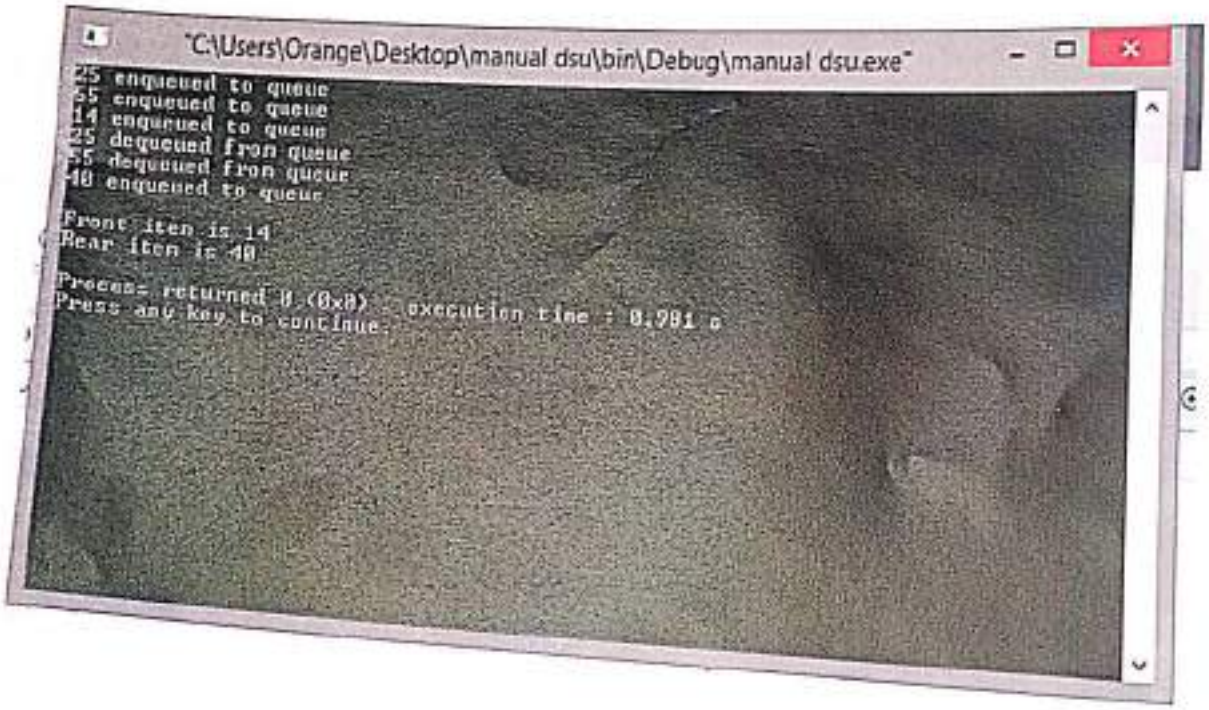
Front = 0
Rear = 3

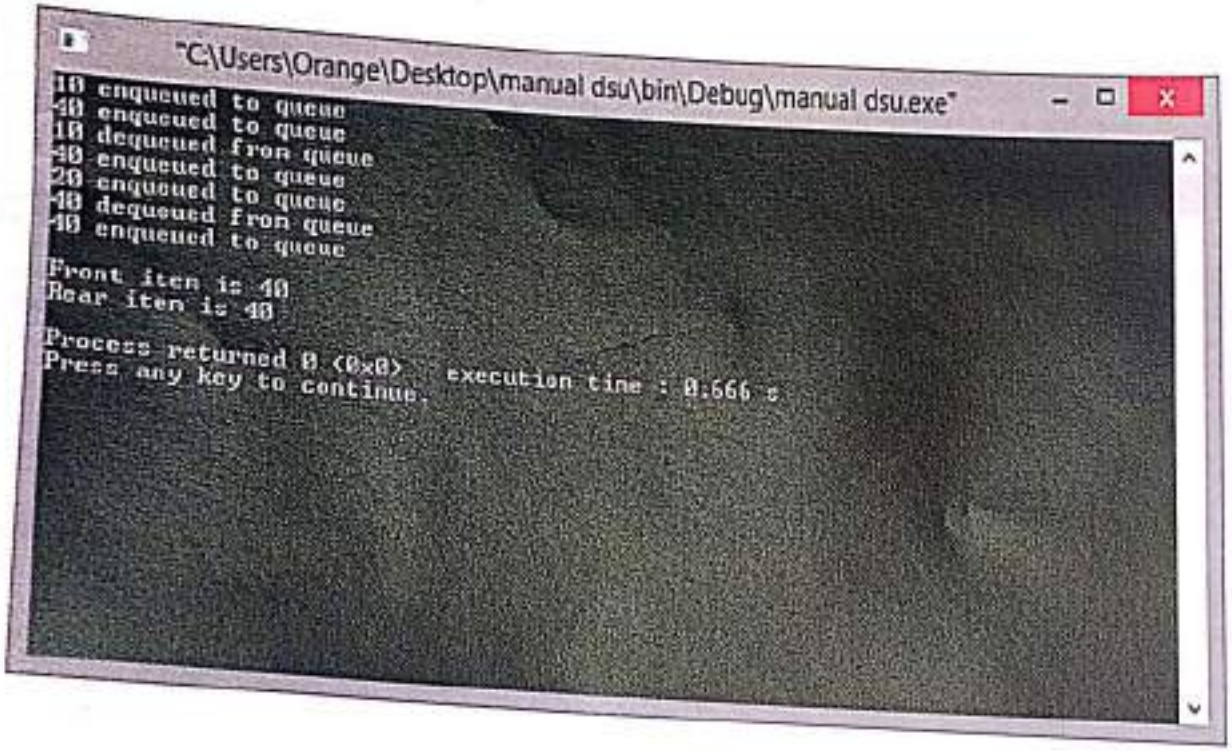
* Exercise

1] Implement a C program to linear Queue in array size 5 with this Operation Insert (25), Insert (55), Insert (14), delete, delete, Insert (40).

→

```
#include <stdio.h>
Struct Queue {
    int front, rear, capacity;
    int * queue;
```


"C:\Users\Orange\Desktop\manual dsu\bin\Debug\manual dsu.exe"
25 enqueued to queue
55 enqueued to queue
14 enqueued to queue
25 dequeued from queue
55 dequeued from queue
40 enqueued to queue
Front item is 14
Rear item is 40
Process returned 0 (0x0) execution time : 0.701 s
Press any key to continue.


