Pimpri Chinchwad Poly.
Mrs. M. M. Mali
Data structure
IT dept.

Ch1 - Introduction to data structures

Data is a collection of numbers, alphabets and symbols combined to represent information.
A computer takes data as input and after processing of data it produces outputs.

There are 2 types of data :—

1) Atomic data :— It is a non-decomposible entity. e.g. : an integer value 523 or character value 'A'.

2) Composite data :— It is a combination of several atomic data and hence it can be further divided into automic data.
e.g :— date of birth (15/3/1984)
where 15 - day of the month
3 - month
1984 - year

Data structure defines a way to store and organize data in a computer, so that it can be used efficiently.
Data structure deals with the knowledge of :-
1) How the data should be organized ?
2) How the flow of data should be controlled ?
3) How a data structure should be designed and implemented ?

Data structures includes arrays, linked list, queues, stacks, trees, graphs and so on.

Data structures are used in areas like compiler design, numerical analysis, operating system, artificial intelligence, DBMS, simulation, graphics etc.

* Defination of data structure :- Data structure is a container who stores the data.
- A data structure is a scheme for organizing data in the memory of a computers.
- Data structure deals with representation of data considering not only the elements stored but also their relationship to each others.
- For writing an efficient program, a proper data structure should be selected.
- A proper data structure should be selected so that the relationship between data elements can be expressed.
- Processing and accessing of data should be efficient.

- Data structure also be defined as an instance of ADT.
- A data structure is a triplet of D, F, A where D :- set of domains
F :- set of operations
A :- set of axioms

* Need of data structure :-

1) A data structure helps us to understand the relationship of one data element with the others.

2) data structure helps us to analyze the data in a logical or mathematical manner.

3) Data structure helps us to store the data and organize in a logical or mathematical manner.

4) Now a day computing problems are complex and large, so there is need to handle large amount of data which is overcome using ds.

To identify a solution for a data processing problem, 4 things have to be identified :-

1) The data elements that are concerned with the problem.

2) The operations that will be performed on thes data elements.

3) methods of storing the data elements in memory to retain the logical relationship between them.

4) The programming language which suits the current requirements.

example : calculate area of circle : —

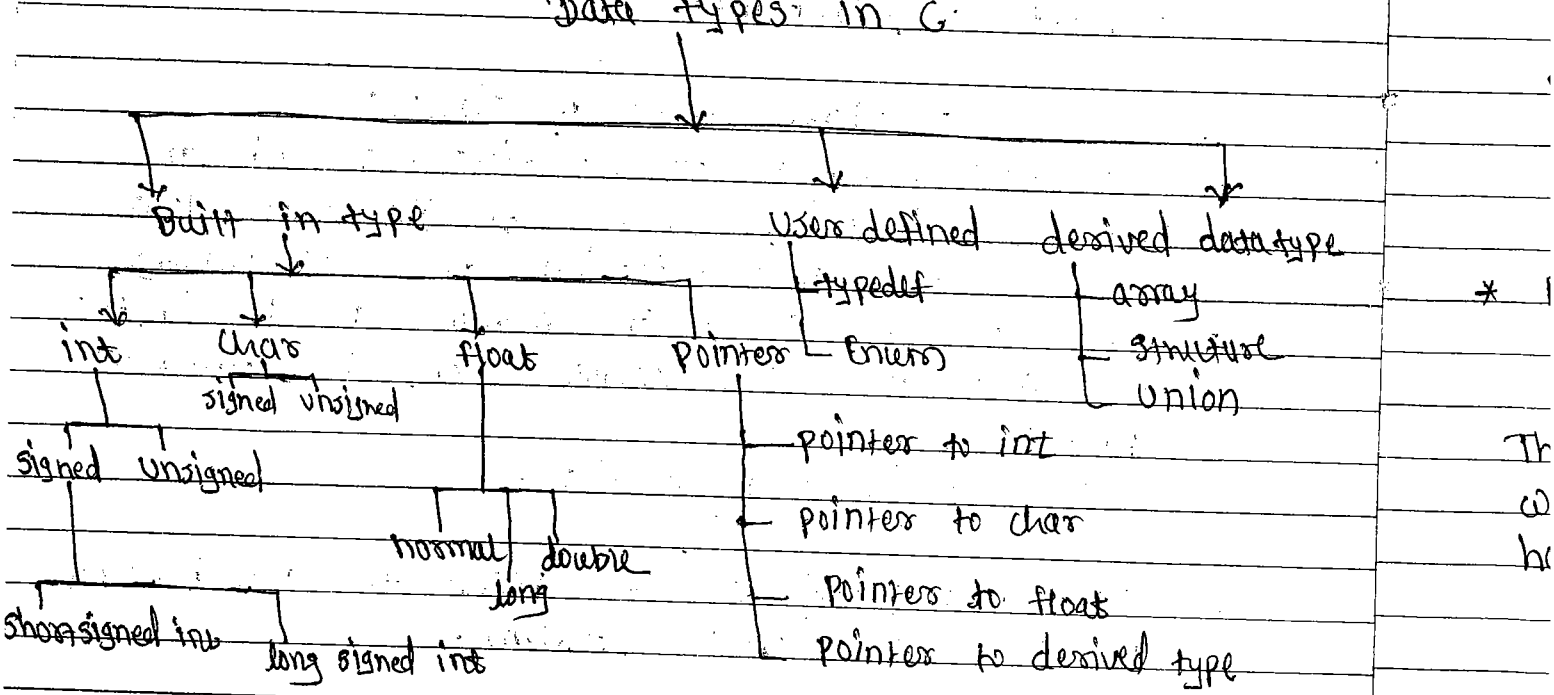
1) The data elements are radius, pi and area.
2) The operations performed on the above data elements are multiplication and assignment.
3) They can be stored in memory as bits / bytes and the user can access them in different forms.
eg :- integer, float etc.

4) Languages like c, c++, pascal etc can be used to solve the above problem.

Data types in 'C' :— A method of interpreting a bit pattern is called as a data type.
Every programming language has its own set of data types.
Data types indicate the type of data that a variable can hold. The data may be numeric or non-numeric in nature.

In 'C', the data types are categorized into following types :—

1) Built in data types
2) Derived data types
3) User defined data types.



* Users Defined Type :- "Type definition" allows users to define an identifier that will represent an existing data type.

eg:- Syntax :- typedef datatype newdatatype ;
where typedef is a keyword for declaring the new data items and datatype is an existing datatype being converted to the new name.

e.g :- typedef float marks
- After above declaration marks can be used for float. i.e. marks phy, che, math
Here marks is another name for floats.

The advantage of typedef is that we can create meaningful datatype names for increasing the readability of the program.

* Enumerated data type :-
Syntax :- enum identifier { value 1, value 2, ... value n

The 'identifier' is a users defined enumerated data type which can be used to declare variables that can have one of the values enclosed within the braces.

e.g :- enum day { mon, tue, wed, thu, fri, sat, sun }
enum day today;

Here variable today which is declared of enum day type can be assigned a value from the set (mon, tue, wed, thu, fri, Sat, sun)
i.e. today = thu;
i.e if (today == mon)
i.e. today = fri

All above 3 statements are valid statements.

