Name : Bhavana Bafna
Class : SE-9
Batch : C-9
Subject : Data Structures and Algorithms (DSA)
Topic : Assignment - 1
Roll no. : 23109

1) Title : Assignment 1 : Searching and Sorting

2) Aim : To implement Searching and Sorting on Array as a linear Data structure.

3) Problem Statement : Consider a student database of SE IT class (at least 10 records). Database contains different fields of every student like Name, Roll no., Name and SGPA. (Array of structures)

a) Design a roll call list, arrange list of students according to roll no. in ascending order (Use Bubble Sort)
b) Arrange list of students alphabetically. (Use Insertion 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 search without recursion.

4) Objective :
- To study the concepts of array of structure
- Know the concepts, algorithms and applications of Sorting.
- Understand the concepts, algorithms and applications of Searching.

5) Outcome :
- Understand linear data structure - array of structure
- Apply different sorting techniques on array of structure (Bubble, Insertion and Quick Sort) and display o/p of every pass
- Apply different searching techniques on array of structure (Linear Search, Binary Search) and display the output for every pass.
- Calculate Time complexity.

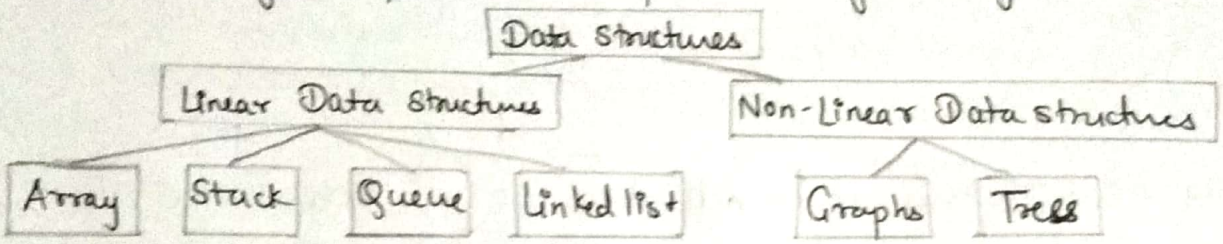
6) Theory:

1) Linear Data structures -

- Those data structure where data items are organized sequentially or organized linearly one after another is called linear data structures.

Linear Data Structures → Stacks
Linear Data Structures → Queues

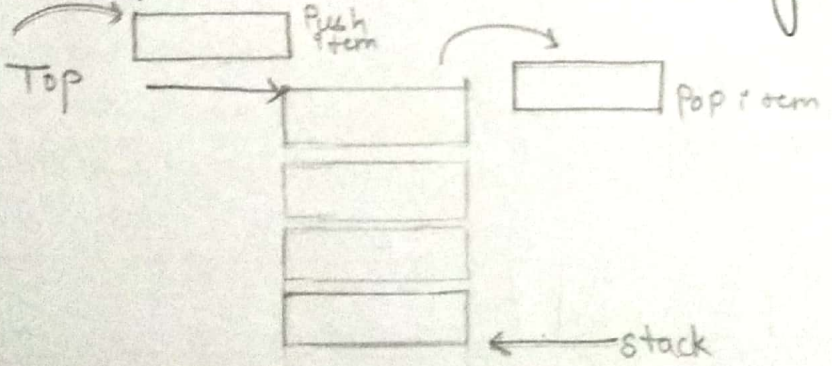
- Elements can be ~~traversed~~ transversed one at a time and only one element is reached while traversing.
- These are easy to implement as computer memory is designed in same fashion.



2) Examples:

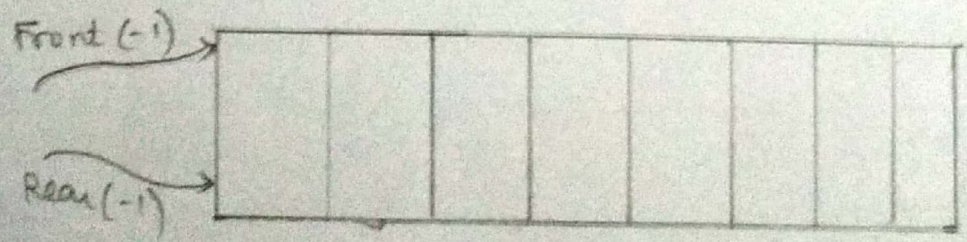
Stacks - A LIFO (Last in First out) data structure where element that added at last will be treated as first.