"C:\Users\Orange\Desktop\manual dsu\bin\Debug\manual dsu.exe"
10 enqueued to queue
40 enqueued to queue
10 dequeued from queue
40 enqueued to queue
20 enqueued to queue
40 dequeued from queue
40 enqueued to queue
Front iten is 40
Rear iten is 40
Process returned 0 (0x0) execution time : 0.666 s
Press any key to continue.

```
de<stdio.h>
he size 2
ueue[size],front=-1,rear=-1;
d insert(int item){

if(rear==size-1){
printf("Queue is full \n");

}
else{
  if(front==-1){
      front=0;
  }
  rear=rear+1;
  queue[rear]=item;
  printf("Element Inserted %d\n",item);
}
}
void delete(){
  if(front==-1){
    printf("Queue is Empty\n");
  }
  else{
    printf("Element is deleted %d\n",queue[front]);
    front=front+1;
    if(front>rear){
      front=-1;
      rear=-1;
    }
  }
}
void display(){
  if(rear==-1)
    printf("Queue is empty\n");
  else{
    int i;
    printf("Queue:");
    for(i=front;i<=rear;i++)
      printf("%d",queue[i]);

  }
}
int main()
{
  display();
  insert(1);
  insert(2);
  delete();
```

9. Perform insert and delete operations on Circular Queue using array.

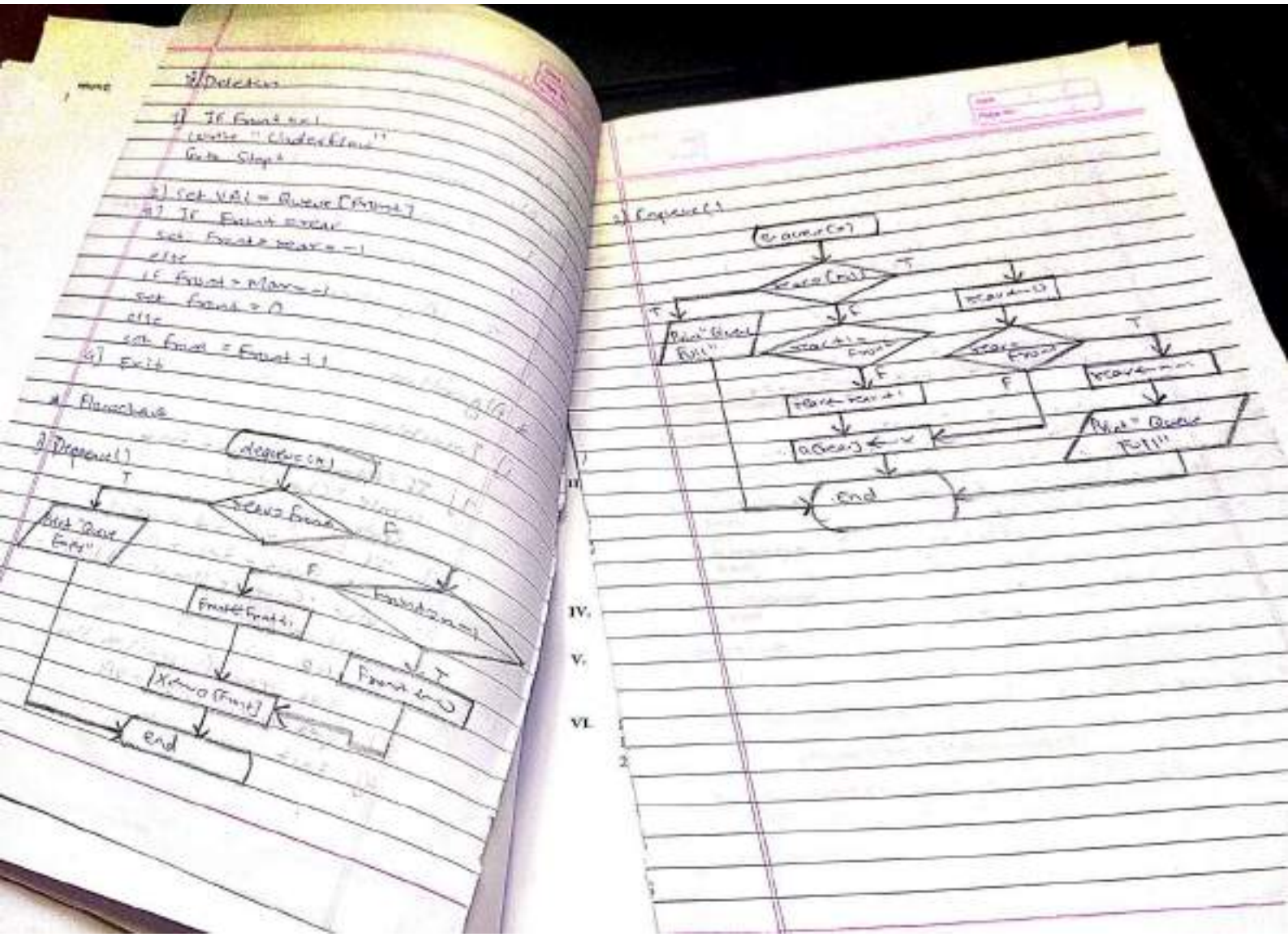
* Resources Used :

Sr. No.	Name of Resource	Specification
1	Computer System	Dual core, 6gb RAM
2	Software	Turbo C
3	Any other	—

* Algorithm.

i] Insertion

1] If (rear+1) % Max = front
   Write "Overflow"
   Goto Step 4
2] If front = -1 and rear = -1
   set front = Rear = 0
   else if rear = Max - 1 and front != 0
   set rear = 0
   else
   set rear = (rear+1) % Max
3] set queue [rear] = VAL
4] Exit


Deletion
1] If Front = -1
Write "Underflow"
Goto Step 4
2] Set VAL = Queue[Front]
3] If Front = Rear
Set Front = Rear = -1
else
if Front = Max - 1
Set Front = 0
else
Set Front = Front + 1
4] Exit
Flowchart
2] Dequeue()
dequeue()
Front = -1
Print "Queue Empty"
Front = Front + 1
Front = Max - 1
Front = 0
X = a[Front]
end
2] Enqueue()
enqueue()
Rear = Max - 1
Print "Queue Full"
Rear = Rear + 1
a[Rear] = x
Rear = 0
end

Code

```
clude <stdio.h>
efine size 5

void insertq(int[], int);
void deleteq(int[]);
void display(int[]);

int front = - 1;
int rear = - 1;

int main()
{
  int n, ch;
  int queue[size];
  do
  {
    printf("\n\n Circular Queue:\n1. Insert \n2. Delete\n3. Display\n0. Exit");
    printf("\nEnter Choice 0-3? : ");
    scanf("%d", &ch);
    switch (ch)
    {
      case 1:
        printf("\nEnter number: ");
        scanf("%d", &n);
        insertq(queue, n);
        break;
      case 2:
        deleteq(queue);
        break;
      case 3:
        display(queue);
        break;
    }
  }while (ch != 0);
}


void insertq(int queue[], int item)
{
  if ((front == 0 && rear == size - 1) || (front == rear + 1))
  {
    printf("queue is full");
    return;
  }
  else if (rear == - 1)
  {
    rear++;
```

