DYP

ASSIGNMENT-1.
SEARCHING AND SORTING.

AIM: To implement various searching and sorting algorithms on database of students implemented using arrays of structure.

THEORY :

1] Structure :
Structure is collection of heterogeneous types of data in C++, a structure collects different data items in such a way that they can be referenced as a single unit. Data items in a structure are called fields or members of the structure.

2] Array of structure.
To declare an array of a structures, we first define a structure and then declare an array variable of that type. To declare an 20 element array of structu of type student define,

```
struct student s[20];
```

To access a specific structure, index the array nam

```
cout << s.[3].name;
```

3] Sorting
It is the process of arranging or ordering information in the ascending or descending order of the key values.

A] Bubble sort
This is one of the simplest and most popular sortin

Step 2 : End if
Step 3 : end of quicksort

5. Linear Search

Linear_search (struct student s[], float key, int n)
// Here s is array of structure student, key is sgpa of student to be searched and
// displayed, n is total number of students in record.
Step 1 : Set i=0 & flag to 0
Step 2 : While i<n
    i. If s[i].sgpa == key
        a. Print s[i].roll no, s[i].name
        b. set flag to 1
        c. i++.
Step 3 : End while
Step 4 : If flag==0
    i. Print No student found with sgpa = value of key
Step 5 : End if
Step 6 : End of linear_sear

6. Binary Search

// Where s is an array of structure, n is the no. of records, key is element to be searched
Step 1 : Set l=0 & h=n-1
Step 2 : While l≤h
    i. mid = (l+h)/2
    ii. If strcmp (s[mid].name, key)==0)
        a. found
        b. stop
    iii. Else
        a. if(Stcmp (key, s[mid].name)<0

DYP

Algorithm : Insert at front()

{
Allocate a memory for new node (new1)
if (new1 == null)
then
display memory is full (insertion is not possible)
end if
else
read element ele;
new1 → data = ele;
new1 → next = header → next;
header → next = new1;
end else
}

Algorithm : Insert at End()

{
Allocate a memory for new node (new1)
if (new1 == NULL)
then
display memory is full (insertion is not possible)
end if
else
ptr = header
while(ptr → next != NULL)
then
ptr = ptr → next
end
new1 → next = NULL
new1 → data = ele
ptr → next = new1

```
temp = ptr -> next;
ptr -> next = null;
free (temp);
end else
}
```

Algorithm : Delete at any position()

```
{
if (header -> next == null)
then
display list is empty
end if
ptr = header;
count = 0
while (count < pos - 1)
{
ptr = ptr -> next;
count ++;
}
temp = ptr -> next;
ptr -> next = temp -> next;
free (temp)
}
```

Traversing a list:

Algorithm : Traverse

```
{
ptr = header;
if (ptr -> next == NULL)
then
display list is empty
```

```
header -> next = header -> next -> next;
temp -> next -> prev = header
free (temp);
end if
}
```

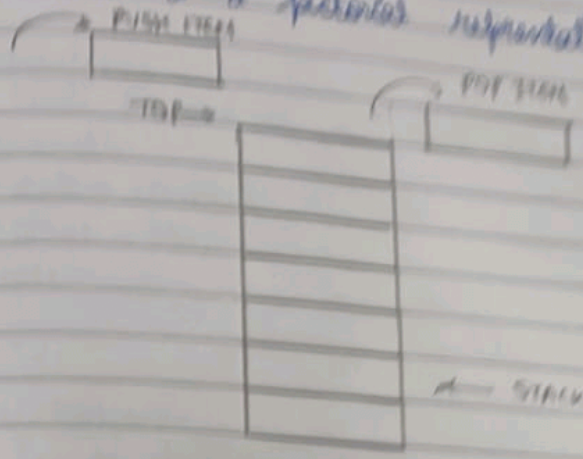
Traverse

1. If head is NULL -> display "List is empty"
2. Else

```
Set temp = head
do
  Print temp -> data
  temp = temp -> next
  While temp != head
```

iv) Diagram

Given below is a pictorial representation of stack.



