Page No.:
Date:

Experiment - (1)

→ **Aim** - Consider a student database of SY. class (at least 10 records). Data-base contains different fields of every student like Roll no., Name and SGPA. (array of Objects of class)

a) Design a rollcall list, arrange list of student according to roll no. in ascending order (use Bubble Sort)

b) Arrange list of Students alphabetically. (use Insection Sort)

c) Arrange list of students to find out first ten toppers from a class. (use Quick Sort).

d) Search students according to SGPA. If more than one student having same SGPA, then print list of all students having same SGPA.

e) Search a particular student according to name using binary search without recursion.

→ **Objective** -

1) To study the concepts of array of structure.
2) To understand the concepts, algorithm and applications of Sorting
3) To understand the concepts, algorithm and application of Searching.

→ **Outcome** -

1) Understand linear data structure - array of structure.
2) Apply different sorting techniques on array of structure (Bubble, Insertion and quicksort) with output for every pass.
3) Apply different searching techiques on array of structure (Linear Search, Binary Search) and display the output for every pass.
4) Calculate time complexity.

Page No.:
Date:

→ Theory

1) Structure:

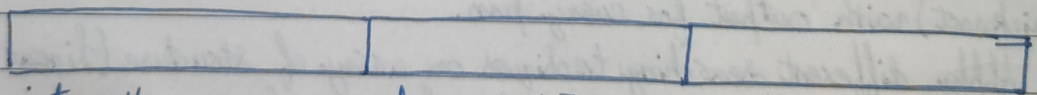
Structure is collection of heterogeneous types of data. In c++, a structure collects different data items in such a way that they can be referenced as a single unit. Data items in a structure are called fields or members of the structure.

- Creating a Structures:

```
Struct student
{
int roll_no;
char name[15];
Float sgpa;
};
```

Here struct is keyword used to declare a structure. Student is the name of the structure. In this example, there are three types of variables int, char and float. The variables have their own names, such as roll_no, name, sgpa.

- Structure is represented as follows:



int roll_no	char name[15]	float sgpa
2 bytes	15 bytes (1 byte of each char)	4 bytes

- Declaring structure variable

A "Structure declaration" names a type of specifies a sequence of variable values that can have different types

Struct student S1, S2, S3;
Here three structure variables S1, S2, S3 are defined by the structure of student. All three structure variables contain the three members of the structure student.

- Referencing Structure Members with the Dot Operator:

Given the structure student and S1 is variable of type struct student we can access the fields in following way:
S1.roll_no
S1.name
S1.sgpa.

Hence, here the structure name and its member's name are separated by the dot (.) operator.

- Pointers to Structures

You can define pointers to structures in the same way as you define pointer to any other variable -

struct student *ptr;
Suppose if we have structure variable declared as:
struct student S1;
then ptr can store the address of S1 by:
ptr = &S1;

Referencing a Structure Member with -> (arrow operator)

Using ptr we can access the fields of S1 as follow:

ptr -> name or (*ptr).name

Page No.:
Date:
youva

Because of its clearness, the → operator is more frequently used in programs than the dot operator.

2) Array of Structure:

The array and Structures can be combined together to form complex data object. To declare an array of structures, you must first define a structure & then declare an array variable of the type.
for eg.

```
Struct student s[20];
```

This creates 20 sets of variable that are organized as defined in the structure student.
To access a specific structure, index the array name, for eg, to print the name of 4th student, we write

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

3) Sorting:

Sorting is a process of ordering or placing a list of elements from a collection in some kind of order. It store data in sorted order. Sorting can be done in ascending or descending order.

a) Bubble Sort

It is the simplest sort method which performs sorting by repeatedly moving the largest element to the highest index of the array. It comprises of comparing each element to its adjacent element and replace them accordingly.

- **Analysis:**

In this sort n-1 comparisons take place in 1st pass, n-2 in 2nd pass, n-3 in 3rd pass, and so on.
Therefore total no. of comparisons will be
$(n-1)+(n-2)+n-3)+---+3+2+1$
This is an Arithmetic series in decresing order

Page No.:
Date:

Sum = $n/2[2a + (n-1)d]$
Where, n = no. of element in series,
a = first element in the series
d = diff bet^n second element & first element.
Sum = $(n-1)/2[2*1 + ((n-1)-1)*1]$
= $2(n-1)/2$
which is of order n_2 i.e $O(n^2)$
The main advantage is simplicity of algorithm and it behaves as $O(n)$ for sorted array of element.
Additional space requirement is only one temporary variable.