- Addition and deletion is allowed at only one end, i.e., on TOD.



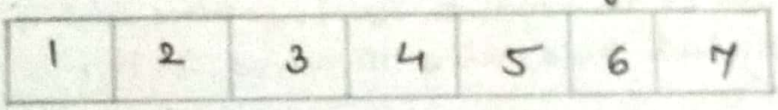
Queues - It is a data structure in which addition is allowed at one end, i.e., 'rear' and deletion is allowed at 'front'.

- It is FIFO Data structure: ~~First~~ in ~~First~~ out.



9) Array:

- Array is a collection of items stored at contiguous memory locations and elements can be accessed randomly using indices of an array.
- They are used to store similar type of elements as in data type must be same for all elements.
- They can be used to store collection of primitive data types such as int, float, double, char, etc of any particular type.
- It can also store derived data types such as structures, pointers, etc.



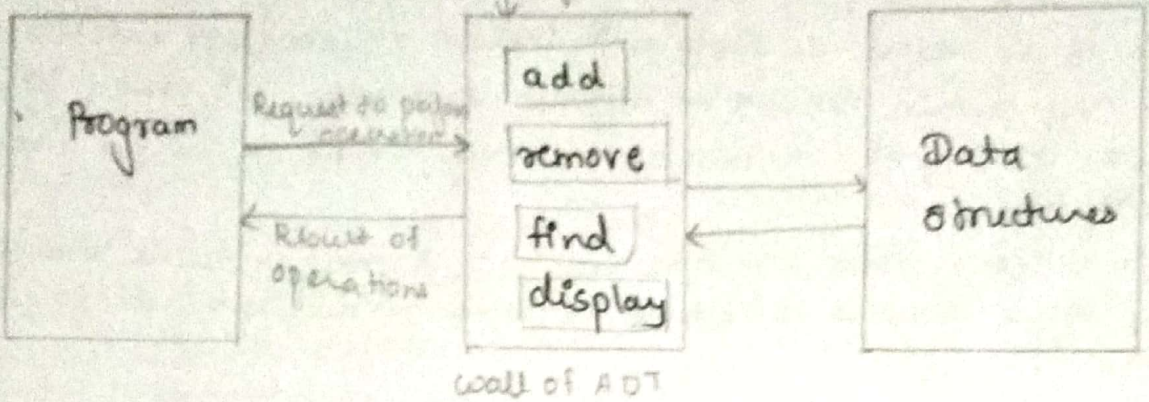
Array length: 7
First Index: 0
Last Index: 6

Need of Array: We can use normal variables (x, y, z ---) when we have small number of objects, but if we want to store a large number of instances, it becomes difficult to manage them with normal variables. The idea of an array is to represent many instances in one variable. To store multiple values.

Array declaration: `int arr[10]`
`int arr[] = {1, 2, 3, 4}`

4) Abstract Data Types (ADT):

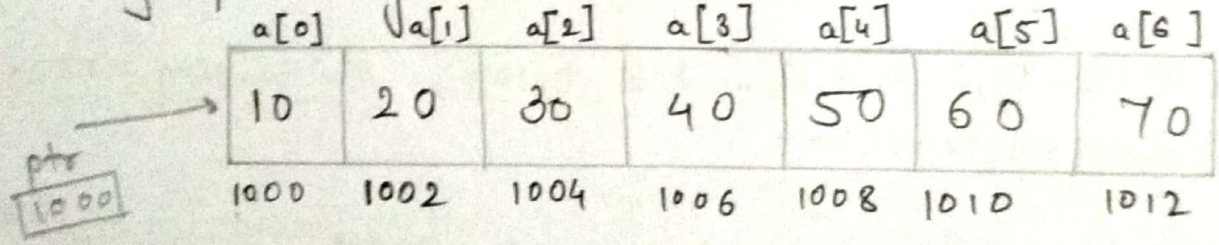
- It is a type (or class) whose behaviour is defined by a set of value and set of operations.
- The properties of ADT are its data (representing internal state of each object) and the behaviours of an ADT are its operations or functions (operations on each instance).