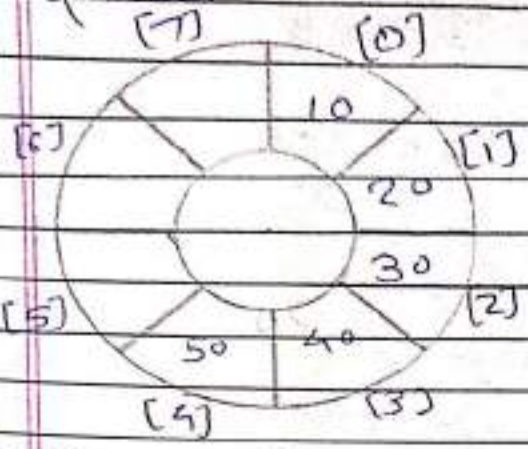
```
    return 0;
}
                front++;
            }
            else if (rear == size - 1 && front > 0)
            {
                rear = 0;
            }
            else
            {
                rear++;
            }
            queue[rear] = item;
        }

        void display(int queue[])
        {
            int i;
            printf("\n");
            if (front > rear)
            {
                for (i = front; i < size; i++)
                {
                    printf("%d ", queue[i]);
                }
                for (i = 0; i <= rear; i++)
                    printf("%d ", queue[i]);
            }
            else
            {
                for (i = front; i <= rear; i++)
                    printf("%d ", queue[i]);
            }
        }

        void deleteq(int queue[])
        {
            if (front == - 1)
            {
                printf("Queue is empty ");
            }
            else if (front == rear)
            {
                printf("\n %d deleted", queue[front]);
                front = - 1;
                rear = - 1;
            }
            else
            {
```

Date / /
Page No.

* Practical Related Questions.

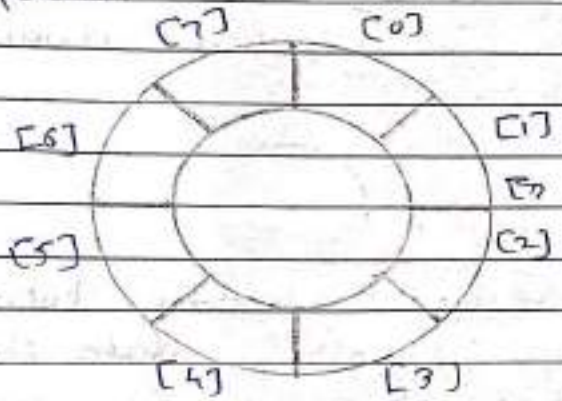
1) Write a Front and rear values of circular queue.

–



→ Front = 0
rear = 4

2) Show the Front and rear in empty circular queue.



→ Front = -1
Max = -1

9] Show the Circular Queue working with value of array (size 5).
15, 20, 30, 10, 5

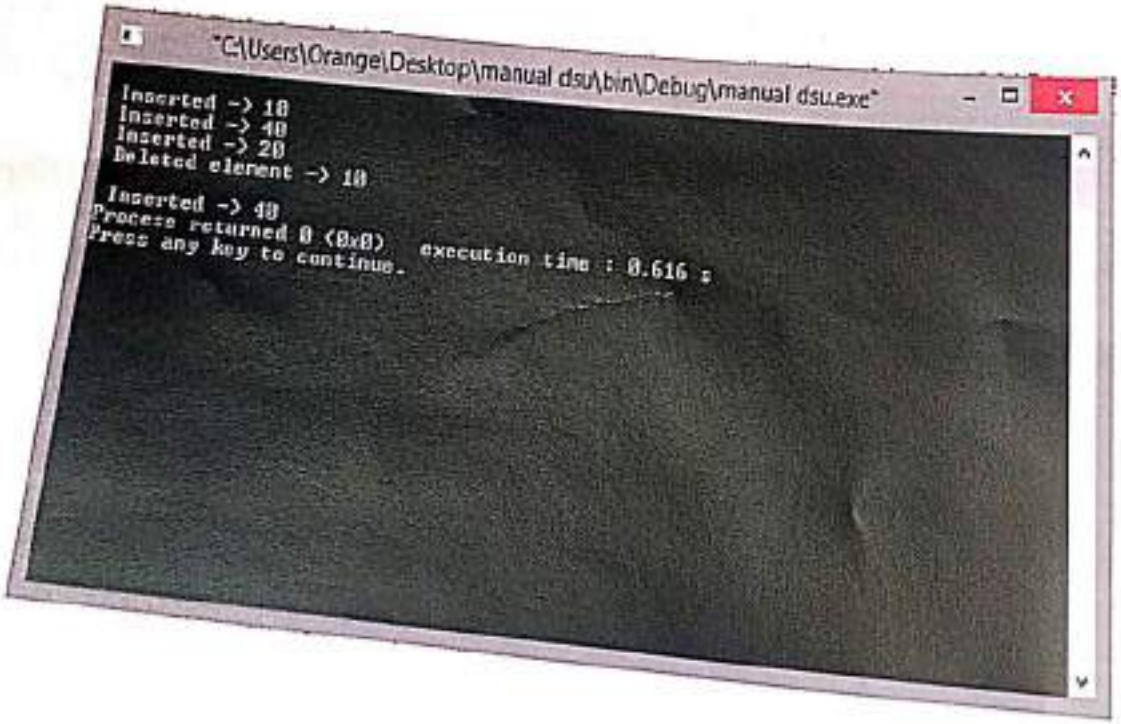
* Result :

Implement a 'C' program Insert and Delete operations in Circular Queue using Array.

* Conclusion :

In linear queue deleted item location is wasted because queue does not cannot occupy that location. So to overcome this Circular Queue can be the feasible option.

Related Questions


"C:\Users\Orange\Desktop\manual dsu\bin\Debug\manual dsu.exe"
Inserted -> 10
Inserted -> 40
Inserted -> 20
Deleted element -> 10
Inserted -> 40
Process returned 0 (0x0) execution time : 0.616 s
Press any key to continue.