As shown above, there is a pile of plates stacked on top of each other. If we want to add another item to it, then we add it at the top of the stack as shown. This operation of adding an item to stack is called "Push"

On the right side, we have shown an opposite operation i.e we remove an item from the stack. This is called "Pop".

ADT of stack

Define structure for stack (data, next pointer)

1. Stack Empty

Return true if stack Empty else False

Top is a pointer of type structure stack.

Empty (Top)

a. if Top == NULL

b. return 1

c. else
d. return 0;

2. Push Operations :
Top & Node pointer of structure stack
Push (element)
a. Node → data = element;
b. Node → Next = Top;
c. Top = Node
d. Stop

3. Pop Operations :
Top & Temp pointer of structure stack
Pop()
a. if Top != NULL
   i. Temp = Top;
   ii. element = Temp → data;
   iii. Top = (Top) → Next;
   iv. delete Temp;
b. Else stack is empty
c. return element

Realization of stack ADT

i. Using sequential Organisation (Array)
The three basic stack operations are push, pop and stack top. Push is used to insert data into the stack. Pop removes data from a stack and returns the data to the calling module. Stack top returns the data at the top of the stack without deleting the data from the stack.

Push:
Push adds an item at the top of the stack. After the push, the new item becomes the top. The only potential problem with this simple operation is that we must ensure that there is room for the new item. If there is not enough room, the stack is an overflow state and the item cannot be added.

Pop:
When we pop a stack, we remove the item at the top of the stack and return it to the user. Because we have removed the top item, the next older item in the stack becomes the top. When the last item in the stack is deleted, the stack must be set to its empty state. A pop is called when the stack is empty, it is in an underflow state. The pop stack overflow.

Stack top.
The third stack operation is stack top. Stack-top copies the item at the top of the stack, i.e it returns the data in the top element to the user but does not delete it.

ii] Using Linked organisation
To implement the linked list stack, we need two different structures, a head and a data node. The head structure contains meta data - that is data about data - and a pointer to the top of the stack.

```
ptr = ptr → next
count ++
}
if (count < pos - 1)
then
display position is not of range
end
ptr = header
count = 0
while (count < pos - 1)
then
ptr = ptr → next
count ++.
end
new1 → data = ele
new1 → next = ptr → next;
new1 → prev = ptr;
ptr → next → prev = new1;
ptr → next = new1
end else
}
```

Deleting a node from the DLL.

Algorithm : delete at front()

```
{
if (header → next == NULL)
then
display list is empty
end if
else
temp = header → next;
```

DYP

end else
}

Algorithm : Insert at middle()
{
Allocate a memory for new node(new1)
if (new1 == NULL)
then
display memory is full (insertion is not possible)
exit
else
read ele, pos
ptr = header
count = 0
while(ptr → next != NULL)
{
ptr = ptr → next;
count ++;
}
if(count < pos - 1)
then
display position is not of range
end
ptr = header
count = 0
while (count < pos - 1)
then
ptr = ptr → next
count++.
end
new1 → data = ele

4] Searching
Searching is a process of retrieving or locating a record with a particular key value.