* Derived data type - These types are derived or constructed from the basic or fundamental data types.
These data types are array, function, structure, union,

In structure each member has its own storage location, whereas all the members of a union use the same location.

Like structure, union allows us store different data types in the same memory location.
Unions provide an efficient way of using the same memory location for multiple purpose.

Syntax:— union unionTag
{
member1;
member2;
...
} One or more union variables;

e.g:— union data {
int i;
float f;
char str[20];
} data;

Now, a variable of "data" type can store an integer, a floating point number and a string of characters. It means a single variable i.e same memory location, can be used to store multiple types of data.
The memory occupied by a union will be large enough to hold the largest member of the union.
In above example data type will occupy 20 bytes of memory space because this is the maximum space which can be occupied by a character string.

ADT :- Abstract Data Type

Data type is a collection of values and set of operations on these values.

ADT is a logical description of how we view the data and operations that are allowed without regard to how they will be implemented.

ADT combines the description of data and its associated operations on these values.

ADT provides 2 characteristics :—

1) Description of element in term of data types
2) Defines relationship among individual elemen

The ADT is a useful tool for specifying the logical properties of data type without going in any details.

ADT simply tells us "what" has to be don It does not tell "how" to do it i.e. the ADT does not tell :—

1) How to store the elements of the data objects.
2) How to perform the operations.
3) How to enforce the rules while performi the operations.

In short ADT means hiding the information without knowing its implementation.

Types of Data structure :—

There are 2 types of data structure.

— > primitive data structure.

— > Non- primitive data structure [derived]

1) Primitive data structure :—

The int, char, float, pointer are primitive data structure and these data types are available in most programming languages as built in type.

2) Non- primitive data structure :—

These data structure are derived from primitive DS. It is a set of similar or different data items.

Non primitive data structures are subdivided into linear and non- linear DS.

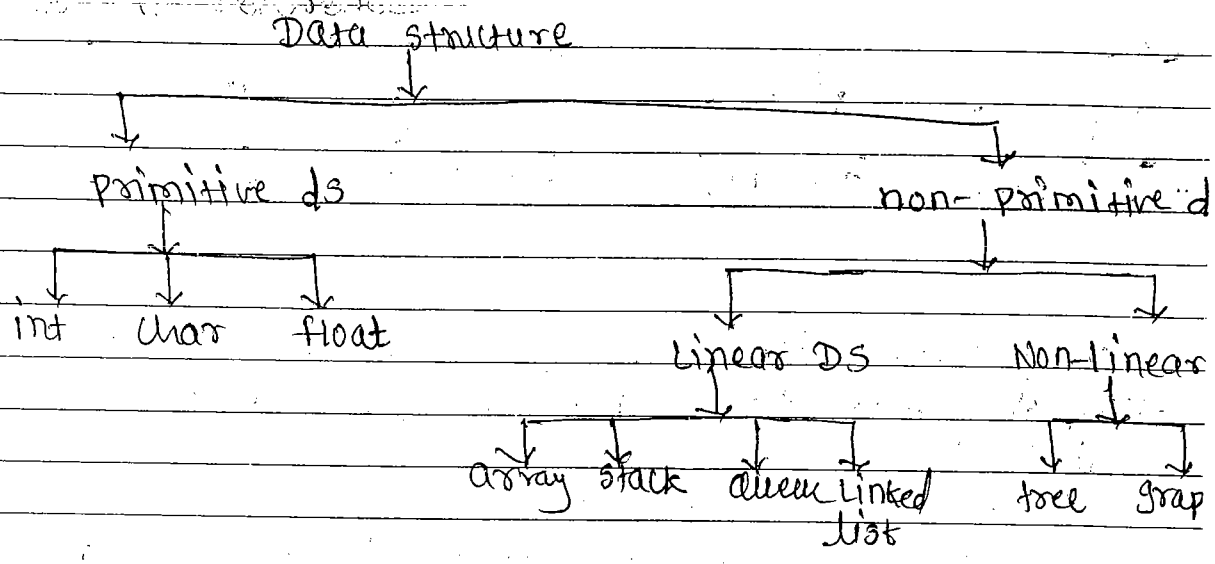
~~Furthermore linear ds consi~~ is converted

Types of linear data structures are :—

Arrays, stack, queue, linked list.

Types of non- linear data structure are :—

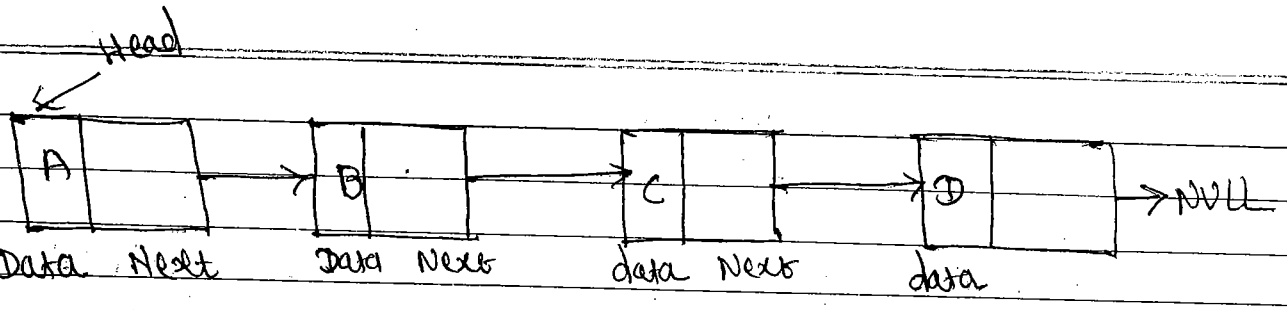
Trees and graph



Linear Data Structure :- Linear means "In a Line"
The elements of a linear data structure are arranged in a line. i.e. in a sequence or list.
Their mapping is one dimensional.
In a linear DS every data element has a unique successor and unique predecessor.
Example :- Array, linked list, stack and queue.

Array :- Array is a collection of data of same data type stored in consecutive memory location and is referred by a common name.

Linked List :- Linked list is a collection of data of same data type but the data items need not be stored in consecutive memory locations.
The elements are linked using pointers.
In linked list each element is a separate object.
Each element [we call it as a node] of a list is comprising of two items — the data and a reference to the next node. The last node has a reference to null.



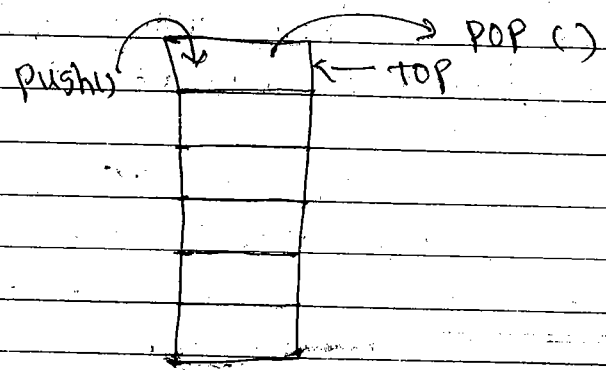
Why Linked list :— Arrays can be used to store data of similar types, but arrays have following limitations :—

1) The size of the array is fixed
2) Inserting a new element in an array of elements is expensive, because room has to be created for the new elements and to create room existing elements have to shifted.

