Data Structure & Algorithims

Recommended Book:-
Data structure using "C" by Sajmoor Lipschutz.

Data: Collection of raw facts and figures.

Raw → pure form
facts → Reality
Figures → NO., symbols, letters, etc.

The word data come from the word Entity.
Entity → attributes | features → The collection of the data across the attributes and features

Data simple means kesi bi specific entity ki kesi bi specific attribute ka name hai.

Columns are attributes.
Rows are Entity.

Data are also known as datum | data items.

data items
- grouped
  - splitable
  - e.g:- Name, full Reg. NO.
  - Name → First name, Middle name, surname.
- Elementary
  - non-splitable.
  - e.g:- Roll no., GPA.

• group aik stage pr a ke elementary ban jae gi means us ko further divide nahi kr sakte.

Rows are also called record and tuple.
Columns are also called attributes and fields.

⇒ Records :-

- Fixed length
- Variable length

Fixed length means ke sab hi length same hogi means ke sab hi rows ke same attributes hoge.

Variable length means ke sab ki length same nahi hogi means ke agar kesi student ne koi course skip kia hai.

Primary Attribute :

Primary attributes mean ke kesi bi data me aik unique column hoga jo kesi person ko identify kre ga like rollno.

Entity set → means same functions.

Data Structure : own words in final

1. First we have a problem of the data. made Algorithim to solve a problem by selecting best data structure.
2. Implementation of the solution.
3. Analysis of the designed solution.

Define : Mathematical model of a specific organization of data

Data structure →

primitive

non-primitive

Data Structure

Data structure

Primitive

Default programming are called primitive data structure.

e.g:- float, int, etc.

simply means default data structure.

Non-primitive

Customizable data structure are called non-primitive data structure.

e.g:- length, etc.

Non-primitive

Linear

- one-to-one relation
- sequence data

e.g:- 10, 20, 30, ...

Non-linear

- one to many
- Like grandfather, to father to child and grand child each of the child has many sons and daughter.
- mesh type of data.
- many to many

Linear data structure :-

1- Arrays :- Store same and homogenous type of data., provide random-access.

2- Linked list :- store non-homogenous type of data. Not define the size of data.
Linked list are made to fullfil the drawback of arrays.
- Linked list not provide the random access of memory.

3- Stack :-
Stack is a single ended List / data structure.
single end means insertion or deletion from the same step/end. e.g:- undo, ctrl + c, etc.
means ke phele last operation perform hoga

4- Queue :- means line or Qataar to make discipline. Simply means JO phele aia hai wo kaam phele kr ke jae ga.
Queue means strictly insertion from one step and strictly deletion from the other step. insertion from backend and deletion from the front end. Back also known as rear.

Non-linear:

1- Tree :-
2- Graphs :-
Combination of vertex and edges. Loops wagera.
3- Hash tables.

Operation on DS :-

1- Insertion
2- Deletion
3- Traversing
4- Searching
5- Sorting
6- Merging

Traversing means searching the entire data at least one time. ↓ access
e.g:- to find the maximum number to search or access the all data.

Data structure

Abstract Data Type: (ADT)

String.h → Library.

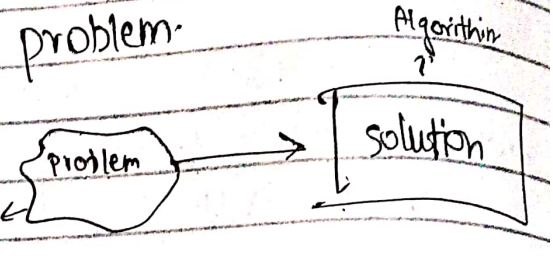
already defined Library function.

Abstract data type means ke dosri library se jo already defined function hai us ko use krna.

Algorithims:

Steps to solve the problem.

- First we analysis the problem.
  - → what to do?
  - → How to do?
  - → Data?



- Based on all analysis we find the solution and this solution is called algorithim.