- Working of Bubble Sort Algorithm.

We take an unsorted array for eg.

Bubble Sort takes $o(n^2)$ time so we're keeping it short & precise.

13	34	28	36	10

Bubble Sorting will start from the initial two element, comparing them to check which one is greater.

13	34	28	36	10

Here, 34 is greater than (13) (34 > 13), so its already sorted, so now compare 34 with 28.

13	34	28	36	10

Here, 28 < 36, so, Swapping is required. After swapping new array will be-

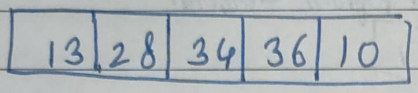
13	28	34	36	10

Now, compare 34 & 36

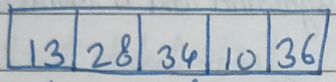
13	28	34	36	10

Here, 35 > 32, So, there is no swapping required.

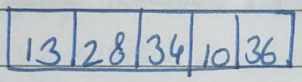
Now, the comparison will be in between 36, 10



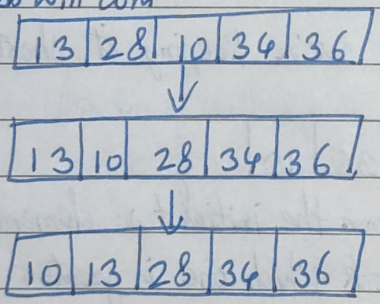
Here, 10 < 35 that arent sorted, So, swapping is required.



We find that we have reached the end of the array. After one iteration, the array should look this



Same process will cont^n



Hence, there is no swapping required, so the array is completely sorted.

6) Inserting Sort

As the name suggests, insertion sort inserts each element of the array to its proper place. It is a very simple sort method which is used to arrange the deck of cards while playing bridge.

The same approach is applied in insertion sort. The idea behind the sort is the first take one element, iterate it through the sorted array. Although it is simple to use, it is not appropiate for large data sets as the time complexity of insertion sort in the avg case & worst case is $O(n^2)$, where n is the no. of items.

- Working of Insertion Sort.

For ex.

12	30	26	8	33	16

First, the first two element are compared in insertion sort in ex 30 is greater than 12. that means both element in order so, now 12 is stored in a sorted sub array.

12	30	26	8	33	16

No next 2 element are compare
these 26 is smaller than 30. So 30 is not at correct position. Now swap 30 with 26. Along with swapping, insertion sort will also check it with all element in the sorted array.
For now, the shorted array has only one element i.e. 12. So 26 is greater then 12. Hence the array will remain sorted after swapping.

12	26	30	8	33	16

Move forward to next element
Both 30 & 8 are not sorted so, swap

12	26	8	30	33	16

After swapping element 26 & 8 are unsorted, so swap them.

12	8	26	30	33	16

Now element 12 & 8 are unsorted, so swap them

8	12	26	30	33	16

Now, the sorted array has 3 items 8, 12, 26. Move to next.

8	12	26	30	33	16

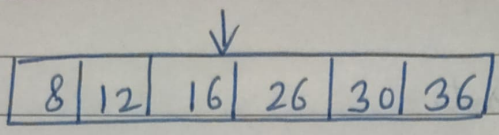
16 is smaller then 33 So, swap them

8	12	26	30	16	33

↓

8	12	26	16	30	33

↓



Now, the array is completely sorted.

c) Quick Sort

Quick sort is the most optimized sort algorithm which perform sorting in O(n log n) comparisons. Like merge sort, quick sort also work by using divide & conquer approach.

- Working of Quick Sort algorithm
  1. Choose the highest index value has pivot
  2. Take two variable to point left & right of the list excluding pivot
  3. Left points to low index
  4. Right points to the high
  5. While value is less than pivot move right
  6. While value at right is greater than the pivot move left
  7. If both step ⑤ & ⑥ does not match swap left & right
  8. If left ≥ right, the point where they met is new pivot.

d) Searching.

Searching is the process of finding some particular element in the list

(i) Linear search

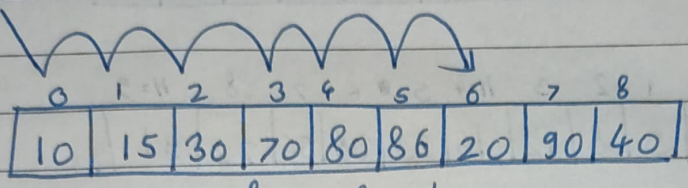
Linear search is the simplest searching algorithm that searches for an element in a list in sequential order. We start at one end and check every element until the desired element is not found.