Structure of ADT:
- Name of ADT
- Types represented in collection of data type called as Data Object.
- Each Dataobject has previously been defined in an ADT.
- Functions that operate on Data (specify domain)
- Pre-conditions for any function.
- Post-conditions for any function.

5) Graphical Representation of an Array:

Memory map of array:



Array Terminology:

Base Address - Address of 0th element.

Pointer - Pointer points to whole array instead of one element. In above example it is pointing to base address. It is useful in multidimensional arrays.

Index - It maps the array value and always starts from 0.

Subscript - Allows to identify an element of array. It is usually placed in brackets. For eg: a[5].

Address calculations in Array:

Base address + (size of datatype) x (index) = Address Calculations.

- Base address plays an very important role here, i.e., a[0] → & a[0].

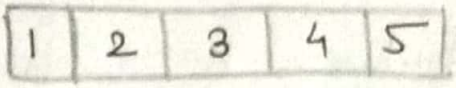
6) Array and structure in C++:

A structure is a data type in C++ that allows a group of related variables to be treated as single unit instead of seperate entities. A structure may contain elements of different data types - int, char, float, double, etc. It may also contain an array, it is called an array within structure.

Array is a linear data structure in C++. Array contains elements of same type whereas structure contains variables of different types.

```
Struct student A     //Initialize a structure
    = {1, 'A', {98.5, 77}};
display(A);          // display structure
```

Eg:



1000 1002 1004 1008 1010

→ continous memory allocation.

7] Declaration, Initialization, input, output of array:

- **Declaration:** int x[5]; char ch[5]; float f[4];
- **Initialization:** int x[5] = {1, 2, 3, 4, 5};
- **Input:**
  ```
  for i → 5 to 1
      Read x[i]
  ```
- **Output:**
  ```
  for i → 5 to 1
      Display x[i]
  ```

8] Applications of Array:

- Arrays are used to implement vectors and lists which are important of C++ STL.
- Arrays are also used to implement stack and queues.
- Trees also use array implementation.
- Matrices which are an important part of mathematical library is implemented using arrays.
- Adjacency list implementation of graphs uses vectors which are implemented using array.
- DS like heap, map and set use binary search tree and balanced binary trees are implemented using arrays.
- Arrays are used to maintain multiple variables with same time.
- CPU scheduling algorithms use implemented using arrays.
- All sorting algorithms use arrays as its core.

9] Basics of structure data type in C++:

Structure is collection of variables of different data types under a single name.

```
struct Person
{
  char name[50];
  int age;
  float salary
}
```

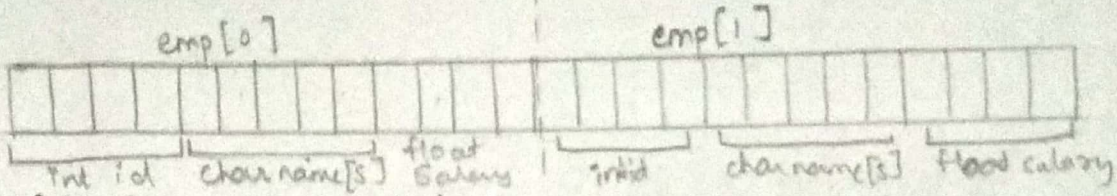
Data Members of structure

The struct keyword defines a structure type followed by an identifier (name of the structure)
Then inside curly braces one or more members of that structure are declared.
We can also include function inside a structure definition.

10] Array of structure:

An array of structures can be defined as the collection of multiple structures variable where each variable contains information about different entities.