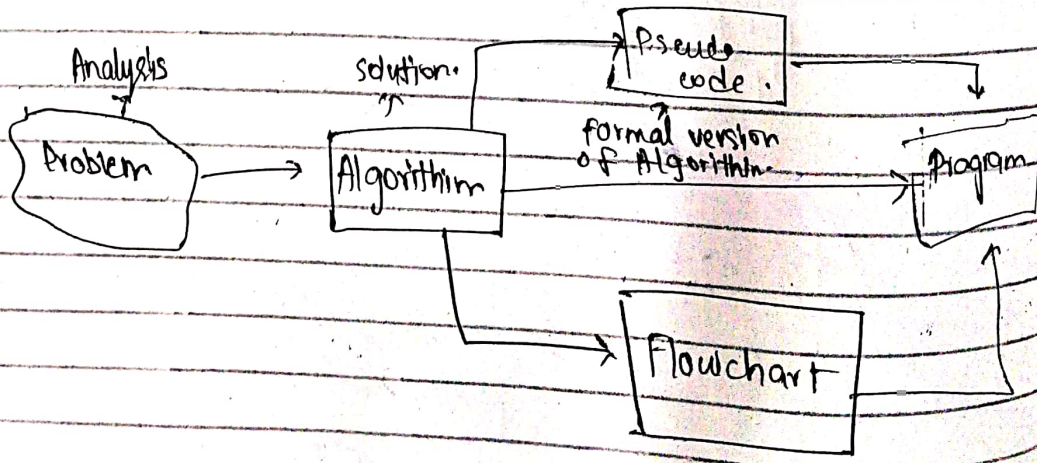
→ Algorithim is the step by step problem solving solution.

→ Non- Executable

→ Natual language in simple English.

→ Language in dependent.

→



→ Need of pseudocode is for complex programme.

Flowchart:- Flowchart is the graphical representation of code.

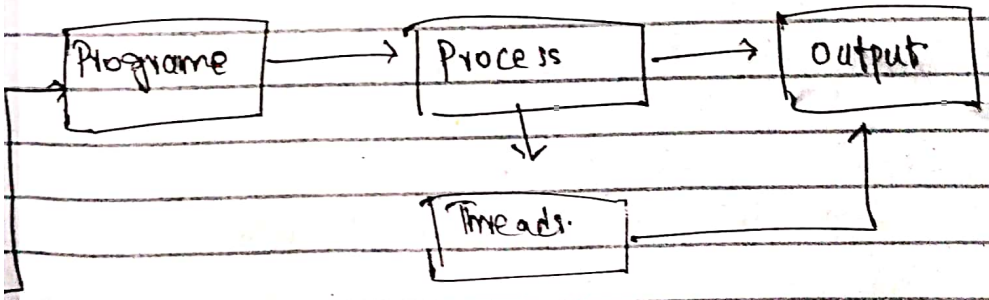
Pseudocode:- Formal version of Algorithim.

Programme:- • It is simly implementation using programming language.
• language dependent
• Executable file
• set of instruction.

Process:- • After programme, the programme is in processing.
• Programme is in executable.

Output:- The final Result.

Threads:- Light weight processes. Example:- Multiple tabs of Browser.



After All this find the Efficiency of the Algorithim.

↓

IMP.

MCQ.

Muhammad Bin Musa Alkawarizmi Find the Algorithim. (Alkawarizim.

How to write an good Algorithim:-

- Identification Number:- First we give the identifie number to the Algorithims, simply mean identity refinement is also possible.

- Steps, Control, Exit
  → give number to step and write step by step.

  Control Structure:-
  - Sequential (line by line)
  - Conditional / selection (if-else, switch)
  - Repetitive / Iterative (loops

  Exit:-
  Output. /One or more Exit point.

→ Every step must be completed.

- Variable names:-
  Every variable must have decent name according to the problem.

- Comments:- For good understanding.
  comment write in square bracket like,
  ← [ ] :- ——

- Assignment:- (x=a+b, x:=a+b, x ← a+b)

- Input / output:- (Read and write for algorithim).

- Procedures (Reuseable functions write use procedure)

DSA (Lab)

Problem solving -?

↓

Process

↙ identify Process

↘ Find Best

There are three technique for problem solving.

↙ Programme → set of instruction

↓ Algorithim → Step by step Procedure to solve a problem.

↘ Flowchart → control flow, Graphical representation of Algorithim.

DFD:- Data Flow diagram. (Flow of data).

Parts of Programme:- 3 parts.

1- Preprocessor Directives
2- Main Function
3- Body

PPD:- Instruction given to the compiler before the programme.

Two types of PPD.

1- Include → to include h.f in programme.
2- Defined → # define M 10

↓ To define the constant value.

Header File:- definiton of already built function in H.F.

Start Programme to main. , with main programme only compile not execute.

```
int main () {

    return 1;
}
```

Basic Programme.

<< → insertion operator.

>> → extraction operator.