Operations on Linked List :—

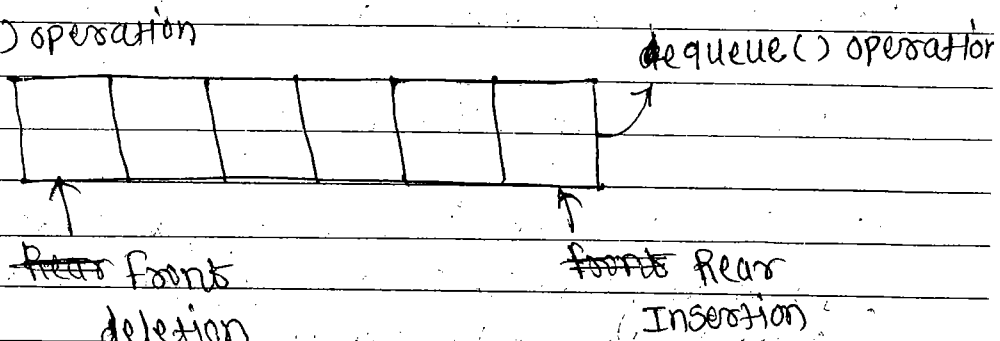
1) Traversing all the elements in a linked list
2) Searching for an element in the linked list
3) Insertion of an element
4) Deletion of an element

Stack :— A stack is a data structure consisting a list of element in which elements can be inserted or deleted at one end which is called as top of the Stack.



- Every time an element is added, it goes on the top of the stack and the only element that can be removed is the element that is at the top of the stack.
- Stack is a LIFO (Last In First Out) structure
- PUSH() function is used to insert new elements into the stack.
- POP() function is used to remove an element from the stack.
- Both insertion and removal are allowed at only one end of stack called TOP.
- Stack is said to be in Overflow state when it is completely full and is said to be in Underflo state if it is completely empty.
- The simplest application of a stack is to reverse a word. You push a given word to stack - letter by letter and then POP letters from the stack.

Queue :- A queue is open at both its ends.



enqueue() is the operation for adding an element into queue.

dequeue() is the operation for removing an element into queue.

— one end is always used to insert data [enqueue] and the other is used to remove data [dequeue]

— Queue follows FIFO [First in first Out] methodology i.e. the data item stored first will be accessed first.

— Operations performed on queue :—

1> Insertion :— Insert or add an element to the queue [Rear end]

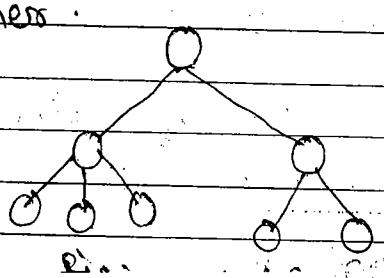
2> Deletion :— Deleting an element from a queue (Front end)

* Non-Linear Data Structure :—

— non linear data structure are used to represent the data containing hierarchical or network relationship between the elements.
— In a non-linear data structure, the elements have a one to many relationship between them.

Types of non-linear DS :— 1> Tree 2> Graph

Tree :— A tree is a widely used data structure that represents a hierarchical relationship among the data elements.
A tree is a collection of nodes in a recursive manner.



leue]
:]

Graph :- Graphs are set of interconnected nodes having a specific value.
— Graph is used to represent data that has relationship between pair of elements not necessary hierarchical in nature.

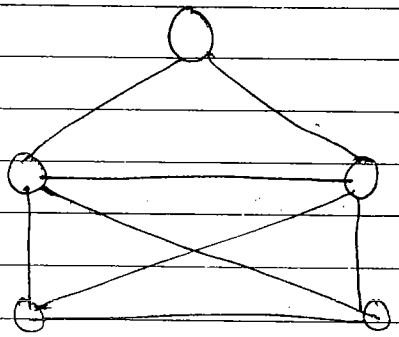


fig :- Graph

* Operations on data structure :—

1) Traversing :— It involves processing each element in the list exactly once.

* eg :— Print the values of all the elements in the array.

2) Searching :— finding any element or the record using a key is called searching.
eg :— find out names of all the students who secured 90 marks in maths.

3) Inserting :— It is used to add new data items in the given list.
eg :— Add the details of a new student who has recently joined the course.

4) Deleting :— It is used to remove a particular data item from the given list.

5) Sorting:- Data items can be arranged in some order like ascending or descending order.

6) Merging:- List of two sorting data items can be combined to form a single list of sorted data items.

* Analysis of Algorithm:- The algorithm can be analyzed by tracing all step by step instruction, reading the algorithm for logical correctness and testing it on some data using mathematical techniques to prove it correct.

* Complexity of Algorithm:- Complexity of an algorithm is a function of size of input of a given problem which determines how much running time and memory space is needed by the algorithm in order to run to completion.

Complexity of an algorithm is stated in terms of time and space.

e.g.

Time complexity:- Time complexity deals with the computing time.

Space complexity:- Space complexity is the amount of memory or storage required for that particular algorithm.

e.g of Space complexity :-

```
void main()
{
    int a;
    float b, c;
    c = a+b
    printf("Result = %d", c);
}
```

∴ Space required = 2+4+4 = 10 bytes

e.g of time complexity :-

```
void main()
{
    int a;          ------ 1
    float b, c;     ------ 1
    c = a+b;        ------ 1
    printf("Result = %d", c);  --- 1
}
```

∴ Total frequency count = 1+1+1+1 = 4

e.g. of time complexity :-

```
void main()
{ int a, n;          ---- 1
  float b, c;        ---- 1
  n = 3;
  for (i=0; i<=n; i++)
  {
    c = a+b+i;       ---- n
    printf("Result = %d", c);  --- n
  }
}
```

Big 'O' notation :- Big 'O' notation is a tool for accessing algorithm efficiency.
Big 'O' notation is used to describe the performance of an algorithm.
The efficiency of an algorithm is expressed as how long it runs in relation to its input.

* Best case, worst case, Average case complexity :-
⇒ many programs do not produce same time complexity in every case.

eg: searching an element in the array with 10 elements.

5	1	9	6	2	11	13	6	7	16

⇒ Element 5 will be found in first attempt.
⇒ Element 16 will require 10 comparisons to find

Best case :- 1 comparison [when the element to be searched is in the beginning]

worst case :- n comparisons [where n is the number of elements and the element to be searched is at the end of the array]

Average case :- number of comparisons will be n/2

Approaches for an Algorithm: —

1) Top down approach 2) Bottom-up approach

Top Down Approach:- In C programming the idea of top down design is done using functions. In C programming where main() starts from top i.e. execution of the program always starts and ends with main().