Eg:
```
struct employee
{
  int id;
  char name[5];
  float salary;
};
sturt employee emp[2];
```



size of (emp) = 4+5+4 = 13 bytes
size of (emp[2]) = 26 bytes

Input : for i = 0 → 5, i++
Read emp[i],

Output : for i = 0 → 5, i++
Display emp[i].

10] Assignment one database definition of your choice:

```
// structure defined
struct student
{
 string name;
 int roll;
 int total;
 int marks[5];
 float per;
 int dd, mm, yy};

class student
{
 student s[15]; // Array of structure
 public; // declaration of methods
};
```

2] Basics of Searching and Sorting:

Searching → Searching refers to finding an item in an list given contain conditions.

Types of Searching techniques:

- Linear Search or sequential search
- Binary Search
- Fibonacci Search
- Recursive Binary Search
- Sublist Search
- Exponential Search
- Jump search

Sorting → Sorting refers to re-arranging all the elements in array in either ascending or descending order.

Types of Sorting Techniques:

- Insertion Sort
- Bubble Sort
- Quick Sort
- Merge Sort
- Heap Sort
- Selection Sort
- Radix Sort
- Bucket Sort

3] Applications of Searching & Sorting:

Sorting:
- Commercial computing.
- Keeping data sorted helps in efficiently searchings.
- Numerical computations
- Operations Research

Searching:
- Dictionary Clients
- Indexing Clients
- Sparse vectors & matrices
- System symbol table

7] ALGORITHM / PSEUDOCODE :

1) Linear Search -

Concept : This is simple method of searching. In this method, the element is searched in sequential manner, hence the search is called sequential search.
This algorithm can be applicable on both sorted and unsorted list.

Example: Consider a simple array arr[] having 5 elements which are not sorted.

```
int arr[] = {31, 55, 27, 16, 49}
```

- We have to seach 49.
- We will see step by step process of linear search in unsorted array.

Element :	31	55	27	16	49
Position:	0	1	2	3	4

49 — search element

First, in an unsorted array, 0th location value is compared with the element to be searched, if found equal, then number is found, otherwise element is compared with values at further locations.

Comparision b/n : 0th element : 31 and 49
Conclusion : 31 != 49
Result : Compare next element.

Element :	31	55	27	16	49
Position:	0	1	2	3	4

49 — search element

Comparision b/n : 1st element 55 and 49
Conclusion : 55 != 49
Result : Compare next element

Element:	31	55	27	16	49
Position:	0	1	2	3	4

49 — seach element

Comparision b/n : 2nd element 27 and 49
Conclusion : 27 != 49
Result : Compare next element

Element :	31	55	27	16	49
Position :	0	1	2	3	4

49
search element

Compaision b/n : 3rd element 16 and 49
Conclusion : 16 != 49
Result : Compare next element

Element :	31	55	27	16	49
Position :	0	1	2	3	4

49
search element

Compaision b/n : 4th element 49 and 49
Conclusion : 49 = 49
Result : Print the element and stop the search

- Pseudocode :

// a[0,n-1] is an array of n elements. Key is the element being searched.

```
Linear search (a, n, key)
Begin
    for i=0 to n-1
        if a[i] = = key then
        return i;    // returning index of array
        end if
End for
    return -1; // key not found
End.
```

- Analysis :

Worst case - When the element 'key' is not present in array the search is said to be coorst case.
- Worst case frequency = O(n)

Eg:
```
int a[] = {5,1,6,2,3};
int key = 9;
for(i=0; i<5; i++){
 if(a[i] = = key)
 return(i); }
 return -1;
```

The above code returns (-1) as no element '9' is present in a[].

Average case - The average case frequency of linear search is when the element is present in list. It is given as O(n)
- The complexity depends on exact position of 'Key' element in array.

Average case time: $\frac{\sum_{i=1}^{n+1} O(1)}{n+1} = O\frac{((n+1) \times (n+2)/2)}{n+1} = O(n)$

Best case - If element 'Key' is found in array of first position then search said to be best case.