String:- Anything write in double comma are called string. " ".

Character constant:- Anything write is single quote are called character constant. 'o', '1'

→ size 1. → maximum.

Constant:- Constant is the quantity that cannot be change during execution of programme.

Types of constant:-

1- Numeric constant

2- String constant

3- Character constant.

N.C:- consists of numbers.

Numeric constant.

- Integer constant → without points value.
- Floating point constant → point ⇒ value

Variable:- → Name → Name of memory location/cell.

Variable deceleration:- → process → spicify two thing.

Decelare name and data type.

Variable intilization:- The process of deceleration and initialization at the same step is called variable initialization.

Expression:- operators are symbols that are used to perform operation on data.

operands:-
A-Z
0-9.

Arithematic operators.
↓
Basic operations
+, -, x, ÷

Relational operators.
↓
compare two thing and show relation.

Combination of operator and operands are called expression.

Escape sequence :- Backslash.
↓
\n, \a, \r, ---

Escape sequence are special symbols or characters that are used to modify the output.
They are written in double quotation.

\n → New line.
\a → Beep sound.
\r → carriage return.
\b → Backspace. → remove one character from left.
\f → Form feed. → for used of printing.
\' → single quotation.
\" → double quotation.
\\ → forward slash.

```
# include <iostream>
using namespace std;
int main () {
    int num1 , num2;
    num1 = 5;
    num2 = 5;
int num3 = num1 + num2;
cout << num3;
    return 0;
}
```

1 2 3 4

1 2 3 4

10

```
#include <iostream>
using namespace std;
int main() {
    cout " Enter the number " << endl;
    int a;
    cin >> a;
int d1 = a / 1000;
a = a % 1000;
int d2 = a / 100;
```

```
#include <iostream>
using namespace std;
int main() {
    cout<<"Enter the number to reverse"<<endl;
    int n;
    cin>>n;
int d1= n/1000;
    n= n%1000;
int d2= n/100;
    n= n%100;
int d3= n/10;
n= n%10;
int d4= n;

cout<< d4 <<d3 <<d2<<d1 ;

return 0;
}
```

$$x - 3y = -7$$
$$2x - 6y = 7$$

$$x + 2y + 3z = 6$$
$$2x - 3y + 2z = 14$$
$$3x + y - z = -2$$

$$x + 2y - 3z = -4$$
$$2x + y - 3z = 4$$

DSA

Functions :-

Function is simply ~~give~~ take input from user and after processing show to the user.

Ceiling

$\lceil x \rceil = y$

$x \Rightarrow \mathbb{R}$, $y \Rightarrow \mathbb{Z}$

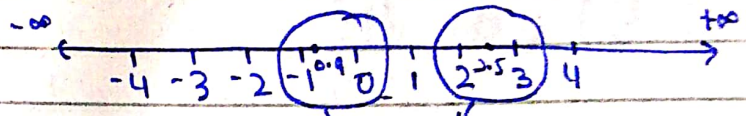
$\lceil 2.5 \rceil = 3$

Floor

$\lfloor x \rfloor = y$

$x \Rightarrow \mathbb{R}$, $y \Rightarrow \mathbb{Z}$

$\lfloor 2.5 \rfloor = 2$



ceil is give the greater integers and Floor give the Least / small integer.

$\lceil -0.9 \rceil = 0$

$\lfloor -0.9 \rfloor = -1$

Integer Function:-

It function give return the integer value.

int (2.5) = 2

int (-3.9) = 3

Absolute Function:-

It can change the negative number to the positive.

$|2.5| = 2.5$

$|-3| = 3$

Remainder Function:-

It give the remainder of the expression after perform operation.

9 % 2 = 1

3 % 1 = 0

5 % 1 = 1

3 % 1.5 = 3

5 % 3 = 2

Note: $5 \overline{|3}$ → Ans.

In this case.

Summation Number:-

$$\sum_{i=1}^{n+1} i = 1+2+3+\cdots+(n+1)$$

$$\prod_{k=1}^{n} k = 1 \cdot 2 \cdot 3 \cdot \cdots \cdot n$$

Factorial Function:-

The factorial function is the recursive function.

$$n! = n(n-1)!$$

$$5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$$

Permutation

$${}^{n}P_r = \frac{n!}{(n-r)!}$$

Combination

$${}^{n}C_r = \frac{n!}{r!(n-r)!}$$

Applications of Permutation and Combination.