A 'C' program is made of one or more functions. The top-down method of design is based on decomposing a large problem into sub problems and keep repeating this process till the resulting sub problems are so simpl

Bottom Up Approach:- This is reverse of top down approach. Here, the different parts of the problem are first solved using a programming language and then these pieces of programs are combined into a complete program

* Linear DS	Non-linear data struct
① In linear data structure the data items are arranged in a linear sequence	① In non-linear DS, the data items are not arranged in a linear sequence.
② Example:- Array, linked list	② Example:- Tree, graph

* Homogeneous

In homogeneous data structure all the elements are of same type

e.g:- Array

Non-Homogeneous

In non-homogeneous data structure the elements may or may not be of the same type

e.g. Records

2 Searching and sorting

The process of finding a particular record in a list is called "searching".
Searching is an operation which finds the place of a given element in the list.
Search is said to be successful or unsuccessful depending upon whether the element that is being searched is found or not.

Types of searching :-
1) Linear search / sequential search
2) Binary search

1) Linear / sequential search :-
It is the simplest method for searching.
The element to be found is searched sequentially in the list.
It can be used on a sorted or an unsorted list.

Procedure of linear search :-
1) Initalize key element i.e. the element to be searched in the list / Read the search element from user
2) Compare key element with 0^{th} position element.
If match is found then search is successfull
3) If match is not found then key element is compared with the next element.
4) Repeat the procedure till $(n-1)^{th}$ comparison.
5) After $(n-1)^{th}$ comparison, if match is not

* Advantages of linear search :—
1) It is a simple method and easy to implement
2) It does not require the data to be ordered.
3) It does not require any additional data.

* Disadvantages of linear search :—
1) If the size of the array is large then it consumes time.
2) It needs more space and time.
3) In the worst case, if n is very large, this method is very inefficient and slow.

* Time complexity of linear search :—
1) Best case :— Record / element found at position 1, then number of comparisons = 1.

2) Worst case :— Record / element not present (or at the end of the list)
So number of comparisions = n+1.

3) Average case :— $\frac{n+1}{2}$ comparisons are required to retrieve record.

So Time complexity of linear search of the order n ie. O(n).

* Algorithm for Linear search :—
Step 1 : Start
Step 2 : Read / accept values from user
Step 3 : Repeat for i=0 to N by 1
if A[i] = item then
write : " item found at location "
exit
end if

step 4 write " Item not found "
step 5 : stop

program for linear search

```
void main()
{
  int a[10], i, n, search;
  clrscr();
  printf("\n Enter the size of array : ");
  scanf("%d", &n);

  printf("\n Enter the elements for array :\n");
for (i=0; i<=n-1; i++)
  {
  scanf("%d", &a[i]);
  }

  printf("\n Enter the number to search : ");
  scanf("%d", &search);

  for (i=0; i<=n-1; i++)
  {
  if (a[i] == search)
  {
  printf("%d is present at location %d", search, i);
    break;
  }
  }
  if (i==n)
  {
  printf("%d is not present in array", search);
  }
  getch();
```

Binary search :- Binary search is a fast search algorithm that works efficiently with a sorted list.

In this method, we make a comparison between key and the middle element of the array.

As array is already sorted, this comparison results either in a match between key and the middle element of array.

If key element not matches with the middle position element then the same procedure is repeated on the left half or right half in which the key element (searching number) is to be present.

This search algorithm works on the principal of divide and conquer. For this algorithm to work properly, the data collection should be in the sorted form.

Binary search is a fast search algorithm with run-time complexity of $O(\log 2\, n)$

example of binary search :-

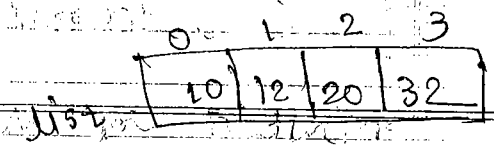
Consider the following list of element and search element 12.

	0	1	2	3	4	5	6	7	8
list	10	12	20	32	50	55	65	80	99

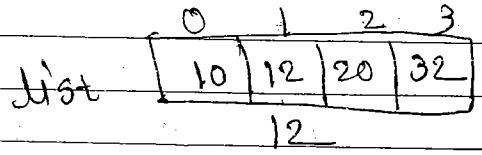
search element 12

Step 1 :- Search element 12 is compared with middle element (50)

So both 12 and 50 are not matching. and 12 is smaller than 50 so we search only in the left sublist (i.e. 10, 12, 20, 32)



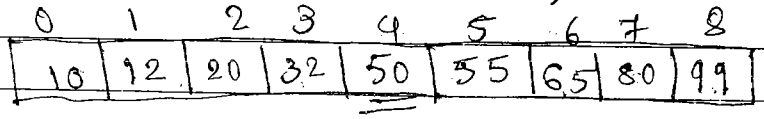
Step 2: Search element 12 is compared with middle element (12)



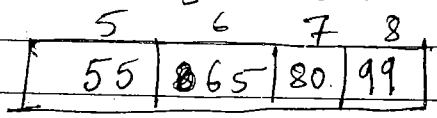
Both are matching, so the result is "Element found at index 1".

Search element 80 :-

Step 1: Search element (80) is compared with middle element (50)

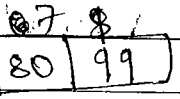


Both 50 and 80 are not matching. And 80 is larger than 50. So we search only in the right sublist (i.e. 55, 65, 80, 99)

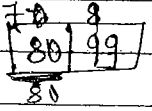


Step 2: Search element 80 is compared with middle element 65

Both 65 and 80 are not matching. And 80 is larger than 65, so we search only in the right sublist (i.e. 80 & 99).



Step 3: Search element is compared with middle element (80)



Steps in Binary search :—

Step 1 :— Read the search element from the user.

Step 2 : Find the middle element in the sorted list.

Step 3 : Compare the search element with the middle element in the sorted list.

Step 4 : If both are matching, then display "Given element found" and terminate the function.

Step 5 : If both are not matching, then check whether the search element is smaller or greater than middle element.

Step 6 : If the search element is smaller than middle element, then repeat steps 2, 3, 4 and 5 for the left sub list of the middle element.

Step 7 : If the search element is larger than middle element, then repeat steps 2, 3, 4 and 5 for the right sublist of the middle element.

Step 8 : Repeat the same process until we find the search element in the list or until sublist contains only one element.

Step 9 : If that element also does not match with the search element, then display "element not found" and terminate the function.

Algorithm for Binary search :-

Step 1 : Start

Step 2 : Initialize the variables beg, end, mid, position

Step 3 : Repeat Step 4 and Step 5 while beg <= end

Step 4 : find the mid = (beg + end) / 2

Step 5 : if A[mid] == val then
position = mid
print position
Go to Step 7
if A[mid] > val then
set beg = mid - 1
else
set beg = mid + 1 ;
end of if
end of loop

Step 6 : if position = -1 then
print " Value is not present in the array "
end of if

Step 7 : Exit

* Advantages of Binary search :-

① If the array size is large then binary search is very much faster than linear search.