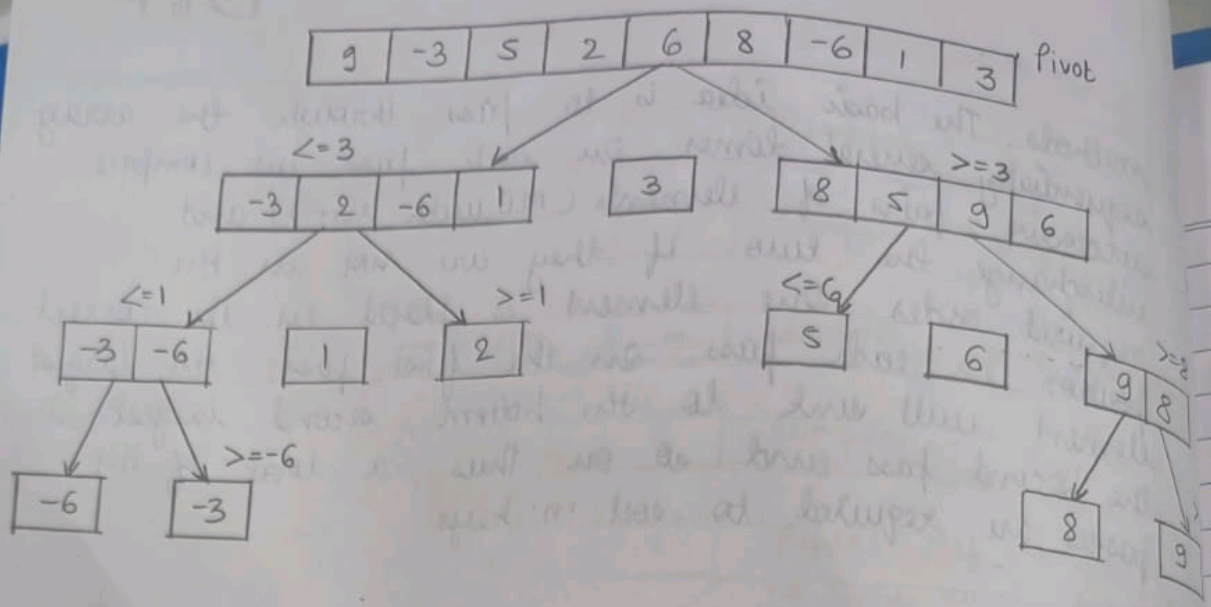
A] Linear search
Linear search is the simplest searching algorithm that searches for an element in a list in sequential order. We start at one end and check every element until the desired element is not found.

B] Binary search
This is an effecient searching technique used on sorted data (either in ascending or descending order).
It works by repeatedly comparing the target key with the middle element of the list.
- If the target equals the middle element - search is complete.
- If the target is less → search continues in the left half.
- If the target is greater → search continues in the right half.
- Each comparison eliminates half of the remaining elements, making the process very efficient. This halving continues until the element is found or the list can't be divided further.

ALGORITHM / PSEUDOCODE :

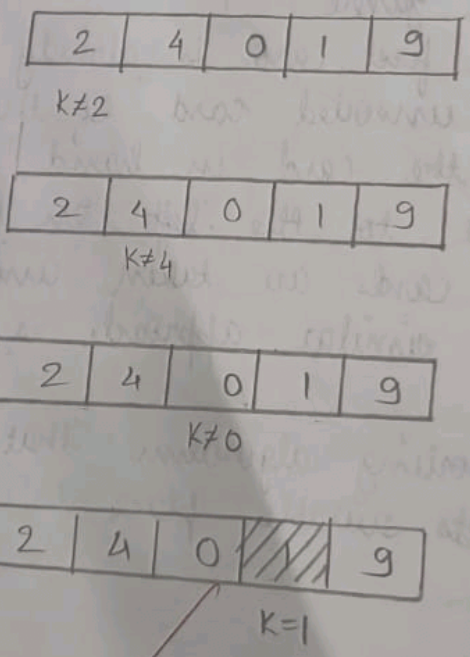
1. Create a structure
Create_database(struct student s[])
step 1: Accept how many records user need to add, say

Quick Sort



Linear Search

K=1



DYP

methods. The basic idea is to pass through the array sequentially several times. On each pass we compare successive pairs of elements (A[i] with A[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. On the 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.

b) Insertion sort.

Insertion sort works similarly as we sort cards in our hand in a card game.