---

2) Binary Search:

Concept - Binary search is a fast searching technique.
- This algorithm can be applied on only sorted list.
- Mid = $\frac{\text{lower} + \text{upper}}{2}$, used to find mid of array

Example: - Consider an array arr[7] having 7 elements
- Int arr[] = {5, 17, 21, 25, 47, 73, 92}
- Search element: 17

	L			M			U
Element:	5	17	21	25	47	73	92
Position:	0	1	2	3	4	5	6

Here, lower = 0, upper = 6

Mid = $\frac{\text{lower} + \text{upper}}{2} = \frac{0+6}{2} = 3$

Check a[5] == 17

No, here 17 < 25, hence search is continued in first part of list.

Then, upper = mid - 1 = 3 - 1 = 2

Again, (new) mid = $\frac{\text{lower} + \text{upper}}{2} = \frac{0+2}{2} = 1$

	L	M	U				
Element:	5	17	21	25	47	78	92
Position:	0	1	2	3	4	5	6

Here 17 == 17, search is successful

- In this algorithm, the given element is compared with the middle element of the list, if these are equal, search is successful.
- Otherwise list is divided into two parts. First part contains elements from first to (mid - 1), and other part contains from (mid + 1) to (end / last) element of list.
- If given element is less than middle element, then continue searching in first part otherwise in second part.
- Repeat the steps till element is found.

• Pseudocode:

```
// a [0, n-1] is array of n elements, key being the element to be searched
Binary search (a, n, key)
Begin
Set start = 0, end = n-1, mid = (start + end) / 2;
while (start <= end && [mid] != key) do
if (key < a [mid]) then
set end = mid - 1; // search in left half
else
 set start = mid+1; // search in right half
end if.
set mid = (start + end) / 2
end while
if (start > end)
    return -1;      // key not found
    return mid;     // return key index.
End.
```

Analysis:

Complexity of binary search algorithm is $O(\log n)$

Time complexity of binary search is $\log_2(n)$

Worst case: When element 'key' is not present in array the search is said to be worst case.

Worst complexity = $O(\log n)$

Best case: When element 'key' is found at first position then it is best case.

Best complexity = $O(1)$

Average case: When 'key' is present but not at first position.

Average complexity = $O(\log n)$

3) Explain Bubble sort :

Concept : The algorithm divides the input list into two parts.
- sorted sublist
- unsorted sublist.

- Initially, sorted sublist is empty and unsorted sublist is entire input sublist.
- In Bubble sort, every element is compared with its next element and then swapping is performed, if first element is greater than second.
- After completion of first iteration largest element in given list is placed in sorted sublist.
- After completion of second iteration, second largest element in given list is placed in sorted sublist.
- The process is repeated in unsorted list until all elements get sorted.

• Example:

Now, consider an array arr[5] having 5 elements.
int arr[] = {31, 55, 27, 16, 49}

- Iteration 1 :

Element:	31	55	27	16	49
Position	0	1	2	3	4

Comparision b/n : 0th element 31 and 55 (1st element)
Conclusion : 31 < 55
Result : No Interation

- ~~Iteration 2~~:

Element :	31	55	27	16	49
Position :	0	1	2	3	4

Comparition b/n : 1st and 2nd element (55 & 27)
Conclusion : 55 > 27
Result : They are interchanged.

- ~~Iteration 3~~.

Element :	31	27	55	16	49
Position :	0	1	2	3	4

Comparison b/n : 2nd element 55 and 3rd element 16
Conclusion : 55 > 16
Result : They are interchanged

Element :	31	27	16	55	49
Position :	0	1	2	3	4

Comparision b/n : 3rd element 55 & 4th element 49
Conclusion : 55 > 49
Result : They are interchanged

- Iteration 2 :

Element :	31	27	16	49
Position :	0	1	2	3

55
4

Comparision b/n : 0th → 31 & 1st → 27
Conclusion : 31 > 27
Result : They are interchanged

Element :	27	31	16	49
Position :	0	1	2	3

55
4

Comparision b/n : 1st → 31 & 2nd → 16
Conclusion : 31 > 16
Result : They are interchanged

Element :	27	16	31	49
Position :	0	1	2	3

55
4

Comparision b/n : 2nd → 31 & 3rd → 49
Result : 49 > 31
Conclusion : Not interchanged.