② It is more efficient

* Disadvantages of Binary search :-

① Data has to be in sorted order

② This method can only be applied to linear and sequential data structure.

complexity of binary search

① During each iteration the value of mid is calculated as mid = (beg + end)/2

② for binary search maximum number of comparisons for successful and unsuccessful search is given by O(log2 n)

program for binary search

```
void main()
{
int a[10], n, i, item, beg, end, mid, flag=0;
clrscr();
printf("\n Enter size of array :");
scanf("%d", &n);

for(i=0; i<=n-1; i++)
{
scanf("%d", &a[i]);
}

printf("\n Enter element to be searched :");
scanf("%d", &item);

beg=0, end=n-1;
while(beg<=end)
{
mid = (big+end)/2;
if(item==a[mid])
{
flag=1;
break;
}
```

```
else if (item < a[mid])
    end = mid-1;
else
    beg = mid+1;
}
if (flag == 1)
printf("\n Item %d present at position %d",
                                item, mid);
  else
printf("\n Item not present");

getch();
}
```

* Difference between linear and binary searc

Linear search	binary search
1) Data can be in any order	1) Data should be in a sorted order.
2) multidimensional array also can be used	2) only single dimension array is used
3) Time complexity :- O(n).	3) time complexity :- O(log2n).
4) Not an efficient method to be used if there is a large list	4) Efficient for large inputs also.

Sorting :- Sorting means arranging the elements of an array either in ascending or descending order.

Each sorting algorithm may be analyzed on amount of time necessary for running the program and amount of space required for the program.

The amount of time is proportional to the number of key comparisons.

Types of Sorting :-

1) Bubble sort
2) Selection sort
3) Insertion sort
4) Quick sort
5) Radix sort

1) Bubble Sort :- Bubble sort is very simple method.
- This method begins with the 0th element.
- The 0th element is compared with the 1st element.
- If 0th element is found to be greater than the 1st element then they are interchanged otherwise not interchanged.
- In this way all the elements are compared with their next element and are interchanged if required.
- The output of first iteration is the largest element gets placed at the last position.
- Similary in the second iteration, second largest element gets placed at the second last position.

In bubble sort we just need to compare and swap the elements.

- As a result after $(n-1)^{th}$ iteration, the list becomes a sorted list.

Procedure of Bubble Sort :-

1) In the first iteration 0^{th} element is compared with the 1^{st} element, if 0^{th} element is greater then they are interchanged.

2) I^{st} element is compared with the 2^{nd} element if 1^{st} element is greater ~~then~~ then they are interchanged else not.

3) Repeat the procedure until $(n-2)^{th}$ is compared with $(n-1)^{th}$ element.

Advantages of Bubble sort :-

1) It is simple to write and easy to understand.
2) It only takes a few lines of code.
3) The data is sorted in place so there is little memory overhead.
4) Once sorted, the data is in memory and ready processing.

Disadvantages of Bubble sort :-

1) The major disadvantage is the amount of time it takes to sort.

2) The average time increases almost exponentially as the number of table elements increase.

Complexity of Bubble sort :-

1) If there are 'n' number of elements in an array then $(n-1)^{th}$ comparisons are required.

2) The worst case arises when the given array is sorted in reverse order.

3) In this case all the iterations required will be : $(n-1)+(n-2)+\ldots 2+1 = \frac{n(n-1)}{2}$

Complexity of Bubble sort :-

(1) Worst case complexity $= O(n^2)$

(2) Average case complexity $= O(n^2)$

(3) Best case complexity $= O(n^2)$

Algorithm for Bubble sort :-

Step 1 :- Accept n numbers
Step 2 : Repeat step 3 for i = 1 to n-1 by 1.
step 3 : Repeat for j = 1 to n-i by 1
if A[j] > A[j+1], then
set temp = A[j]
A[j] = A[j+1]
A[j+1] = temp

end of if
End of step 3 loop
End of step 2 loop

Step 4 : Write Sorted array : A

Step 5 : stop

Note :- for bubble sort use the swapping tech. by using third variable temp

e.g. swap a = 10 and b = 20 it means after swapping, a = 20 and b = 10

temp = a → 10 it means 'a' value (10) stored in temp

a = b = b → 20 it means b value (20) stored in a

b = temp → b = 10 so that now b becomes 10.

e.g for bubble sort :—

70 40 50 10

here n = 4 where n is the array size

Pass - 1 | iteration 1 :— 70 40 50 10

↑swap↑

40 70 50 10

↑swap↑

40 50 70 10

↑swap↑

40 50 10 70

40 50 10 [70] → result of iteration 1

So now you can observe 70 is the largest no it means in first iteration the largest element gets placed at the last position.

Pass-2 | iteration 2 :- 40 50 10 70

no swap

40 50 10 70

swap

40 10 50 70

no swap

40 10 [50] [70] ← result of iteration 2

now you can see in second iteration we will get the second largest no. i.e. 50

pass 3 | iteration 3 :- 40 10 [50] [70]

now here 50 and 70 are fixed as we get the largest elements

so compare 40 10 50 70

swap

10 40 50 70

no swap

10 40 50 70

no swap

10 40 50 70

it means if the size of n is 4 then iterations required are 3 i.e n-1.

i (outer) = used for passes | iterations [outer loop i]

j used for comparisons [inner loop]

program for bubble sort :-

```
void main()
{
 int a[10];
 int i, j, temp, n;
   clrscr();
printf("\n Enter the array size :");
scanf("%d", &n);

printf("\n Enter array numbers you want Sort:")
for(i=0; i<=n-1; i++)
 {
 scanf("%d", &a[i]);
 }

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

printf("\n Array after sorting :\n");
 for(i=0; i<=n-1; i++)
{
 pf("%d \t", a[i]);
}

   getch();
}
```

swapping technique

Selection sort :- The selection sort is the easiest method of sorting.

Procedure for selection sort :-

1) Find the smallest number in the array.
2) Swap it with the number at the first position.
3) Then find the second smallest number in the array and place it in the second position.
4) Repeat this procedure until the entire array is sorted.

eg. of selection sort

78	34	45	12	88	23	67	56

Scan all the array elements, find the smallest no in the array i.e. 12

78	34	45	12	88	23	67	56

12 is smaller so swap it with 78.

12	34	45	78	88	23	67	56

Again find smallest no from remaining array so now 23 is small.

12	34	45	78	88	23	67	56

23 is smaller so swap it with 34

12	23	45	78	88	34	67	56

Again find smallest no in the remaining array 34 is smaller so swap it with 45

12	23	45	78	88	34	67	56

∴ | 12 | 23 | 34 | 78 | 88 | 45 | 67 | 56 |

Again find smaller no from the remaining array now smaller no is 45 so swap it with 78

| 12 | 23 | 34 | 78 | 88 | 45 | 67 | 56 |

∴ | 12 | 23 | 34 | 45 | 88 | 78 | 67 | 56 |

Again find smaller no in the remaining array now smaller no is 56 so swap it with 88

∴ | 12 | 23 | 34 | 45 | 88 | 78 | 67 | 56 |

∴ | 12 | 23 | 34 | 45 | 56 | 78 | 67 | 88 |

Again find smaller no in the remaining array now smaller is 67 so swap it with 67

| 12 | 23 | 34 | 45 | 56 | 78 | 67 | 88 |

∴ | 12 | 23 | 34 | 45 | 56 | 67 | 78 | 88 |

So in this way the sorted list is:-

| 12 | 23 | 34 | 45 | 56 | 67 | 78 | 88 |

Advantages of Selection sort:-

1) It is simple and easy to implement
2) It can be used for small data sets.