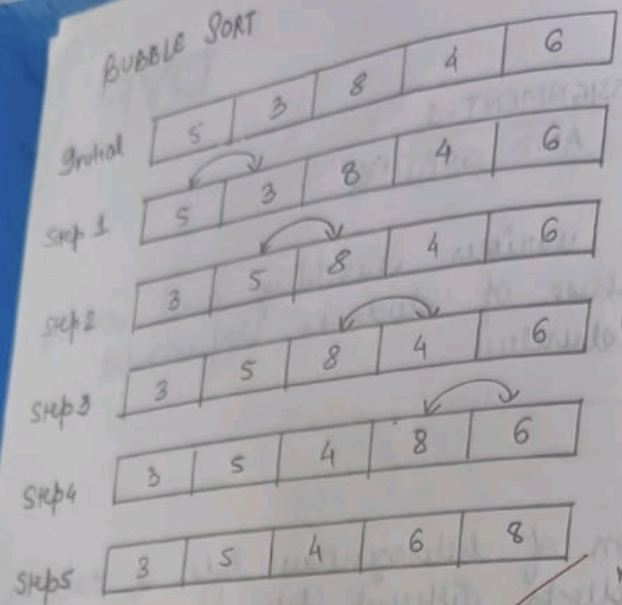
We assume that the first card is already sorted then, we select an unsorted card. If the unsorted card is greater than the card in hand, it is placed on the right otherwise, to the left. In the same way, other unsorted cards are taken and put at their right place. A similar approach is used by insertion sort.

Insertion sort is a sorting algorithm that places an unsorted element at its suitable place in every situation.

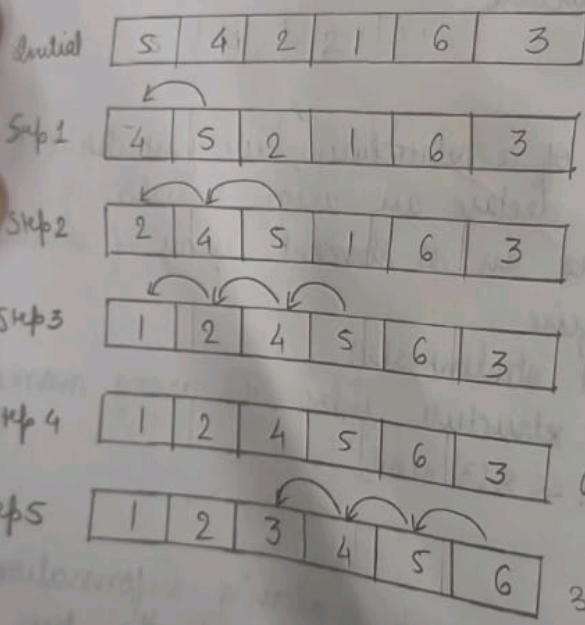
c) Quick sort

Quick sort is an algorithm based on divide and conquers approach in which the array is split into sub arrays and these sub arrays are recursively called to sort the elements.

BUBBLE SORT



Insertion Sort



DYP

i. h = mid - 1

b. Else

ii. l = mid + 1

c. End if

iv. End if

Step 4: End while

Step 5: not found // search is unsuccessful.

```
end if
else
while (ptr → next != NULL)
{
display ptr → data
ptr = ptr → next
}
else if
}
```

AIM : Implement the double linked list inserting a node into DLL.

Algorithm : Insert at front()

```
{
Allocate a memory for new node (new1)
if (new1 == null)
then
display memory is full (insertion not possible)
end if
else
read element ele;
new1 → data = ele;
new1 → next = header → next;
header → next → prev = new1;
header → next = new1;
new1 → prev = header;
end else
}
```

Algorithm : Insert at end()

```
{
```

ASSIGNMENT 3
IMPLEMENTATION OF STACK.

AIM: Implementation of stack ADT and expression conversion.

THEORY:

What is an abstract datatype?

An abstract data type is defined as a mathematical model of the data object that makes up a data type as well as the functions that operate on these objects. There are no standard conventions for defining them. A broad division may be drawn between "imperative" and "functional" definition styles. In general terms, an abstract data type is a specification of the values and the operations that has two properties:

- It specifies everything you need to know in order to use the datatype.
- It makes absolutely no reference to the manner in which the data type will be implemented.

Stack.

1) Concept.

Stacks are more common data objects found in computer algorithms. It is special case of more general data object, an ordered or linear list. It can be defined as the ordered list, in which all insertions and deletions are made at one end, called as 'top' i.e last element inserted is

Stack Head

Generally, the head for a stack requires only two attributes: a top pointer and a count of the number of elements in the stack. These two elements are placed in a structure. Other stack attributes can be placed here also.