-3

```
#include <stdio.h>
#define SIZE 10
int items[SIZE];
int front = -1, rear = -1;

int isFull() {
 if ((front == rear + 1) || (front == 0 && rear == SIZE - 1)) return 1;
 return 0;
}

int isEmpty() {
 if (front == -1) return 1;
 return 0;
}

void enQueue(int element) {
 if (isFull())
  printf("\n Queue is full!! \n");
 else {
  if (front == -1) front = 0;
  rear = (rear + 1) % SIZE;
  items[rear] = element;
  printf("\n Inserted -> %d", element);
 }
}

int deQueue() {
 int element;
 if (isEmpty()) {
  printf("\n Queue is empty !! \n");
  return (-1);
 } else {
  element = items[front];
  if (front == rear) {
   front = -1;
   rear = -1;
  }
  else {
   front = (front + 1) % SIZE;
  }
  printf("\n Deleted element -> %d \n", element);
  return (element);
 }
}

void display() {
 int i;
```

```
    if (isEmpty())
      printf(" \n Empty Queue\n");
    else {
     printf("\n Front -> %d ", front);
     printf("\n Items -> ");
     for (i = front; i != rear; i = (i + 1) % SIZE) {
      printf("%d ", items[i]);
     }
     printf("%d ", items[i]);
     printf("\n Rear -> %d \n", rear);
   }
  }

  int main() {
   enQueue(23);
   enQueue(12);
   enQueue(45);
   enQueue(23);
   enQueue(47);
   enQueue(50);
return 0;
}
```

10. Perform Operations of Singly linked List.

Flowchart Resource Used

Sr. No	Name of Resource	Specifications
1	Computer System	Dual-core, 4gb RAM
2	Software	Turbo C
3	Any other	—

* Algorithm

i] Insertion at head

1] addFirst (string newdata);
2] Create a new node v containing newData
3] v.setNext (head)
4] head = v
5] size = size + 1

ii] Insertion at last

1] addLast (string newdata);
2] Create a new node v containing new data
3] v.setNext (null)
4] if (head = null):
head = v

~~Code~~

5] else {
tail.setnext(v)
6] tail=v
7] size = size + 1

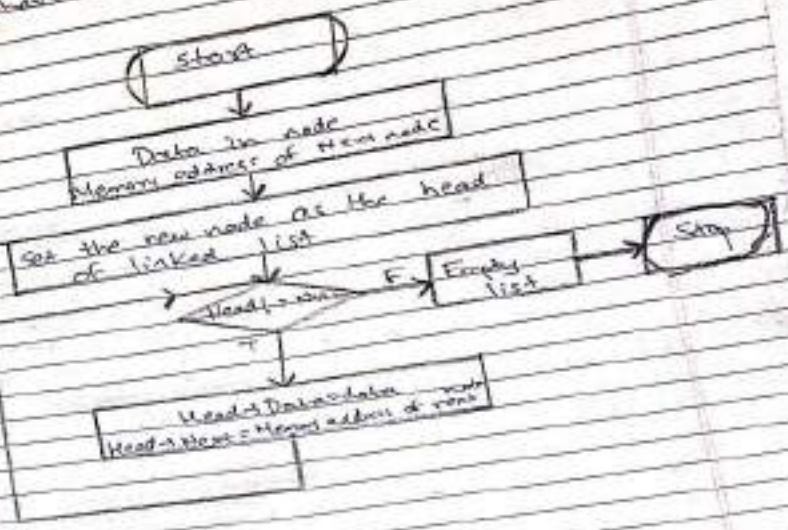
ii] Deletion

1] removeFirst()
2] if(head==null)
Error: list is empty
3] temp=head
4] head=head.getNext()
5] temp.setNext(null)
6] size = size - 1

iii] Traversing

1] traverse list()
2] current = head
3] while (current != null)
current = current.getNext()
}

Flowchart



Code

```
#include<iostream>
#include<cstdio>
#include<cstdlib>
using namespace std;
struct node
{
   int info;
   struct node *next;
}*start;
class single_llist
{
   public:
      node* create_node(int);
      void insert_begin();
      void delete_pos();
      void search();
      void display();
      single_llist()
      {
         start = NULL;
      }
};
int main()
{
   int choice, nodes, element, position, i;
   single_llist sl;
   start = NULL;
   while (1)
   {
      cout<<"1.Insert Node at beginning"<<endl;
      cout<<"2.Delete a Particular Node"<<endl;
      cout<<"3.Search Element"<<endl;
      cout<<"4.Display Linked List"<<endl;
      cout<<"5.Exit "<<endl;
      cout<<"Enter your choice : ";
      cin>>choice;
      switch(choice)
      {
      case 1:
         cout<<"Inserting Node at Beginning: "<<endl;
         sl.insert_begin();
         cout<<endl;
         break;
      case 2:
         cout<<"Delete a particular node: "<<endl;
         sl.delete_pos();
         break;
```

```
            case 3:
              cout<<"Search element in Link List: "<<endl;
              sl.search();
              cout<<endl;
              break;
            case 4:
              cout<<"Display elements of link list"<<endl;
              sl.display();
              cout<<endl;
              break;
            case 5:
              cout<<"Exiting.."<<endl;
              exit(1);
              break;
            default:
              cout<<"Wrong choice"<<endl;
          }
        }
      }
      node *single_llist::create_node(int value)
      {
        struct node *temp, *s;
        temp = new(struct node);
        if (temp == NULL)
        {
          cout<<"Memory not allocated "<<endl;
          return 0;
        }
        else
        {
          temp->info = value;
          temp->next = NULL;
          return temp;
        }
      }
      void single_llist::insert_begin()
      {
        int value;
        cout<<"Enter the value to be inserted: ";
        cin>>value;
        struct node *temp, *p;
        temp = create_node(value);
        if (start == NULL)
        {
          start = temp;
          start->next = NULL;
        }
```

```
        else
        {
          p = start;
          start = temp;
          start->next = p;
        }
        cout<<"Element inserted at beginning"<<endl;
      }
      void single_llist::delete_pos()
      {
        int pos, i, counter = 0;
        if (start == NULL)
        {
          cout<<"List is empty"<<endl;
          return;
        }
        cout<<"Enter the position of value to be deleted: ";
        cin>>pos;
        struct node *s, *ptr;
        s = start;
        if (pos == 1)
        {
          start = s->next;
        }
        else
        {
          while (s != NULL)
          {
            s = s->next;
            counter++;
          }
          if (pos > 0 && pos <= counter)
          {
            s = start;
            for (i = 1;i < pos;i++)
            {
              ptr = s;
              s = s->next;
            }
            ptr->next = s->next;
          }
          else
          {
            cout<<"Position out of range"<<endl;
          }
          free(s);
          cout<<"Element Deleted"<<endl;
```

```
}
void single_llist::search()
{
    int value, pos = 0;
    bool flag = false;
    if (start == NULL)
    {
        cout<<"List is empty"<<endl;
        return;
    }
    cout<<"Enter the value to be searched: ";
    cin>>value;
    struct node *s;
    s = start;
    while (s != NULL)
    {
        pos++;
        if (s->info == value)
        {
            flag = true;
            cout<<"Element "<<value<<" is found at position "<<pos<<endl;
        }
        s = s->next;
    }
    if (!flag)
        cout<<"Element "<<value<<" not found in the list"<<endl;
}
void single_llist::display()
{
    struct node *temp;
    if (start == NULL)
    {
        cout<<"The List is Empty"<<endl;
        return;
    }
    temp = start;
    cout<<"Elements of list are: "<<endl;
    while (temp != NULL)
    {
        cout<<temp->info<<"->";
        temp = temp->next;
    }
    cout<<"NULL"<<endl;
```

Exercise: Create a program to perform the operations (Insert, Delete, Traverse and Search) on Singly linked list.

