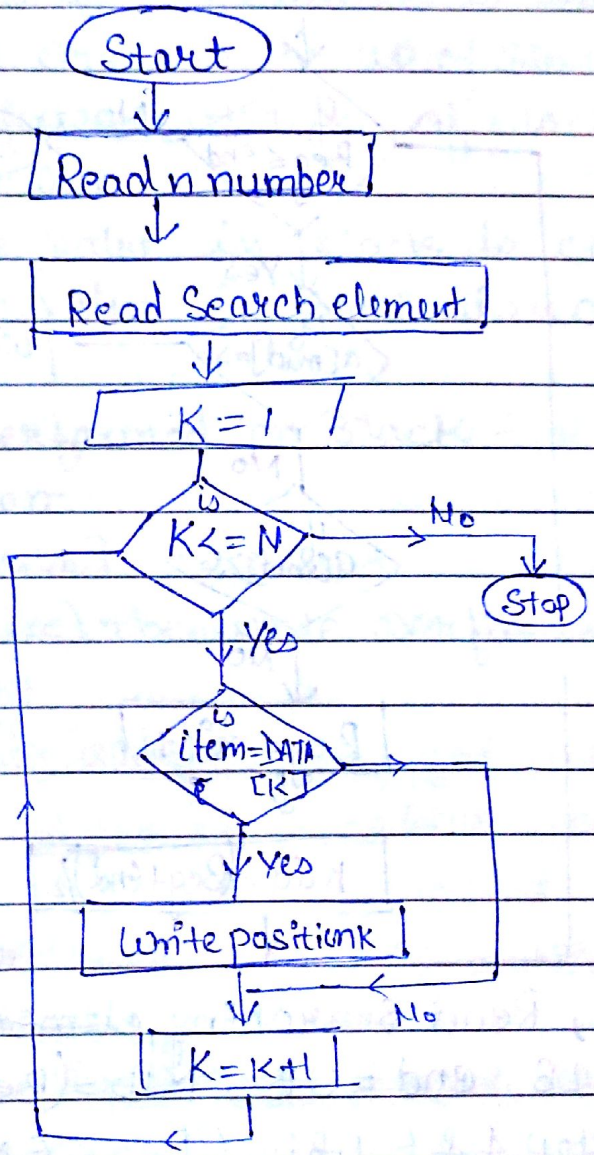
ADA

Searching :- Searching can be done in two way

1) Linear Search
2) Binary Search

1) Linear Search :-

Flow Chart :-



Algorithm :- Read n number, Read Search element

Step II :- [Initialize] Set K := 1

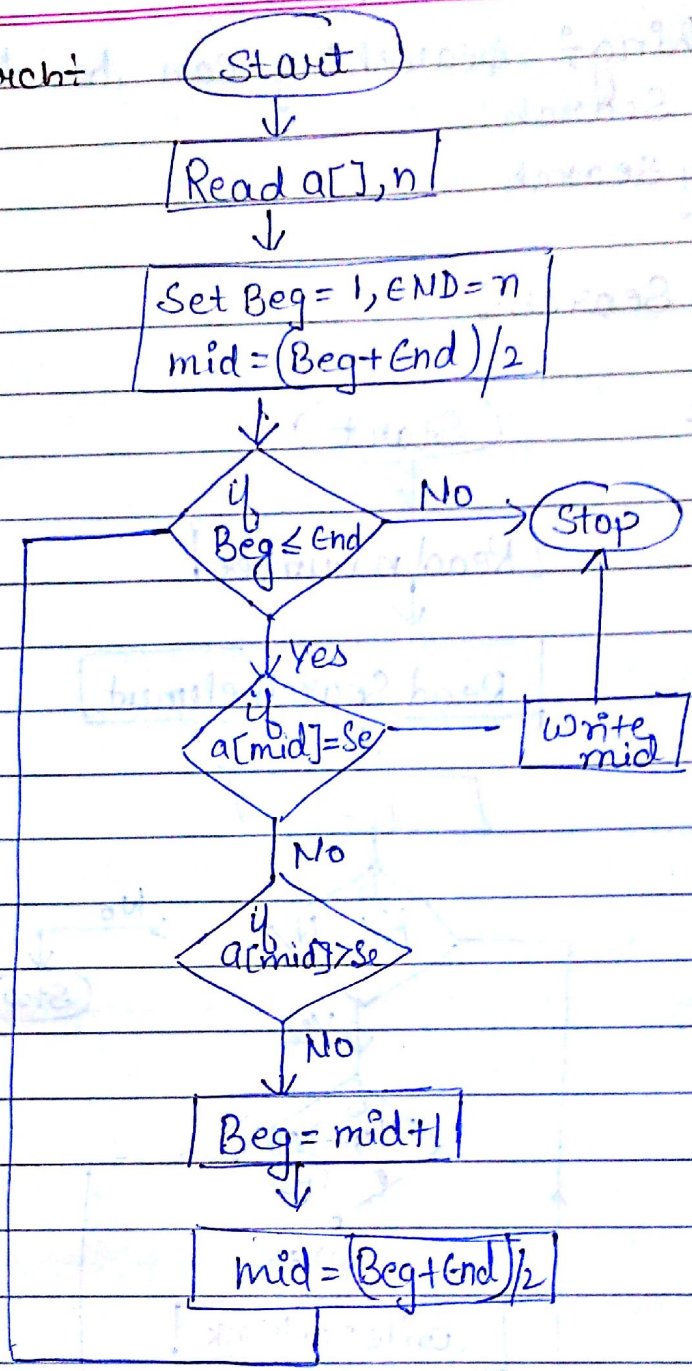
Step III :- Repeat Step 4 & 5 while K <= N