- Iteration 3 :

Element :	27	16	31
Position :	0	1	2

49	55
3	4

Comparision b/n : 0th → 27 & 1st → 16
Result : 27 > 16
Conclusion : They are interchanged

- Analysis:

Worst case :- Bubble sort is not data sensitive. Hence worst case time complexity is $O(n^2)$.
- The bubble sort will give worst case if data sorted in opp. order than which order is required.

Average case :- Average case is what data is sorted partially
Time complexity $O(n^2)$

Best case :- Sort gives best case if data is sorted in first case itself.
Time complexity = $[\Omega(n)]$

4] Insertion Sort:

It is a simple sorting algorithm. The array is virtually split into a sorted & unsorted part.
- Values from the unsorted part are picked & placed at correct position in sorted part.

- Pseudocode:

```
Insertion_sort ( )
{int array, j, n
  for (int i=1 to n)
   { key = a[i]
     j = i - 1
  while (j > 0 && a[j] > key)
  {
    a[j+1] = a[j]
     j = j - 1 }
  a[j+1] = key }
  for (int i=0 to n)
  {cout << a[i]
  }
```

Element :	16	27	31	49	55
Position :	0	1	2	3	4

Comparision : 1st → 27 & 2nd → 31
Result : 31 > 27
Conclusion : No interchange.

- Interation 4:

Element :	16	27	31	49	55
Position :	0	1	2	3	4

Comparision : 0th → 16 & 1st → 27
Result : 27 > 16
Conclusion : No interchange.

∴ Sorted array = Element :

Element :	16	27	31	49	55
Position :	0	1	2	3	4

- Pseudocode:

```
// a[0,n-1] is an array of n elements
// temp is a variable to facilitate exchange.
Bubble sort (a,n)
Begin
   for k=1 to n-1   // increament by 1  // k for pass
      for j=0 to n-k-1    // j for comparision
         if a[j] > a[j+1] then
            set temp = a[j];
            set a[j] = a[j+1];
            seta[j+1] = temp;
         end if
      end for
   end for
   end for
End.
```

Thus, the total no. of movement of data for sorting using insertion sort $O(n^2)$.

worst case = $O(n^2)$

Best case - When array is already sorted then it is best case.

5] Quick sort:

Quick sort is a devide and conquer algorithm. It picks an element as pivot & partitions the given array around picked pivot.
The key process in quick sort is partition

Best case running time : $O(n \log_2 n)$

Algorithm: Given array of 'n' element (eg. integer)
- If array only one element, return
- Else :- pick one element to use as pivot.
  - partition elements into 2 sub-arrays.
    - Elements less than or equal to pivot.
    - Elements greater than pivot.
  - Quick sort two sub arrays
  - Return results

Pseudocode:

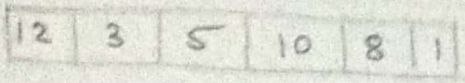
```
void quick sort (int a[], int lower, int upper)
{ int i
 if (upper > lower)
{ i= split (a, lower, upper)
  quicksort (a, lower, i-1)
  quicksort (a, i+1, upper) }
}
int split (int a[], int lower, int upper) {
int i, p, q, t
p= lower +1
q = upper
i = a[lower]
while (q >= p) {    // 1
  while(a[p]<i) p++   // 2
     while (a[q]>i) q--  // 3
```

How Insertion sort work -

Let array { 12, 3, 5, 10, 8, 1 } which is unsorted

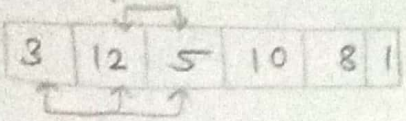
When insertion sort will work -

Pass - 1 :



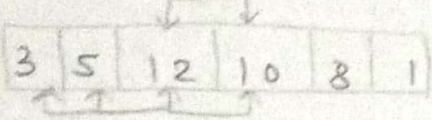
→ Compare 2nd element to 1st before it & sort