Conclusion

In order to use and perform operations on dynamic memory allocation in application non-contiguous data structure is required. So linked list is used to organize data in non-contiguous form.

Practical Related Questions

1) Single linked list uses ___ number of pointers.
a) zero
b) one
c) two
d) three
→ two

2) No. of pointers to be manipulated in a linked list to insert an item in the middle ___
a) two
b) three
c) one
d) zero
→ a) two

IV.
V. Pra
VI. Refer
1. Fo
2. Fol

2] Linked list are not suitable for the structure of which one of the following problem?
a] Insertion Sort
b] Binary Search
c] Radix Sort
d] Polynomial Manipulation problem
→ b] Binary Search

3] The linked list field(s) are
a] Data
b] Pointer
c] Pointer next node
d] Data & pointer to next node

Exercise

1] Give syntax of creation of a node,

typedef struct LinkedList *node;
node createNode() {
node temp;
temp = (node)malloc(sizeof(struct LinkedList));
temp→next = NULL;
return temp;
}

2] Represent linked list as Stack and Queue.

→

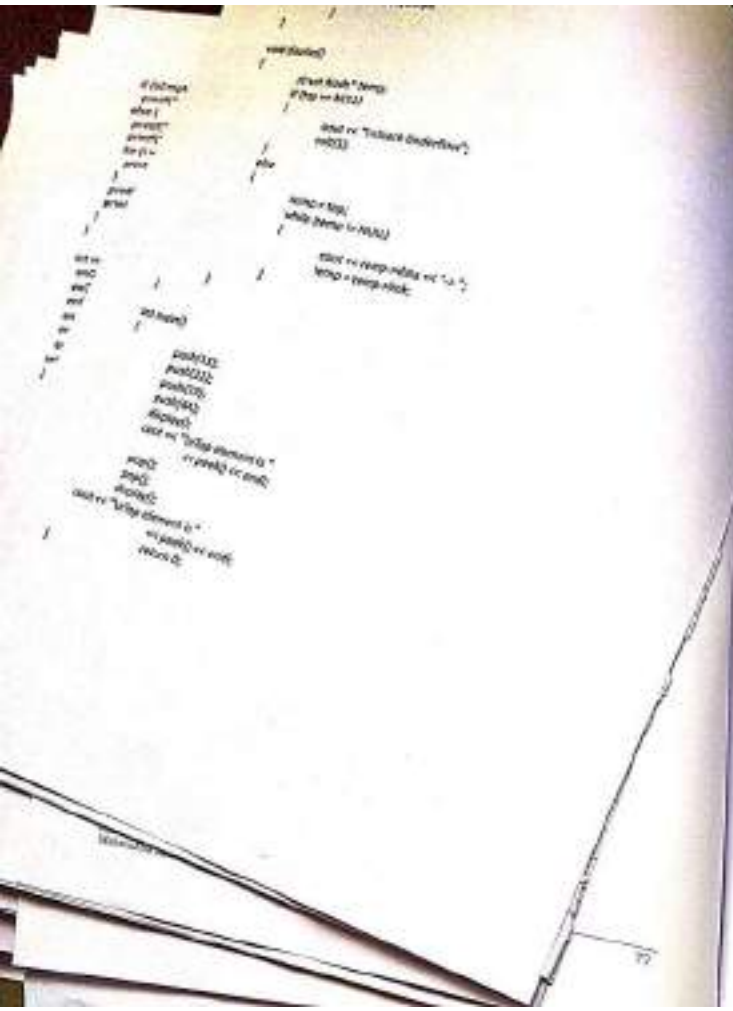
3] Give output of

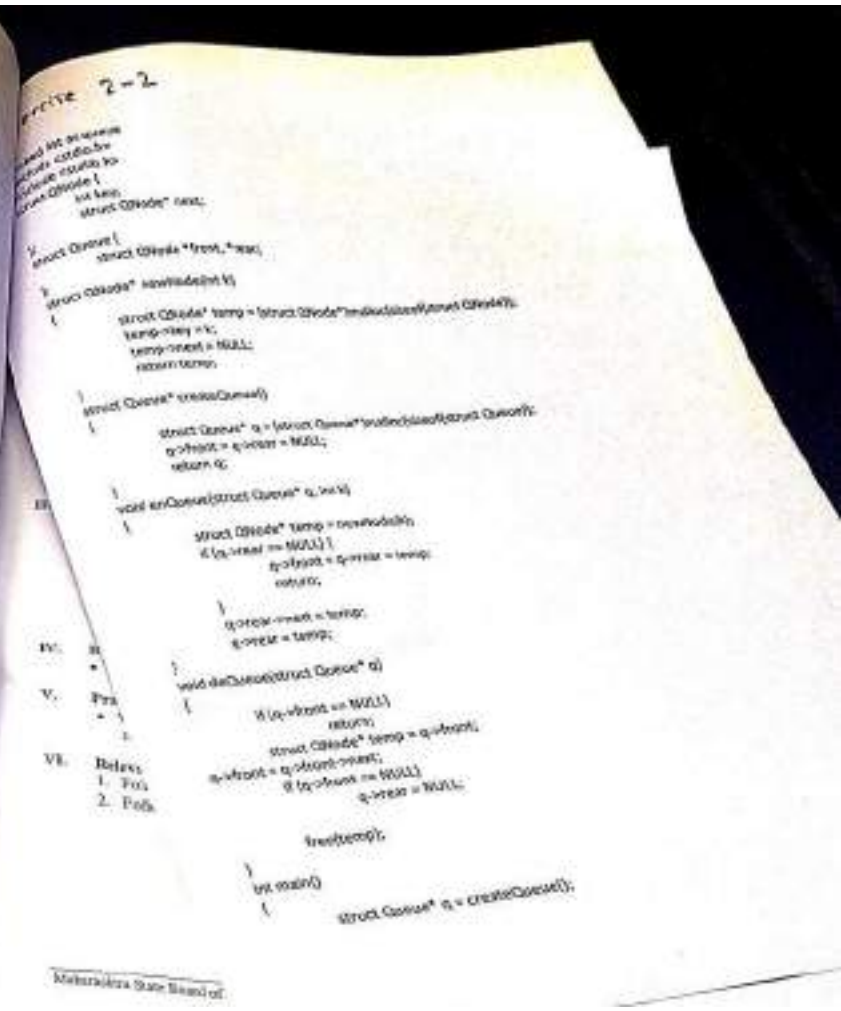
void printList()
{
struct node *ptr = head;
printf("\n[");
// start from begin
while(ptr != null) {
printf("(%d,%d)", ptr→key, ptr→
ptr = ptr→next;}
printf("]");}