Disadvantages of selection sort:-

1) In case of larger data sets, the efficiency of selection sort drops.

Complexity of Selection sort:-

① Selecting the smallest number requires scanning all n elements. This takes n-1 comparisons. Then swap it with the number at first position.

② finding the second smallest number requires we scan the remaining n-1 numbers. This takes n-2 comparisons.
And so on until last element.

③ Therefore $(n-1)+(n-2)+\cdots+2+1 = \frac{n(n-1)}{2}$

$= \frac{n^2-n}{2}$ $= O(n^2)$ comparisons.

∴ worst case $= O(n^2)$
Avg case $= O(n^2)$
Best case $= O(n^2)$

Program for selection sort

```
void main()
{
int a[10], n, i, j, small, temp;
clrscr();
pf("\n Enter the size of the array:");
sf("%d", &n);
pf("\n Enter the array numbers:");
for(i=0; i<=n-1; i++)
{
pf("\n Enter the array numbers:");
sf("%d", &a[i]);
}
}
```

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

printf("\n The array values after sorting");
for (i=0; i<=n-1; i++)
{
pf("\n %d", a[i]);
}
getch();
}
```

Simple way

OR Second way for selection sort :—

~~Procedure :—~~

~~1~~ to sort the data in ascending order, the 0^{th} element is compared with all other elements.

If 0^{th} element is found to be greater than the compared element then they are interchanged. So that after first iteration, the smallest element is placed at the 0^{th} position.

Procedure :—

1) In the first iteration, 0^{th} element is compared with 1^{st} element, if 0^{th} element is greater ~~than~~ then they are interchanged.

2) If 0^{th} element is not greater than 1^{st} element then they are not interchanged.

3) The process is repeated till the 0^{th} element is compared with rest of the elements.

4) At the end of first iteration 0^{th} element holds the smallest no.

5) Second iteration starts with the 1^{st} element.

6) The process of comparison and swapping is repeated.

7) After $(n-1)^{th}$ interation, the array is sorted.

simple way :-

program for selection sort :-

```
void main()
 {
   int a[10], i, j, temp, n;
   clrscr();
 printf("\n Enter the array size :");
 scanf("%d", &n);

printf("\n Enter array numbers you want to sort\n");

for(i=0; i<=n-1; i++)
 {
 scanf("%d", &a[i]);
 }

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

printf("\n Array after sorting : \n");
 for(i=0; i<=n-1; i++)
{
printf("%d \n", a[i]);
}
```

eg:- of selection sort:-

0	1	2	3	4	5
25	15	4	103	62	9

- 0th element is compared with 1st element.
if 0th element is greater then interchange

Iteration 1:-

25 > 15

So swapp

15	25	4	103	62	9

Again compared 0th element with remaining nos.

15	25	4	103	62	9

15 > 4 so swap

4	25	15	103	62	9

Again compared 0th element with the remaining nos.

4	25	15	103	62	9

4 > 103 no swap

4	25	15	103	62	9

4 > 62 no swap

4	25	15	103	62	9

4 > 9 no swap

o/p:- 0th element is sorted.

Iteration 2:-

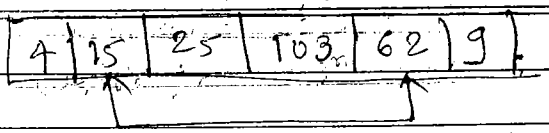
0	1	2	3	4	5
4	25	15	103	62	9

now 1st element is compared with next no. if greater then interchange.

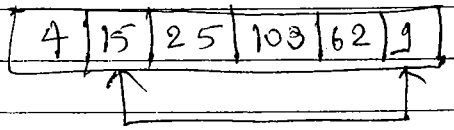
25 > 15 so swap

4	15	25	103	62	9

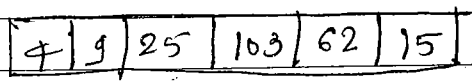
15 > 103 no swap



15 > 62 no swap

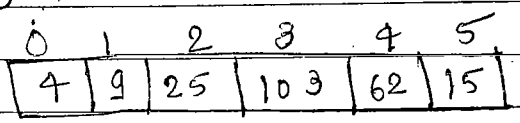


15 > 9 so swap

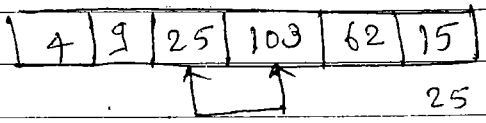
∴ ~~4 15~~ 

o/p :- 1st element is sorted.

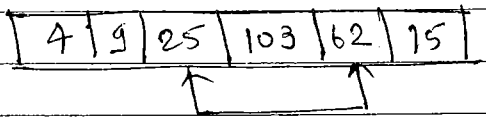
Iteration 3 :-



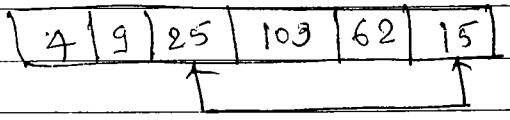
now 2nd element is compared with rest of the nos if greater then interchange



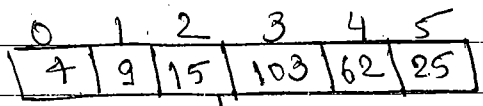
25 > 103 — no swap



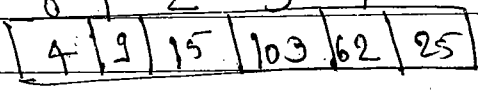
25 > 62 — no swap



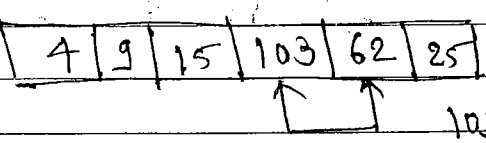
25 > 15 so swap



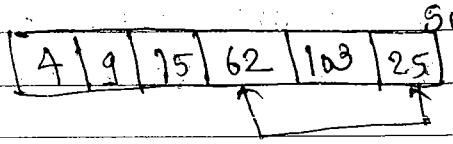
∴ o/p :- 2nd element is sorted.

Iteration 4 :- 

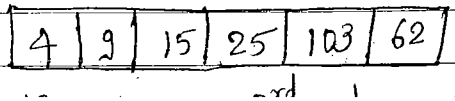
now 3rd element is compared with rest of the nos if greater then interchang



103 > 62 so swap



62 > 25 so swap



Iteration 5:-

0	1	2	3	4	5
4	9	15	25	103	62

now 4th element is compared with the remaining number

i.e.

4	9	15	25	103	62

103 > 62 so swap

4	9	15	25	62	103

The list is sorted.

3) Insertion Sort :- Insertion sort inserts each item into its proper place in the final list. The final list is already in sorted order.

- The insertion sort algorithm sorts a list of values by respectively inserting a particular value into a subset of the list that has already been sorted.

- sorts the first two values in the list relative to each other by exchanging them if necessary.

- Insert the list's third value into the appropriate position relative to the first two sorted values.

- Then insert the fourth value into its proper position relative to the first three values in the list.

- Continue this process for each position in the list.

e.g. suppose this is an unsorted array

0	1	2	3	4
77	33	44	11	88

Step 1 :- Select current element i.e. 77

Step 2 :- Compare current element with all elements in the left side.
// now here, there is no element in the left side of 77, since it is first element

Step 3 :- Increment current element i.e. 33
now compare it with all elements in the left side ∴ 33 < 77.
33 is smaller than 77 so swap it

0	1	2	3	4
33	77	44	11	88

now there is no element on the left side of 2

Step 4 :- Again increment the current element it is 4
now compare it with all elements in the left side 44 < 77 so swap

0	1	2	3	4
33	44	77	11	88

Again compare 44 with 33
44 < 33 no swap

Step 5 :- Again increment the current element, now it is 11., so compare it with all elements in the left side. 11 < 77 so swap
i.e.

33	44	11	77	88

Again 11 < 44 so swap

i.e.

33	11	44	77	88

again 11 < 33 so swap

0	1	2	3	4
11	33	44	77	88

Step 6:- Increment current element i.e. 88. compare 88 with left side of all the elements i.e. 88 > 77 no swap

~~88 > 44~~

so in this way get the sorted list by using insertion sort method

sorted array is

0	1	2	3	4
11	33	44	77	88

* Advantages of Insertion sort:-
1) It is simple
2) It also exhibits a good performance when dealing with a small list
3) The insertion sort is in place sorting algorithm so the space requirement is minimal.
4) It requires less memory space

* Disadvantages of insertion sort:-

The insertion sort does not deal well with a huge list.

* Complexity of Insertion sort:-
when inserting A[n] into the sorted A[0....n-1], only need to compare A[n] with A[n-1] and there is no data movement.

Thus, $T(n) = 1+1+1+\dots+1$ (n times)

$T(n) = O(n)$

Best case : $O(n)$
worst case : $O(n^2)$
Avg case : $O(n^2)$

Algorithm for insertion sort :—

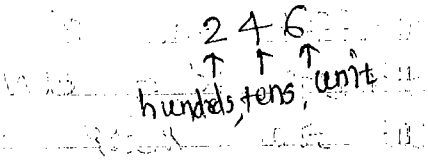
Step 1 : Accept n number into array data.
Step 2 : Data [0] is considered a sorted file of one element.
Step 3 : Next = 1
Step 4 : New element = data [next]
Step 5 : move all new element by one position to the right.
Step 6 : Insert new element in the array at the position where step 5 is terminated.

Step 7 : Next — next + 1
Step 8 : Continue from step 4 as long as next $\leq n-1$

Step 9 : Stop.

Program for insertion sort :-