Stack Data Node

The rest of the data structure is a typical linked list data node. Although the application determines the data that are stored in the stack data node looks like any linked list node.

Push Stack

Push stack inserts an element into the stack. The first thing we need to do when we push data into a stack is find memory for the node. We must therefore allocate a node from dynamic memory.

Pop Stack

Pop stack sends the data in the node at the top of the stack back to the calling algorithm

Application of Stack

1. reversing data
2. parsing data
3. postponing data usage
4. backtracking steps.

outputted first (Last in First Out or LIFO)

ii) Definition

In computer science, a stack is a last in, first out (LIFO) abstract data type. A stack can have any abstract data type as an element, but is characterized by only two fundamental operations: push and pop. The push operation adds an item to the top of the stack, hiding any item already on the stack, or initializing the stack if it is empty. A pop either reveals previously concealed items already, or results in an empty stack. A stack is a restricted data structure, because only a small number of operations are performed on it.

iii) Terminologies

An abstract data type (ADT) consist of a data structure and a set of primitives

a. Initialize - creates / initializes the stack

b. Push - add a new element

c. Pop - removes a element

d. Isempty - report whether the stack is empty

e. Isfull - report whether the stack is full

f. Destroy - delete the contents of the stack

Allocate a memory for new node (new1)
if (new1 == NULL)
then
display memory is full (insertion is not possible)
end if
else
ptr = header
while (ptr → next != NULL)
then
ptr = ptr → next
end
new1 → next = NULL
new1 → prev = ptr;
new1 → data = ele
ptr → next = new1
end else
}

Algorithm : Insert at middle)
{
Allocate a memory for new node (new1)
if (new1 == NULL)
then
display memory is full (insertion is not possible)
exist
else
read ele, pos
ptr = header
count = 0
while(ptr → next != NULL)
{

```
new1 -> next = ptr -> next
ptr -> next = new1
end else
}
```

Deleting a node from the SLL

Algorithm : Delete at front()

```
{
if (header -> next == NULL)
then
display list is empty
end if
else
temp = header -> next;
header -> next = header -> next -> next;
free (temp);
end if
}
```

Algorithm : Delete at end()

```
{
if (header -> next == NULL)
then
display list is empty
end if
else
ptr = header;
while (ptr -> next -> next != NULL)
{
ptr -> = ptr -> next;
}
```

DYP

ASSIGNMENT - 2
LINKED LIST

AIM: To implement basic operations (creation, insertion, deletion, and transversal) for singly, doubly, and circular linked lists

THEORY:

a] Singly Linked List - It is a linear data structure where each element is called node, and each node contains two fields:
- Data - stores the actual value
- Next Pointer - points to the next node in the li

b] Doubly Linked List - It extends the concept of SLL by adding a previous pointer to each node. Each node contains:
- Data
- Pointer to the next node
- Pointer to the previous node

c] Circular Linked List - In the last node's next pointer points back to the first node, forming a circle. There are two types:
- Circular singly linked list - Like SLL, but the last node to the first.
- Circular Doubly linked list - Like SLL, but links in both directions and the last connects to the first.

1] AIM: Implement the single linked list.
Inserting a node into SLL:

DYP

iv. End While
Step 2: Assign key to s[j+1]
Step 3: End for
Step 4: end of insertion sort

4. Quick sort to sort students on the basis of their sgpa
partition (struct student s[], int l, int h)
// where s is the array of structure, l is the index of starting element
// and h is the index of last element
Step 1: Select s[l].sgpa as the pivot element
Step 2: Set i=l
Step 3: Set j=h-l
Step 4: While i≤j
i. Increment i till s[i].sgpa ≤ pivot element
ii. Decrement j till s[j].sgpa > pivot element
iii. If i<j
iv. Swap (s[i], s[j])
v. End if
Step 5: End while
Step 6: Swap (s[j], s[l])
Step 7: return j
Step 8: end of Partition.