→ [ ]

ercise - 2 - 1

```
nked list as stack
nclude <iostream>
sing namespace std;
truct Node
{
        int data;
        struct Node* link;
};
struct Node* top;
void push(int data)
{
        struct Node* temp;
        temp = new Node();
        if (!temp)
        {
                cout << "\nHeap Overflow";
                exit(1);
        }
        temp->data = data;
        temp->link = top;
        top = temp;
}
int isEmpty()
{
        return top == NULL;
}
int peek()
{
        if (!isEmpty())
                return top->data;
        else
                exit(1);
}
void pop()
{
        struct Node* temp;
        if (top == NULL)
        {
                cout << "\nStack Underflow" << endl;
                exit(1);
        }
        else
        {
                temp = top;
                top = top->link;
                temp->link = NULL;
```





```
    enQueue(q, 10);
    enQueue(q, 20);
    deQueue(q);
    deQueue(q);
    enQueue(q, 30);
    enQueue(q, 40);
    printf("Queue Front : %d \n", q->front->key);
    printf("Queue Rear : %d", q->rear->key);
    return 0;
}
```

Date: / /
Page No.:

Perform Operations on Circular singly linked list

Resources Used:

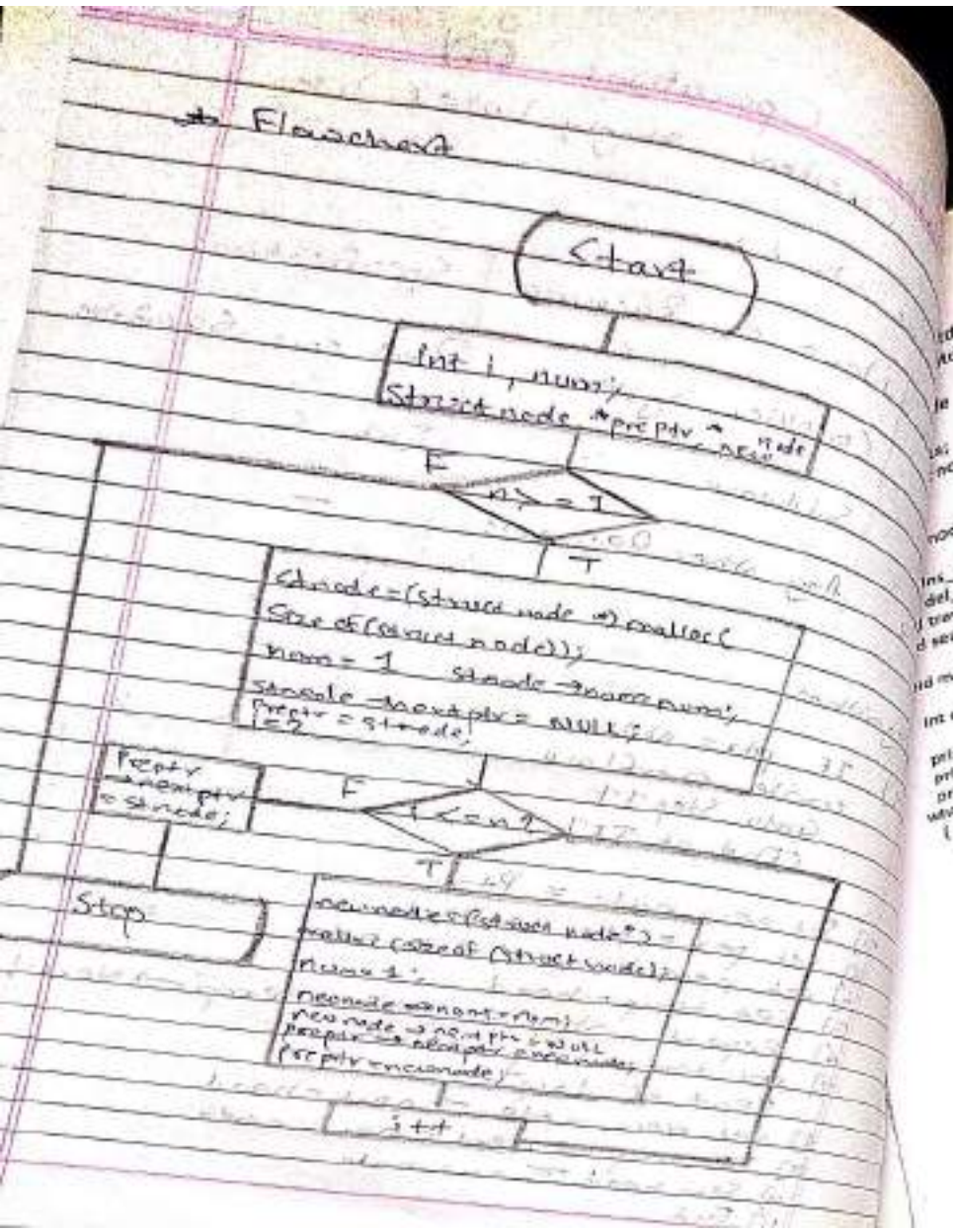
Sr. No	Name of Resource	Specifications
1	Computer System	Dual core, 6GB RAM
2	Software	Turbo C
3	Any other Resources	—

* Algorithm

1] If ptr = NULL
write Overflow
Goto Step 11
[End of If]

2] Set New_node = Ptr
3] Set ptr = ptr → Next
4] Set new_node → Data = VAL
5] Set temp = head
6] Repeat step 8 while temp → Next != head
7] Set temp = temp → Next
[End of Loop]
8] Set new_node → next = head
9] Set temp → next = New_node
10] Set head = new_node
11] Exit