Pass - 2 :



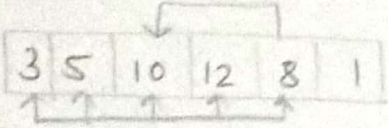
→ Compare 3rd element to all elements before it & sort

Pass - 3 :



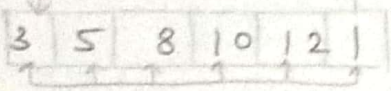
→ Compare 4th element to all elements before it & sort

Pass - 4 :



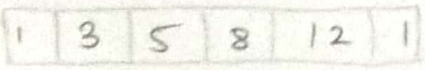
→ Compare 5th elements to all elements before it & sort

Pass - 5 :



→ Compare 6th element to all elements before it & sort

Pass - 6 :



→ Sorted array

- Analysis -

For loop of lines (3 - 4) will be executed $\sum_{i=1}^{n-1} i$ times ~~until~~ its worst case behaviour. If the numbers to be sorted are initially in descending order then loop of lines (3 - 4) will make i iteration during pass i.

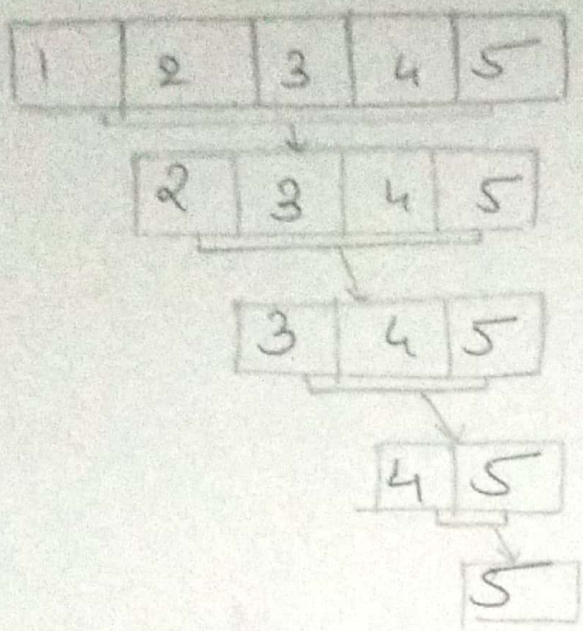
element → 20, 18, 8, 6, 4, 2, 1

Element	Pass	Position moved
10, 20, 8, 6, 4, 2, 1	1	1
8, 10, 20, 6, 4, 2, 1	2	2
6, 8, 10, 20, 4, 2, 1	3	3
4, 6, 8, 10, 20, 2, 1	4	4
2, 4, 6, 8, 10, 20, 1	5	5
1, 2, 4, 6, 8, 10, 20	6	6

Best case - Best case is when data of array is already sorted.

Eg

Consider array : {1, 2, 3, 4, 5}



Number of recursive calls = 4.

8] Test cases / validations:

1) Validations:

- Roll numbers should not be negative.
- Roll numbers should not repeat.
- Roll numbers should be in range 23101-23181.
- Marks should be greater than 100.
- Name should only be alphabets, space.
- Should not allow any other operations before user input.
- Data of students should be between 1-15.
- Before binary search, records must be sorted.
- Limit array should not be -ve, should not cross bound.
- No operations before input.

Test cases: A

A) Sorting -

• Bubble Sort:

1) Already sorted according to requirement consider the array - 1,2,3,4,5

Pass	Array Status	Swapped
1	1 2 3 4 5	False

No of passes = 1
Number of comparisions = 4
Number of swaps = 0

2) Sorted in reverse order:

Consider array: 5 4 3 2 1

Pass	Array Status	Swapped
1	4 3 2 1 5	True
2	3 2 1 4 5	True
3	2 1 3 4 5	True
4	1 2 3 4 5	True
5	1 2 3 4 5	False

No. of passes = 5
No. of comparisions = 20
No. of swaps = 10

3) Partially sorted:

Array: 1, 2, 4, 5, 3

Pass	Array Status	Swapped
1	1 2 4 3 5	True
2	1 2 3 4 5	True
3	1 2 3 4 5	False

No. of pares = 3
No. of comparisions = 12
No. of swaps = 2

4) Completely random list:

Array: 5 2 4 3 1

Pass	Array Status	Swapped
1	2 4 3 1 5	True
2	2 3 1 4 5	True
3	2 1 3 4 5	True
4	1 2 3 4 5	True
5	1 2 3 4 5	False

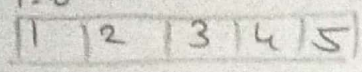
No. of passes = 5
No. of comparisions 20
No. of swaps = 8

2) Searching

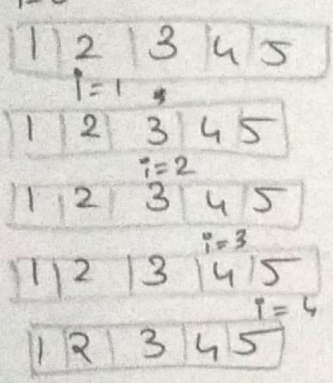
Linear Search:

Array = {1, 2, 3, 4, 5}

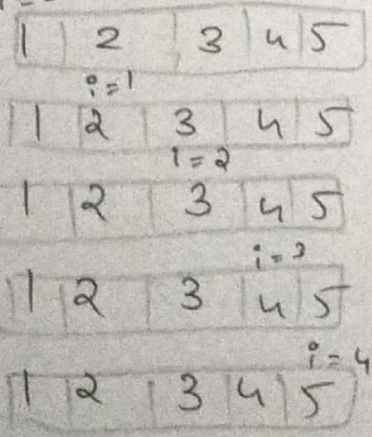
1) Key = 1

Array Pointer(i)	o/p
i=0 	1 found at index 0

2) Key = 5

Array pointer(i)	o/p
i=0 	5 found at index 4

3) Key = 6

Array pointer(i)	o/p
i=0 	Element not found

2) Test Cases:

Sorting: 4 test cases - Already sorted according to the requirement
- Sorted in reverse order
- Partially sorted
- Completely random list.

Searching - Search element at first, middle, last position.
- Search element is not in the list.

9] Program:

Print out / soft copy is attached.

10] Results / output:

Including test cases, validations and valid based results

11] Conclusion:

Add the comparitive study and analysis in table format.

1) All sorthing algorithms: consider following factors for comparision: Time complexity - best, worst, average
Space complexity
Stable sort or not
Methods used for sorting.

2) Searching Algorithms: Consider following factors for comparision:
1) Time complexity: best, average, worst
2) Input list constraint.
3) Methods used for sorting.

Sort / Factor		Bubble Sort	Insertion Sort	Quick Sort
Time Complexity	Best	$O(n)$	$O(n)$	$O(n \log(n))$
	Avg	$O(n^2)$	$O(n^2)$	$O(n \log(n))$
	Worst	$O(n^2)$	$O(n^2)$	$O(n^2)$
Space Complexity		Constant $O(1)$	Constant $O(1)$	$O(\log(n))$
Stability		Stable	Stable	Unstable
Method used for Sorting		The elements are checked using a loop. If left element is greater than right then they are swapped. This is repeated till we get sorted array.	The 1st element is considered sort. Remaining elements are checked and placed at right position one at a time.	The 1st element of array is considered pivot. Remaining elements are arranged as per pivot. then left and right are sorted separately considering another pivot.

Searching: Consider following factors: Time complexity, best, avg, worst case. Input list constraint method using for sorting and searching.

Factors \ Algorithm		Linear Search	Binary Search
Time complexity	Best	O(1)	O(1)
	Avg	O(n)	O(log n)
	Worst	O(n)	O(log n)
Input List constraints		If searched element is present at last index or not present, then it is worst case	The array needs to be sorted for using binary search.
Method used		The array is transversed from first element to last element. If key is found it is displayed.	The array is sorted. Then, low and high are set. The middle element is checked. According, to middle the process is repeated.