```
Void main()
{
  int i,j;
   int a[10], i, j, temp, n;
     clrscr();
  Printf("\n Enter the array size : ");
  scanf("%d", &n);

  Printf("\n Enter array numbers you want to
        sort : \n");
  for (i=0; i<=n-1; i++)
  {
  scanf("%d", &a[i]);
  }

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

while ((temp < a[j]) && (j>=0))
  {
    a[j+1] = a[j];
     j--;

  }
    a[j+1] = temp;
}
Printf("\n Array after sorting :");

for (i=0; i<=n-1; i++)
{
printf("\n %d", a[i]);
}
  getch();
}
```



Radix sort :-

Radix sort is based on the position of digits. The number is represented with different positions. The number has units, tens, hundreds positions onwards. Based on its position the sorting is done.

Procedure for Radix sort :-

1) In the first iteration the elements are picked up and kept in various pockets by checking their unit digit.

2) In second iteration, the tens digits are sorted.

3) ~~Repeat through step~~ In third iteration, the digit at hundreds position are sorted.

4) Repeat the procedure until we will get a sorted array

Advantages of Radix sort :-

1) Radix sort is a very simple algorithm.

2) Radix sort is one of the fastest sorting algorithm for numbers or strings of letters.

Disadvantages of Radix sort :-

1) Radix sort takes more space than other sorting algorithms.

2) Radix sort is dependant on the digits or letters

3) For every different data type, the algorithm has to be rewritten.

4) Radix sort takes more time to write.

Complexity of Radix sort:- Assume there are 'n' numbers, we need to sort and k is the no. of digits in the largest number.
In this case radix sort algorithm is called k times.
The entire radix sort algorithm for n numbers takes kn times to execute.
Therefore, complexity of radix sort is $O(n \cdot k)$.

Algorithm for Radix sort:-

1) Start
2) Read an array of size n.
3) Partition numbers in ten groups based on least significant digit. Ten queues are required for this. Thus all numbers with least significant digit as '0' will be kept in queue[0], with least significant as "1" will be kept in queue[1] and so on till queue[9].
4) for k = least significant digit to most significant digit do
   a) for i = 1 to n-1 do
   b) y = x[i]
   c) j = k^{th} digit of y
   d) place y at rear of queue[j]
   e) for q = 0 to 9 do
      Place elements of queue[q] in next sequential position of 'x'.
5) Display sorted array
6) Stop

Radix sort :-

0	1	2	3	4	5	6	7	8	9
246	174	349	451	538	319	466	563	123	230

First pass :-

I/P nos	0	1	2	3	4	5	6	7	8	9
246							246			
174					174					
349										349
451		451								
538									538	
319										319
466							466			
563					563					
123					123					
230	230									

Second pass :-

I/P nos	0	1	2	3	4	5	6	7	8	9
230				230						
451						451				
563							563			
123			123							
174								174		
246					246					
466							466			
538				538						
349					349					
319		319								

Third pass

I/P nos.	0	1	2	3	4	5	6	7	8	9
319				319						
123		123								
230			230							
538						538				
246			246							
349				349						
451					451					
563						563				
466					466					
174		174								

After third pass when the nos. are checked they are in ascending order.

123	174	230	246	319	349	451	466	538	563

Radix sort :-

- It is a sorting algorithm that is used to sort numbers.
- We sort the numbers from least significant digit to the most significant digit.
- for eg. if we are dealing with unsorted numbers then we know that there are 10 digits from 0 to 9 that are used to form any number.
- So, we will need 10 buckets labeled 0 to 9 to sort the unsorted numbers.
- When we are sorting the numbers we will first find the number of digits in the biggest number.
- If there are N digits in the biggest no. then we will need to perform N number of pass.

- We will pad the remaining numbers with leading zeros so they all have N digits.
- Then we will take 10 buckets labeled 0 to 9 and sort the numbers.
- After the sorting is complete we will remove the leading zeros.

Sort the given numbers in ascending order using radix sort:-

348, 14, 614, 5381, 47

⇒ Here in above example, the biggest no. is 5381 having 4 digits. It means we have to perform 4 iteration or passes.

Then for the remaining nos we have to pad leading zeroes, so they all have 4 digit nos.

So now in above example the new list is after leading zeroes:-

0348, 0014, 0614, 5381, 0047

Iteration 1:- [checking units digit]

I/P nos.	0	1	2	3	4	5	6	7	8	9
0348									0348	
0014					0014					
0614					0614					
5381		5381								
0047								0047		

Iteration 2:- Checking tens digit

I/P nos	0	1	2	3	4	5	6	7	8	9
5381									5381	
0014		0014								
0614		0614								
0047					0047					
0348					0348					

Iteration 3 : checking hundreds digits

I/P nos	0	1	2	3	4	5	6	7	8	9
0014	0014									
0614							0614			
0047	0047									
0348				0348						
5381				5381						

Iteration 4 :- checking thousands digit

I/P nos	0	1	2	3	4	5	6	7	8	9
0014	0014									
0047	0047									
0348	0348					5381				
5381										
0614	0614									

Output :- 0014 0047 0348 0614 5381

After removing the leading zeroes sorted array is :-

14 47 348 614 5381

Quick sort:- Quick sort is a very popular internal sorting method.

- It is faster and easier to sort two small array than one large one.
- Quick sort follows divide and conquer strategy.
- In this approach numbers are divided and again subdivided.
- This division of list goes on until it is not possible to divide further.
  The procedure it follows is of recursion.