Flowchart




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

struct node
{
    int info;
    struct node *link;
};

struct node *head = NULL, *x, *y, *z;

void ins_at_beg();
void del_at_pos();
void traverse();
void search();

void main()
{
    int ch;

    printf("\n1.Insertion at beginning");
    printf("\n2.Deletion at position \n3.traverse");
    printf("\n4.Search");
    while (1)
    {
        printf("\n Enter your choice:");
        scanf("%d", &ch);
        switch(ch)
        {
        case 1:
            ins_at_beg();
            break;
        case 2:
            del_at_pos();
            break;
        case 3:
            traverse();
            break;
        case 4:
            search();
            break;
        default:
            exit(0);
        }
    }
}
void create()
```

```
{
  int c;

  x = (struct node*)malloc(sizeof(struct node));
  printf("\n Enter the data:");
  scanf("%d", &x->data);
  x->link = x;
  head = x;
  printf("\n If you wish to continue press 1 otherwise 0:");
  scanf("%d", &c);
  while (c != 0)
  {
    y = (struct node*)malloc(sizeof(struct node));
    printf("\n Enter the data:");
    scanf("%d", &y->data);
    x->link = y;
    y->link = head;
    x = y;
    printf("\n If you wish to continue press 1 otherwise 0:");
    scanf("%d", &c);
  }
}
void ins_at_beg()
{
  x = head;
  y = (struct node*)malloc(sizeof(struct node));
  printf("\n Enter the data:");
  scanf("%d", &y->data);
  while (x->link != head)
  {
    x = x->link;
  }
  x->link = y;
  y->link = head;
  head = y;
}
void del_at_pos()
{
  if (head == NULL)
    printf("\n List is empty");
  else
  {
    int c = 1, pos;
    printf("\n Enter the position to be deleted:");
    scanf("%d", &pos);
    x = head;
    while (c < pos)
```

```
c
x->link;
;

ink = x->link;
e(x);


raverse()

head == NULL)
printf("\n List is empty");
se
{
  x = head;
  while (x->link != head)
  {
    printf("%d->", x->data);
    x = x->link;
  }
  printf("%d", x->data);
}
}
void search()
{
  int search_val, count = 0, flag = 0;
  printf("\nenter the element to search\n");
  scanf("%d", &search_val);
  if (head == NULL)
    printf("\nList is empty nothing to search");
  else
  {
    x = head;
    while (x->link != head)
    {
      if (x->data == search_val)
      {
        printf("\nthe element is found at %d", count);
        flag = 1;
        break;
      }
      count++;
      x = x->link;
    }
    if (x->data == search_val)
    {
```

```
        printf("element found at postion %d", count);
    }
    if (flag == 0)
    {
        printf("\nelement not found");
    }
}
}
```

Result :

Write a C program to perform the operations (Traverse, Delete, Search, Count) in Circular Singly linked list

Conclusion :

In some application Dynamic memory allocations using Single singly linked list may cause to storage wastage if data items are deleted. So Singly linked list can be used in circular manner. It gives flexibility in storage management.

Practical Related Questions

1) Circular Singly linked list needs ____ no. of pointer
a) zero b) two
c) one d) three
→ b) one

2) No. of pointers to be manipulated in a Circular linked list to insert an item in middle ____.
a) two b) one
c) three d) zero
→ a) one

Linked list are not suitable for the data structure of which one of the following problem?
a] Insertion sort
b] Binary search
c] Radix sort
d] Polynomial Manipulation

b] Binary Search

1] The last node of Circular linked list fields having
a] data
b] Pointer
c] pointer next to node
d] Pointer to first node.

→ d] Pointer to first node

★ Exercise

1] Give the benefit of Circular Singly linked list.

→

1] In circular linked list, end node will point to first node whereas in singly linked list it won't point to first node.
2] Circular linked list can be used for Implementation of Queue.
3] It saves time when we have to go to the first node from the last node. It can be done in single step because there is no need to traverse the in between nodes.

2] Represent Linked list as circular Queue.

```
e - 2

dio.h>
dlib.h>
de

int data;
struct node* next;

t node *f = NULL;
t node *r = NULL;
d enqueue(int d)

    struct node* n;
    n = (struct node*)malloc(sizeof(struct node));
    n->data = d;
    n->next = NULL;
    if((r==NULL)&&(f==NULL))
    {       f = r = n;
            r->next = f;

    }
    else
    {       r->next = n;
            r = n;
            n->next = f;

    }
}
void dequeue()
{       struct node* t;
        t = f;
        if((f==NULL)&&(r==NULL))
                printf("\nQueue is Empty");
        else if(f == r){
                f = r = NULL;
                free(t);

        }
        else{   f = f->next;
                r->next = f;
                free(t);

        }
}
void print()
{       struct node* t;
        t = f;
        if((f==NULL)&&(r==NULL))
```

```
                printf("\nQueue is Empty");
        else{
                do{
                        printf("\n%d",t->data);
                        t = t->next;
                }while(t != f);
        }
}
int main()
{
        int opt,n,i,data;
        printf("Enter Your Choice:-");
        do{
                printf("\n\n1 for Insert the Data in Queue\n2 for show the Data in Queue \n3
for Delete the data from the Queue\n0 for Exit");
                scanf("%d",&opt);
                switch(opt){
                        case 1:
                                printf("\nEnter the number of data");
                                scanf("%d",&n);
                                printf("\nEnter your data");
                                i=0;
                                while(i<n){
                                        scanf("%d",&data);
                                        enqueue(data);
                                        i++;
                                }
                                break;
                        case 2:
                                print();
                                break;
                        case 3:
                                dequeue();
                                break;
                        case 0:
                                break;
                        default:
                                printf("\nIncorrect Choice");