9MP

Exponent

n^x

$1^2 = 1$

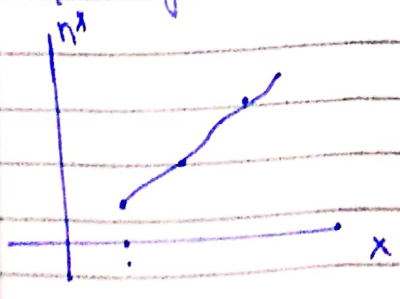
$2^2 = 4$

$3^2 = 9$

Plot on graph.

input → x-axis

output → y-axis

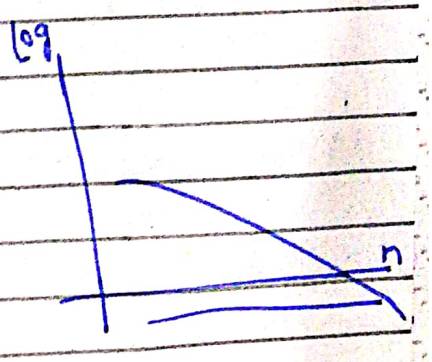


Logarithmic

log n=?

log 1000

log 2001



Complexity:

- Time complexity is most important.

It's totally depend on the machine CPU cycle.

- space complexity
- computational complexity.

Linear search:

10	20	30	40	50

one by one search all the array to find the number.

```
int arr[5]={10,20,30,40,50};
int key;
int pos=-1;
cin>>key;
for(int i=0; i<5; i++)
{ if(arr[i]==key)
  { pos=i;
    cout<<"value found"<<pos;
      Break;}
if(pos==-1){
  cout<<"Invalid";
  }
}
```

Now see the execution time of the programme.

Different system has different system speed and clockspeed.

Every system has run the same programme and answer in different time.

Time complexity is not the find the exact time. It find the no. of cycles of CPU.

One CPU cycle give to the one step of the programme.

Linear search give output of these cases:.

① Best search:- | Best case:-

$T_{ls}(n) = 1$

Find the value in the single comparison.

② Worst or Case:-

$T_{ls}(n) = n$

Maximum time run the bad output mille gi.

③ Average Case:-

$$T_{2n}(n) = 1 \times \frac{1}{n} + 2 \times \frac{1}{n} + 3 \times \frac{1}{n} + \cdots + n \times \frac{1}{n}$$

$$= \frac{1}{n}\left(\frac{n(n+1)}{2}\right) = \frac{n+1}{2} = \frac{n}{2} = \frac{1}{2}$$

★ good.

$$= i_1 P(i_1) + i_2 P(i_2) + \cdots + i_n P(i_n)$$

$$= 1 \cdot \frac{1}{5} + 2 \cdot \frac{1}{5} + \cdots + 5 \times \frac{1}{5}$$

$$= \frac{1}{5}(1+2+\cdots+5)$$

$$= \frac{1}{5}\left(\frac{5(6)}{2}\right) = \frac{6}{2} = 3$$

$\frac{1}{5}(1+2+3+4+5)$

$\frac{1}{5}$ 15 $\frac{15}{5} = 3$

$\frac{5+1}{2} = 3$

Time complexity of Algorithim

halt means to stop. with
For correct output and halt are the best correct algorithim.

Asymptotic:- It simply means boundary.
e.g.:-
$10 \leq x$ → we don't know the exact value of x but we know the range.

If the algorithim input is not lie in the (polynomial, quadratic, linear, logarithm, exponential) → classes. So means the complexity of algorithim is very high.

If the running is in the 5 classes, which means our Algorithim is efficient.

→ After find the efficiency compare the algorithm with the market existing best algorithm to solve the particular same problem.

→ The function are always boundary if is inside the ( ). → Target function for comparison.

$2n^2 \in O(n^3)$
↑ $f(n)$ ↓ $g(n)$

$0 \leq f(n) \leq c\, g(n)$
↑ Not right always. ↑ non-zero any positive constant it cannot effect

Note:- We can't pick the value bcz it retest and it create violation.

$0 \le f(n) \le c\,g(n)$

$f(n) \le c\,g(n)$

Input ∈ Natural Numbers.