- It is also known as partition exchange sort.

Procedure of Quick sort is as follows:-

1) Select first element of list as a pivot element
2) Two index variable a and b are taken to divide the array.
3) a index refers to the first element and b index refers to the $(n-1)^{th}$ i.e. last element.
4) The job of index variable a is to search an element greater than pivot element.
5) The job of index variable b is to search an element less than pivot element.
6) When these elements are found they are interchanged.
7) Index variable a and b are incremented and decremented respectively. Exchanges are made appropriately if desired.
8) The process ends whenever the index pointers meet or crossed.
9) place the pivot element in such way that all the element before pivot are lessers and after pivot are greater.

10) original array is divided into two sub arrays.
11) Repeat through step 4 to 9 recursively for the

with one element.

Advantages of Quick sort :-

1) It is faster than others algorithms such as bubble sort, selection sort and insertion sort.

2) Quick sort can be used to sort arrays of small size, medium size or large size.

Disadvantages of Quick sort :-

1) Quick sort is complex
2) Quick sort is massively recursive.

* Complexity of Quick sort :-
- In quick sort, these will be approximately (n-1) comparisons in the first iteration. After which the file is split into two sublists each of size (n/2).
- For each sublist, there are approximately n/2 comparisons.
- There are T terms because the file is subdivided T times.
- Therefore best case complexity is $O(n \log n)$
- worst case complexity is $O(n^2)$.

* Algorithm for Quick Sort :-
⇒ Quicksort (A, p, r)
Step 1 : if p < r then
q = Partition (A, p, r)

Step 2 : quicksort (A, p, q)
Step 3 : quick sort (A, q+1, r)
Step 4 : stop.

The key to the algorithm is the partition function which rearranges the subarray A[p,r] in place.

Partition (A, p, r):

Step 1: $x = A[p]$, $i = p-1$, $j = r+1$

Step 2: As long as $A[j] \leq x$, $j = j-1$

Step 3: Increment 'i' value as long as $A[i] \geq x$

step 4: if ($i < j$) then
swap A[i] and A[j]
else return j

Step 5: if ($i < j$) then step 2

Step 6: Stop

example of Quick sort:-

44	33	11	55	77	90	40	60	99	22	88	66

Array size = 12

Consider first element 44 from the list now scan the array or list from Right to Left and find out first small numbers than 44 Here it is 22 so swap it

22	33	11	55	77	90	40	60	99	44	88	66

now considering above list, scan it from left to Right and find out first biggest no than 44, here it is 55 so swap it

22	33	11	44	77	90	40	60	99	55	88	66

Again scan from Right to left and find first smallest no than 44, here it is 40 so swap it

22 33 11 40 77 90 44 60 99 55 88 66

Again scan from left to right and find first biggest no than 44 it is 77, so swap it

22 33 11 40 44 90 77 60 99 55 88 66
← left → Right

Now in above list from no. 44 you can see left part have smaller nos than 44 and right part have all greater no than 44.

It means we got 2 lists :-

22 33 11 40 and 90 77 60 99 55 88 66

List 1 List 2

i.e. 44 no. got its correct position.

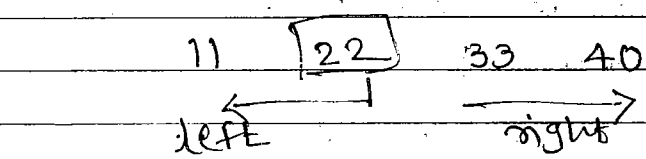
Now considers List 1: 22 33 11 40
considering first element 22, scan the list from right to left and find first smallest no. than 22, here it 11, so swap it -

11 33 22 40

again scan the list from left to right and find first biggest no than 22, it is 33, so swap it

11 22 33 40

Now from above list you can see from element 22 left side have small nos and right side have greater no.



it means 22 got its correct position

How consider List 2 :-

90 77 60 99 55 88 66

considering first element 90, scan the list from Right to left and find first small no than 90, here it is 66, so swap it

66 77 60 99 55 88 90

again scan from left to right and find first big no. than 90, here it is 99, so swap it

66 77 60 90 55 88 99

again scan from Right to left and find first small no than 90, here it is 88 so swap it

66 77 60 88 55 90 99

Now from above list you can see from element 90, left side have all small nos and right side have greater nos. i.e:

66 77 60 55 90 88 99

66 77 60 88 55 90 99

← left → Right

Now from above list you can from element 90 left side have small no and right side have greater nos [it means 90 got it correct position.

We got 2 lists again it is:—

66 77 60 88 55 | 99

list1 list2

Consider list 1 again it is

[66] 77 60 88 55

Consider first element 66, scan list from Right to left and find first small no than 66, here it is 55 so swap it

55 77 60 88 66

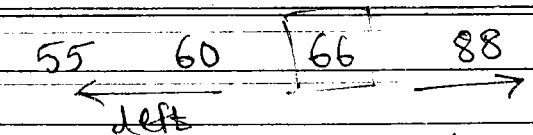
again scan from left to Right and find first bigger no than 66, here it is 77 so swap it

55 66 60 88 77

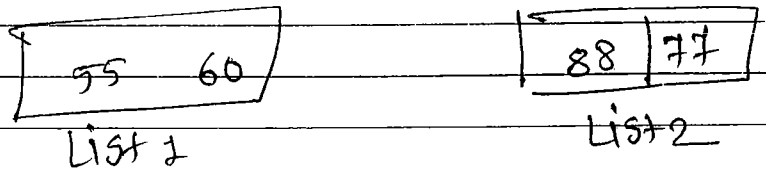
Again scan from Right to left find first small no than 66 it is 60 so swap it

55 60 66 88 77

Now from above list you can see from element 66 left have small no and Right have greater nos.



it means no. 66 has got its correct position
we get again 2 lists it is



now list 1 already sorted.

Consider list 2 it is 88 77
considering 88 scan the list from Right to left find first small no it is 77 so swap it

77 88

In this way we sort all the elements.

11	22	33	40	44	55	60	66	77	88	90	99

* Points to remember for Quick sort :—

1) This sorting algorithm uses the idea of divide and Conquer Strategy.

2) It finds the element called pivot which divides the array into two parts in such a way that elements in the left half are smaller than pivot and elements in the right half are greater than pivot.