                }
        }while(opt!=0);
return 0;
}
```

Date: / /
Page No.:

2. Perform traversing on binary Search tree

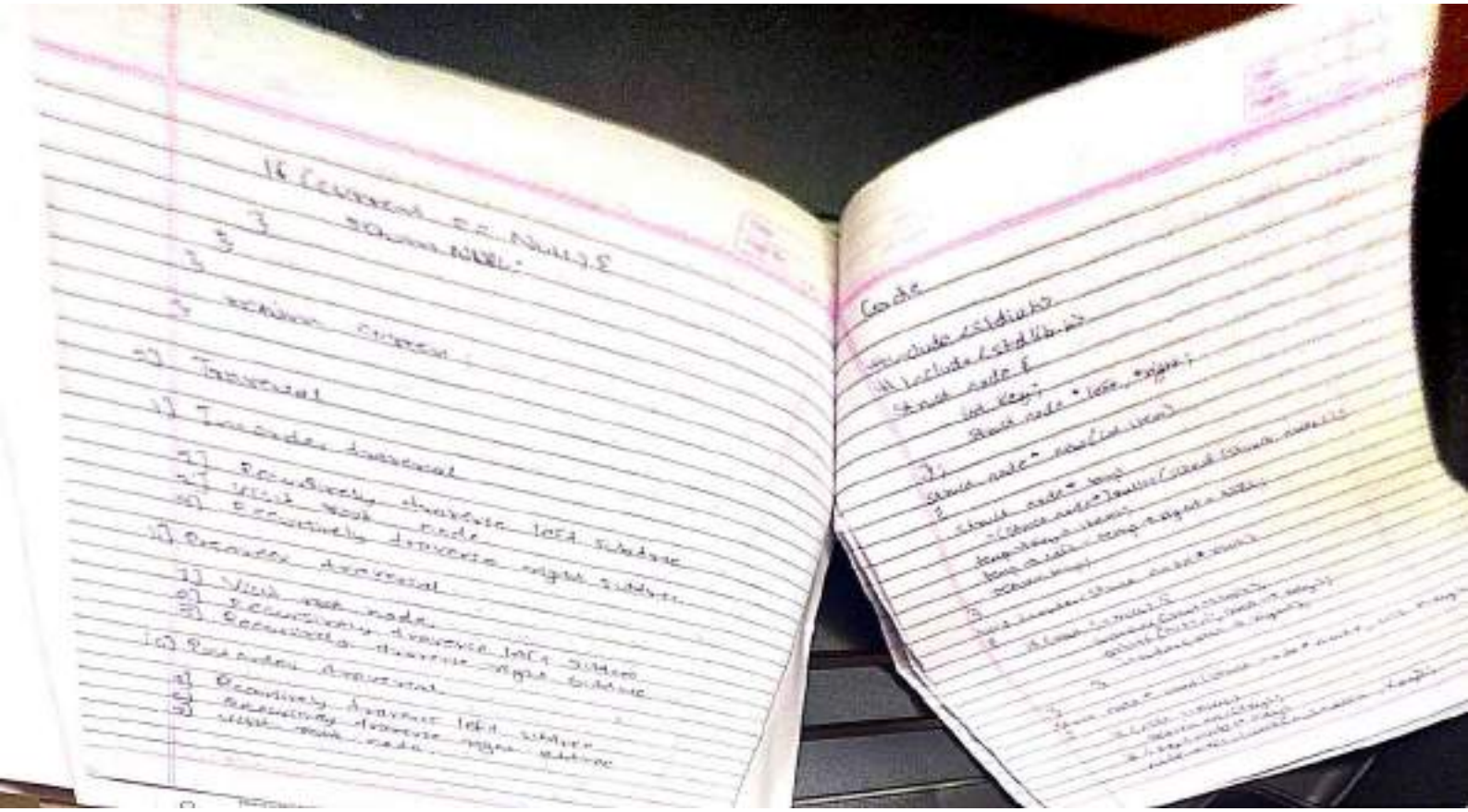
* Resources Used:

Sr. No.	Name of Resource	Specifications
1	Computer System	Dual-core, 6gb RAM
2	Software	Turbo C
3	Any other resources	—

* Algorithm

1) Create B.S.T

```
struct node* search(int data){
    struct node *current = root;
    printf("Visiting element:");
while (current->data != data){

    if (current!= NULL){
        printf("%d", current->data);

    if (current->data > data){
        current = current->leftChild;
    }
    else
        { current = current->rightChild;
    }
```



```
if(current == NULL){
    return NULL;
}
}
}
return current;
}
```

2] Traversal

i] In-order traversal

1] Recursively traverse left subtree
2] Visit root node.
3] Recursively traverse right subtree.

ii] Preorder traversal

1] Visit root node.
2] Recursively traverse left subtree
3] Recursively traverse right subtree

iii] Postorder traversal

1] Recursively traverse left subtree
2] Recursively traverse right subtree
3] Visit root node.

Code

```
#include <stdio.h>
#include <stdlib.h>
struct node {
    int key;
    struct node * left, *right;
};
struct node * new(int item)
{
  struct node* temp
    =(struct node*)malloc(sizeof(struct node));
  temp->key = item;
  temp->left = temp->right = NULL;
  return temp;
}
void inorder(struct node* root)
{
    if(root != NULL) {
          inorder(root->left);
          printf("%d\n", root->key);
          inorder(root->right);
        }
}
struct node * insert(struct node* node, int key)
{
    if(node == NULL)
        return new(key);
    if (key < node->key)
      node->left=insert(node->left, key);
```

```
elseif (key > node->key)
    node->right = insert(node->right, key);
    return node;
}
int main()
{
    struct node *root = NULL;
    root = insert(root, 50);
    insert(root, 30);
    insert(root, 20);
    insert(root, 40);
    insert(root, 70);
    insert(root, 60);
    insert(root, 80);
    inorder(root);
    return 0;
}
```

Result :

wrote a C program to implement BST and traverse the tree (inorder, preorder, postorder)

* Conclusion :

To perform operations in some applications which work on hierarchical data a Tree data structure is most known data structure. Tree is used to represent data in hierarchical manner. A BST can be used to represent ordered array and a linked list.

* Practical Related Questions.

1] Every binary tree with n elements have ____ no. of edges

a] n
b] n-1
→ c] n-2
d] n+1

2] Define inorder traversal ____

a] left, root, right
b] root, left, right
→ c] left, root, right
d] left, right, root
d] none

3] Define postorder traversal
a] left, right, root
b] root, left, right
c] left, right, root
d] none

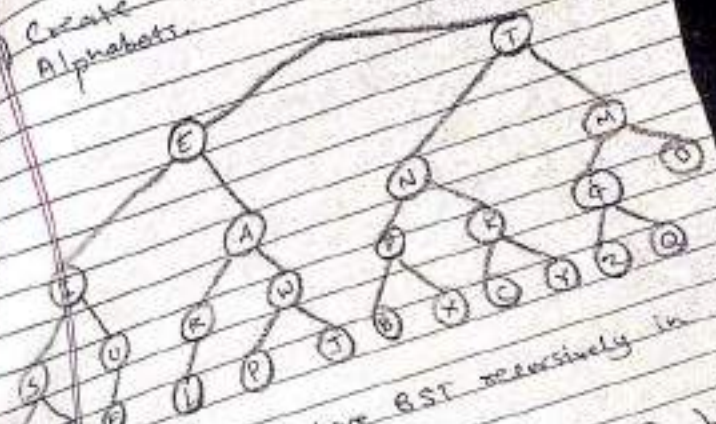
→ b] root, left, right

4] Define converse postorder traversal
a] left, root, right
b] root, left, right
c] right, left, root
d] none

→ d] none

Exercise

1] Create binary search tree of Capital Alphabets.



2] Traverse the above BST recursively in Preorder.

→ E I S H V U E A R L W P J I N D B X K C Y M G Z O Q