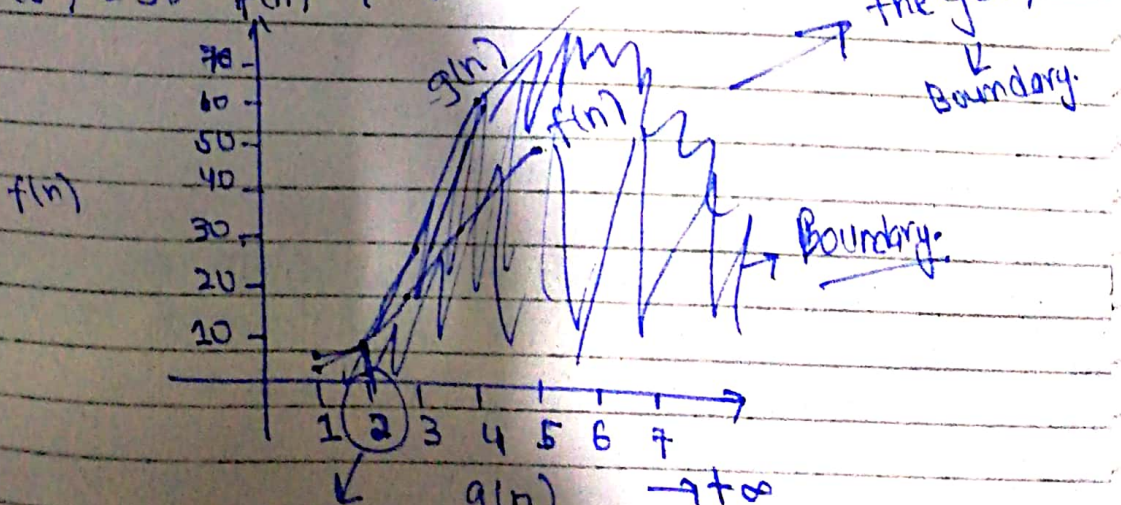
Note:- $an^2 \le cn^3$

$a \le cn$ → Both are good.

Note:-

$c=1$	$c=2$
$n=2$	$n=1$

$f(n) = an^2$	$g(n) = n^3$
$f(1) = 2$	$g(1) = 1$
$f(2) = 8$	$g(2) = 8$
$f(3) = 18$	$g(3) = 27$
$f(4) = 32$	$g(4) = 64$
$f(5) = 50$	$g(5) = 125$



function

f(n) is inside the g(n)

Boundary.

Boundary.

2 is the point it means ke vidation nhi hai means [n=2.]

We must say.

$f(n) = O(g(n))$ - or - $f(n) \in O(g(n))$

Data Structure

Assuming 1 unit cycle take the one steps of execution.

→ Big O Notation analysis is the most importan

Big Omega Notation:-

$5n^2 \in \Omega(n)$ → $g(n)$

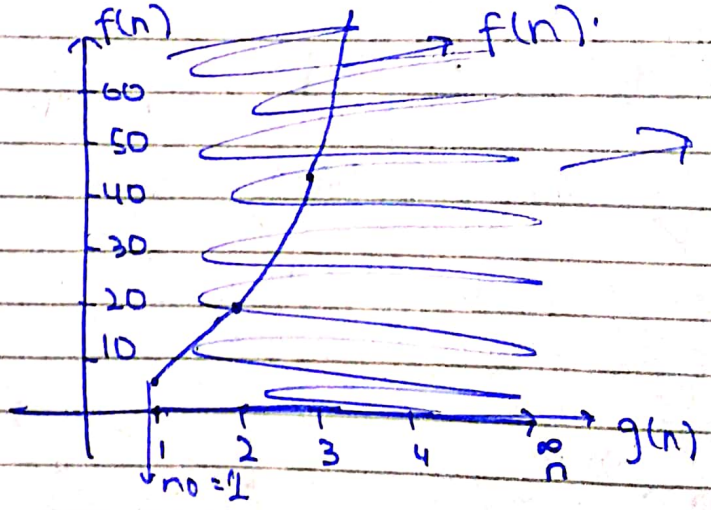
↓

$f(n)$

$0 \leq cg(n) \leq f(n)$

$c n \leq 5n^2$

$c \leq 5n$



$f(n) = 5n^2$	$g(n) = n$
$f(1) = 5$	$g(1) = 1$
$f(2) = 20$	$g(2) = 2$
$f(3) = 45$	$g(3) = 3$
$f(4) = 80$	$g(4) = 4$

→ since, $f(n)$ is in the boundary $g(n)$. we found $n_0 = 1$ and we satisfy the equation with taking $c = 5$.

$5 \leq 5(1)$, $5 \leq 5$

Linked List | Array

Most IMP.