- Analysis:
  Time complexity $O(n)$
  Space complexity $O(1)$

Page No.:
Date:

Example-

Search 20



Linear Search.

(ii) Binary Search

Binary Search is the search technique which works efficiently on the sorted lists. Hence, in order to search an element into some list by using binary search ~~techn~~ technique, we must ensure that the list is sorted.
Binary Search follows divide & conque approch in which, the list is divided into two halves & the item is compared with the middle element of the list. If the match is found then, the location of middle element is returned otherwise, we search into either of the halves depending upon the result produced through the match.

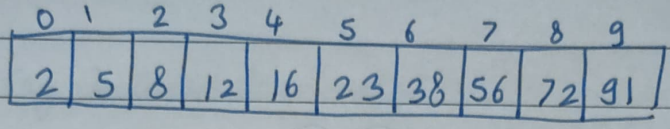
- Analysis:

After 0 comparisons Remaining file size = n
After 1 comparisons Remaining file size = $n/2 = n/2^1$
:
:
After K comparisons Remaining size = $n/2^K$

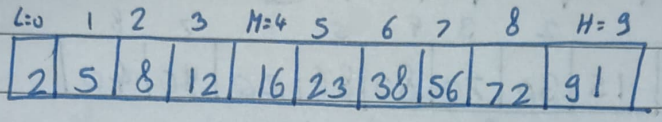
The process terminates when no more partitions are possible i.e remaing file size = 1 $n/2^K = 1$
$K = \log_2 n$
Thus, the time complexity is $O(\log_2 n)$ which is very efficient.

- example

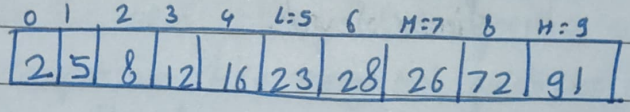
Search 23



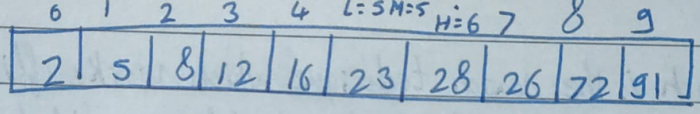
23 > 16
take 2nd half



23 > 56
take 3rd half



found 23,
returns.



→ Algorithm / Pseudocode:

- Create a structure

create_database(struct student s[])
Step 1: Accept how many records user need to add, say, no of records as n
step 2: For i = 0 to n-1
i. Accept the record & 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 field s.roll_no, s.name, s.sgpa
step 2: End for
step 3: Stop

- Bubble Sort according to sort to roll no.
BubbleSort(struct 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
20
ii). swap (s[i], s[i+1])
iii) End if
Step 3: End for
Step 4: End for
Step 5: Stop

- 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[j] to s[j+1]
b. Decrement j
iv. End while
Step 2: Assign key to s[j+1]
Step 3: End for
Step 4: End of insertion Sort

- 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

Page No.:
Date:

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 index of starting element
// & 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)

Step 2: End if
Step 3: End of quicksort

21

- Algorithm of line are search to search student with sgpa given & display all of them

Linear_search (Struct student s[], float key, int n)

// Here s is array of structure student, key is sgpa of student be searched and
// display, n is total no. of student in record.

Step 1: Set i to 0 & flag to 0
Step 2: While i<n
    a. 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

- Algorithm for Binary Search to search student having given string in their names Binary_Search (S, n, key)

// 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 (strcmp (key, S[mid].name) < 0
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.

Page No.:
Date:

Validation:

- Limit of the array should not be -ve and should not be cross the lower & upper bound.
- Roll no. should not repeat, should not -ve
- Name should only contain alphabets, space and .
- Should not allowed any other operations before the input list is entered by the user.
- Before going to (binary search) records should be sorted according a names.

Test cases:

- Sorting

Four test cases:

22

i. Already sorted according to the requirement
ii. Sorted in reverse order
iii. Completely Random list

Expected output/analysis is.

i. Test algorithm for above test cases
ii. Analyze the algorithm based on no of comparisons & swapping/shifting required.
iii. Check for sort stability factor
iv. No of passes needed
v. Best/average/worst case of the each algorithm based on above on above test case
vi. Memory space required to sort.

Searching:

- Find the max & min comparison required to search element
- Calculate how many comparison are required for unsuccessful search

Application:- Useful in managing large amount of data.