quicksort (struct student s[], int l, int h)
// where s is the array of structure, l is the idex of starting
//and h is the index of last element
Step 1: If l<h
i. p = partition (s, l, h)
ii. quicksort (s, l, p-1)
iii. quicksort (s, p+1, h)

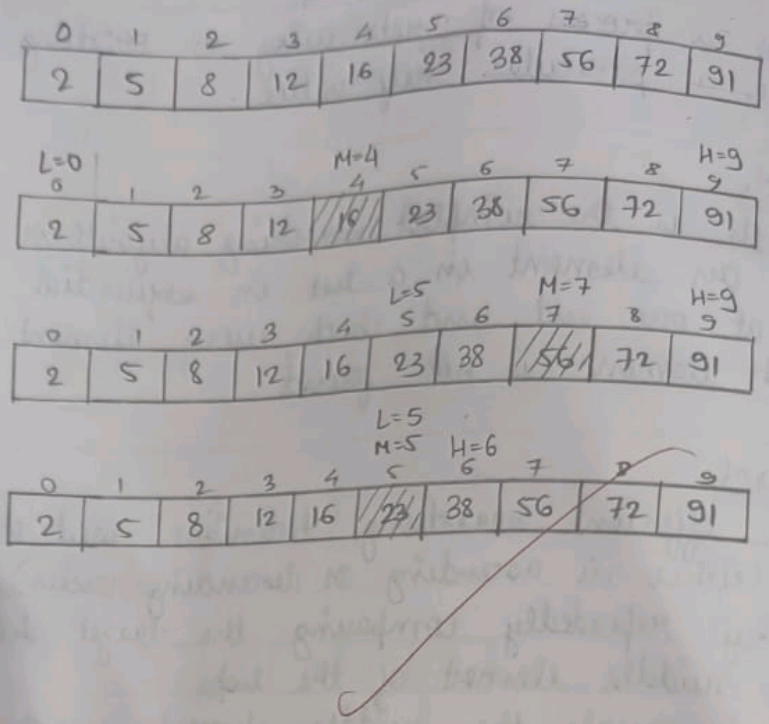
no of records as n.
Step 2 : For i=0 to n-1
    i. Accept the record and store it in s[i]
Step 3 : End for
Step 4 : Stop

Display database(struct student s[], int n)
Step 1 : For i=0 to n-1
    i. Display the fields s.roll_no, s.name, s.sgpa
Step 2 : End for
Step 3 : Stop

2. Bubble sort student according to sort to roll numbers
Bubble sort(Student s[], n)
Step 1 : For Pass=1 to n-1
Step 2 : For i=0 to (n-pass-1)
    i. If s[i].roll_no <s[i+1].roll_no
        a. swap(s[i]+s[i+1])
    iii. End if
Step 3 : End for
Step 4 : End for
Step 5 : Stop

3. Insertion sort to sort student on the basis of names
insertion Sort (Struct student s[], int n)
Step 1 : For i=1 to n-1
    i. Set key to s[i]
    ii. Set j to i-1
    iii. While j>=0 AND strcmp(s[i].name, key.name)>0
        a. Assign s[i] to s[j+1]
        b. decrement j

Binary search



Search 23

23 > 16
take 2nd half

23 < 56
take 1st half

found 23
Return 5

D. Y. Patil College of Engineering, Akurdi, Pune -44

EARN & LEARN SCHEME

Date : 18/08/2025

This academic year, our institute will be implementing 'Earn & Learn Scheme' for the benefit of the needy students. This scheme is being carried out in collaboration with the University of Pune. Students must work during their free time.

The main objectives of the scheme are:

- To support financially weak & needy students
- To increase understanding of social responsibility and the value of labour.
- To develop work culture in students
- To encourage students for self employment

Working time will be maximum 03 Hrs/Day

Remuneration:
Rs.55/- per hour. Remuneration will be paid monthly.

Duration of the Scheme:
Till 28th February 2025

Students who want to take advantage of the aforementioned scheme should contact the department faculty coordinator.

N.V.Mane
Students Welfare Officer

21/8/25
Dr. (Mrs.) P Malathi
Principal