Data Structure (Lab)

Search Method in array :-

Data Structure

Array :- It is a collection of numbers.

- Store Homogenous type of data | same data type.
- Linear data structure.
- Fixed length | defined | Poor to execute.
- consective memory allocate.
- If consective memory are not available so array can't be defined.
- Random or direct access of data.
- Assign indexes to memory.
- pointers are created of each byte and default first byte assign.

$$Loc[K] = Base\ add + W(K - LB)$$

LB → least Boundary

$$Loc[3] = 496 + 4(3 - 0)$$

$$Loc[3] = 508$$

$$Loc[421] = 496 + 4(421 - 419)$$

$$= 496 + 4(2)$$

$$= 504$$

Traversing :-

read all the array atleast one time.

DSA

Bubble Sort:

It is used to sort the elements of array. It is used only for numeric values.

→ To sort the data both in ascending and descending order.

→ Compare two values in the form of pairs.

→ lesser index value is smaller than higher index is ok so move to next pair, if no, then swap the value. (just for ascending).

→ After first complete iteration the highest value move to the last.

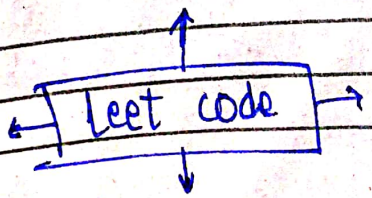
→ After second complete iteration the value is lower than higher is sorted.

→ Bubble sort har sort me iteration chala kr iteration kam kr ke chalta jae ga (Drawback).

→ So, for this kind of sortness the data can sorted only on one iteration bubble sort is not recommended.

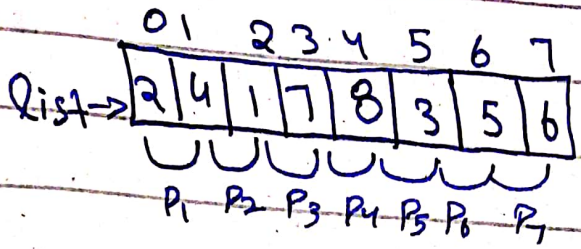
```
Bubble-Sort (List, n) {
    int temp;
    for (int i = 0; i<n; i++) {
        for (int j = 0; j<n-1; j++) {
            if (list[j] > List[j+1]) {
                temp = list[j];
                list[j] = list[j+1];
                list[j+1] = temp;
            }
        }
    }
```

array name → List, n
elements in array.
i<n → This is true condition.
j<n-1-i
This is the true condition for minimize the pairing after one complete iteration.



DSA.

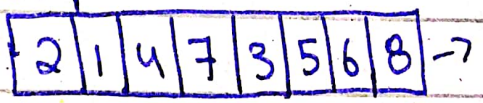
Bubble sort:



Ascending order
smaller – greater.
0, 1, 2, 3 ---- →

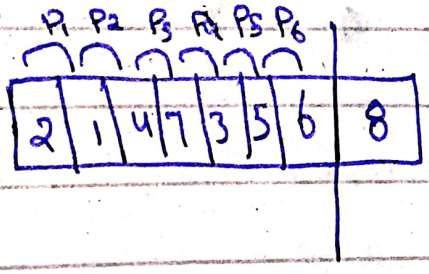
```
if (list[1] > list[2])
  { swap values }
```

After 1st step.

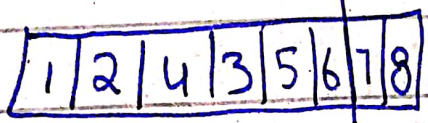


greatest value end index/last index main a gai.

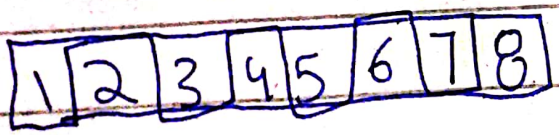
Then again.



After again by pair making ~~and swapping~~



then again ~~by~~ pair making.



✓

leetcode.

Function of bubble sort in C++.

```
(list, n) {

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

```
insert (list, n, k, item)
{
  for( int i=n+1; i<k; i--)
{
      list[i+1] = list[i  ];
                          {
      list
```

"DSA"

Linked List :- (Linear data structure)

=> Because they store the data in a linear form.

=> non-consective memory allocate.

=> Non-homogeneous data store kr skty hain.

=> run-time / dynamic memory management

=> Sequential access.

Linked List.

How to make Linked list (~~kese bnate~~)??

collection of nodes.

minimum 2 parts. block of memory.

i- information / data.

ii- link / next pointer (Address of the next node).

=> alway start with (start / head pointer).

=> In the first start = null - (Linked list is empty).

=> for last node

↳ link part = null

(Matlab end pointer null).

Note: (structure and pointers pr achay sy grip krni pry gi linked list ky liay).

*- pointer take same space for float, charc, int etc....

```
Struct node
{
    int data;
    node *link;
};
node *start=null;

node n1=new node();
n1=10;
n1=null;
start=n1;
node n2=new node();
n2=20;
n2=start
start=n2;

node n3=new node();
n3=30;
n3=start;
start=n3;
```

data ←
link ←

data ⇐
address ←

Loop for printing nodes:- - - - - - -

```
node * temp = start;
while ( temp != null
        cout << temp->d;
        temp = temp->l;
```

Data Structure

Note:- Node,

Linked List:-

Every node insert at first in this scenerio.

```
struct node {
        int data;
        node * link;
    };
node *start = null; (list is empty)
insert_first (* start, x)
{
    if (start == null) {
        node n= new node();
        n=d → x ;
        n = l → null;
        start =n;
    }
else {
        node n = new node();
        n → d = x;
        n → e = start;
        start =n;
}
```

Node minimum has two parts.

•Every element points the next element (Address).

→ Node insert at the last. → The given.

```
insert_last(*start, n) {
    if( start ==null){
        node n= new node();
        n→d = x
        n→l = null;
        start =n;
    }
    else {
/*Traversing in the node to check the last
one and the condition is it point of link
is null. */
        node. *last = start;
        While ( last → l = null)
        { last = last→l; }
        node =n;
        n→d = x;
        n→l = null;
    last→l = n;
    }
```

The traversing method is very time taking to traversing step and search 1000 and more nodes. So, in this case we use another method in which we also initialize the last at the start. to minimize the time (*last =null; (*last), last=n)

Addition in the already given programe.

Deletion in linked List from the begining or from the last.

```
delete_first(*start)
{ if (start == null) // list is
                     // Empty
cout << "Under flow";
else     node *temp = start;
  { start = start -> l;
    delete temp;
  }
```

These two steps are used when garbage collector is not working.

```
delete-last (*start)
{ if(start == null){
    cout << "Underflow";
  }
else {
  node * temp = start;
 while (temp -> l -> l != null)
    { temp = temp -> l; }

    temp -> l = null;
  }
```

Over-flow:- It is the method when you insert new value more than the size.

Under-flow:- It is the method when you delete the already empty array, Linked List.

Gorbage Collector:- After periodic time the garbage see that the user can use all the node and which 1 is not use is transfer to the avail pool.

Avail pool:- It is the free space in the OS.

This is the simple linear and simple search.

Searching in the Linked List.

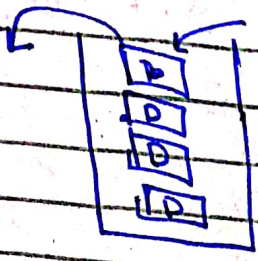
```
search (*start , key)
{  node * temp = start;
   while ( temp ! = null)
   { if ( key == temp -> d)
        cout << "value found ";
        break;
     else
        temp = temp -> l;
   }
   cout << " value not found";
}
```

pop, peek, push
is empty, is Full

DS (Lab)

Stacks:- It is an abstract data type. (ADT).
- only access data from one side.
- It is LIFO (Last in first out).

e.g:- Pile of plates. ≡



→ Last wale data ko sab se phele access kr ke next pr jae ge.

- operations on stacks (Push() or Pop())

Push() → Place the data item on the stack.
Pop() → Remove the data item from the stack.
→ Three methods are used to mange Push() or Pull().
- Peek() → It purpose only pick data and display not remove. on the other pop() is same but they also remove the data. int top = -1;
- is Empty() , • is Full(). stop = Store the top value of array (latest).

```
int Peek() {
    return stack[top];
}
bool
isEmpty() {
  if(top == -1)
    return true;
  else
    return False;
}
Bool
is Full( ) {
  if (top == maximum) {
    return true;
  else
    return false;
}
```

Stack underflow:- not exceed the minimum limit is called underflow Stack empty has or app data access kr rahe ho to uso under-flow (stack).

Stack overflow:- reach the maximum limit and exceed the limit is called overflow. stack overflow wo ha jab limit full ha or app new data add krna chate hai.

```
Void Push (int data){
  if(!isFull()){
    top = top + 1;
    stack[top] = data;
  }
  else {
    cout<<"Stack Already Full";
  }
}
```

isFull = true

Data structure

→ Binary Search is directly not possible in the linked List. Another method is attach index with the node to apply Binary Search.

```
void search SortedLinkedList (*start, key)
{  node *temp = start;
   while (temp != null) {
      if (temp→d == key) {
         cout<< "value Found";
         break; }
      else if (temp→d > key) {
         cout<< " value not found";
         break; }
      else { temp = temp→n; }   → next
   cout<< "value not found";
   }
}
```

Stack:-

```
int stack[size];
int Top = -1;
void push (int x)
{ if(Top == size-1) {
   cout << " stack overflow"; }
  else {
      Top = Top + 1;
   stack [TOP] = x;
   }
```

```
void pop ( )
{ if (Top == -1){
  cout << "Underflow";
  }else {
  cout << stack [Top];
  Top = Top - 1;
}
```

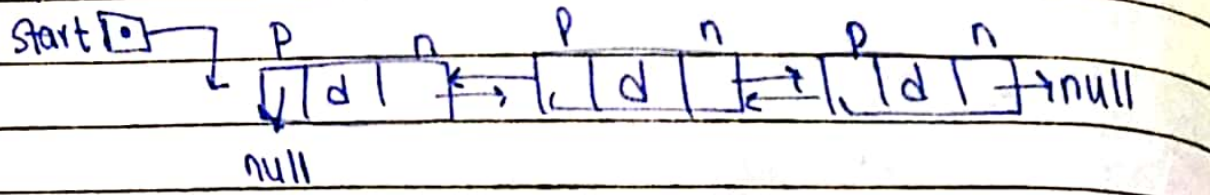
Linked List:-

1. Description
2. Insertion (at begin, at end, at pos)
3. Deletion (at begin, " ")
4. Searching (sorted, unsorted)
5. Traversing
6. Sorting → Bubble sort (Home Task).

DSA

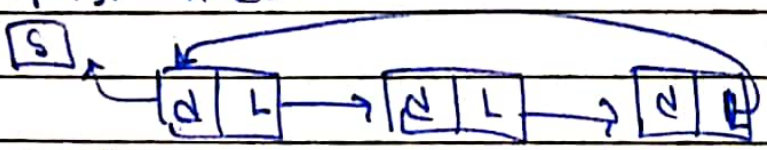
Types of Linked List:-
1- singly | one-way
2- Doubly | Two-way.

→ Singly can store only the address of next data.
→ Doubly can store address of both backward and forward data.



- data part
- next | forward pointer
- previous | Backward pointer

1- Grounded Linked List are the Linked List in which there end part is null.
2. Circular Linked List are the Linked List in which there last part store the address of first node.



3- Header Linked List In which we attach the extra node at the start position this is header node (it can't carry data, it carry meta data).

Meta data:-
It carry information of data. | information about the data of Linked List.

→ The use of header Linked List is we can't program the extra code for start node(null).

M T W T F S

DATE:

Doubly Linked List:-

```
#include <iostream>
using namespace std;
Strut dnode
{ int d;
  d node* n;
  dnode* p;
};
dnode *start ==null;
void insert-begin (dnode* start, int x) {
  if (start == null) {
    dnode *temp = new dnode();
    temp→d=x;
    temp→n=null;
    temp→p=null;
    start=temp;
  }
  else {
    dnode *temp = new dnode();
    temp→d=x;
    temp→p=null;
    temp→n=start;
    start→p=temp;
    start =temp;
  }
```

```
void insert_last (     ) {
        A →previous
else {
    dnode *ptr = start;
  while (ptr →n ! =null) {
      ptr=ptr→n; }
  dnode *temp =new dnode();
   temp →d =x;
   temp→n= null;
  temp→p = ptr;
   ptr→n=temp;
}

delete_begin (*start) {
    if(start ==null) {
    cout<<" Under Flow";
     }
    else {
        start→n→p =null;
        start=start→n;
    }
 // Dry run the code and check all values deleted or no

del_last (*start){
   if(start = =null) {
    Cout << "Over Flow";
    }
  else {
     dnode *ptr  = start;
  While (ptr→n→n! =null) {
     ptr = ptr→n;}
    ptr→n =null;
}
```