Step IV :- [Comparison] If (DATA [K] == item) then write position K

Step V :- [Increment Counter] K = K+1
[End of loop]

Step VI Exit

2) Binary search:



Algorithm: Read n no, Read searching element

Step II: Set Beg = LB, end = UB, MID = (Beg + END)/2

Step III: Repeat step 4 & 5 while (Beg ≤ END & a[mid] ≠ Se)

Step IV: [Test Counter]

If (a[mid] > Se) then

END := MID - 1

Else

Beg = MID + 1

Step V: MID = $\frac{Beg + END}{2}$ [End of step 3 loop]

Step VI:- If (a[mid] == Se) then write Element position & exit the not found & exit

Step VII:- Exit

Stack:- Stack is a non primitive linear data structure in which insertion and deletion is done only at one end of the stack or the top of stack It is called Sequential representation of Stack. It is LIFO Last in First out

Inserting the value in stack is called Push

Deleting the value in stack is called pop

Operation performed on stack :-

1) Push operation:-

Step1:- Set Stack[size]

Step2:- If (Top == Size) then write overflow & exit

Step3:- Top = Top+1

Step4:- Stack [Top] = item

Step5:- Exit

2) Pop operation:-

step1 Set Stack [size]

step2 If (Top == 0) then write Underflow & exit

Step3 item = Stack [Top]

Step4 top = top -1

Step5 Go to Step 2

Mathematic Expression:-

1) Infix:- In which the operator Symbol is placed bet its two operands

eg A + B
C * D

2) Prefix Notation :- eg +AB, *CD. Reffered to the notation in which the operator symbol is placed before its two operands. It is also called the polish notation.

3) Postfix Notation | Reverse polish Notation :- It is reffered to the notation in which the operator symbol is placed after it two operands eg AB+ , CD-

Evaluation of postfix expression

ALGORITHM:- Add a right paranthesis ")" at the end of Expression (P)

Step 2:- Scan P from Left to Right & repeat step 3 & 4 for each element of p until ")" is encount.

Step 3:- If an operand is encountered put it on stack

Step 4:- If an operator (⊗) is encountered then

(a) Remove the top elemend of stack, where A is the top and B is next to top element

(b) Evaluate B⊗A

(c) place result of (b) on stack

Step 5 Set value equal to top element

Step 6 Exit

Conversion of Infix to postfix :-

ALGORITHM:- Let Q be an airthematic operation written in infix notation. This algorithim find the equivalent expression into postfix expression

Step 1:- Push "(" on to stack and ")" to the end of Q

Step 2:- Scan Q from Left to Right & repeat Step 3 to 6 for element Q until the stack is empty

Step 3 If an operand is encountered, add it to p

Step 4: If an left paranthesis "(" is encountered, push it to stack

Step 5: If an ⊗ operator is encountered then
a) Repeatedly pop from stack & add to p each operator which has the same or high precedence than ⊗
b) Add ⊗ to stack.

Step 6: If an right parthesis ")" is encountered then
(a) Repeatedly pop from stack & add to p each operator until a left paranthesis is encountered
(b) Remove the left paranthesis

Step 7 Exit

Queue: A Queue is a non primitive linear data structure it is a homogeneous collection of element in which element are added at one end called rear; and existing element are deleted from the other end called front. Queue are also called First in First out (FIFO)

Inserting the element from Queue:

ALGORITHM- Set the size of Queue, Queue [n];

Step 2 Initialize Front = 0, Rear = 0

Step 3 If Rear => n (max size) [check overflow]
Write Queue overflow & return

Step 4 Rear = Rear + 1

Step 5 Queue [Rear] = item

Step 6 Return

Deleting the element from Queue :-

ALGORITHM: If front = 0 or NULL then write Queue is empty [underflow] & exit

Step 2 item = Queue [front]

Step 3 Front = Front + 1

Step 4 Go to Step 1

LinK List :- It is one way list. It consist of Nodes which has two part information and address part contain address of next node and end of linklist is contain a Null pointer

Insertion at begining

Step: (1) Check to see if free space is available in avail list

If Avail = null the print link list Overflow

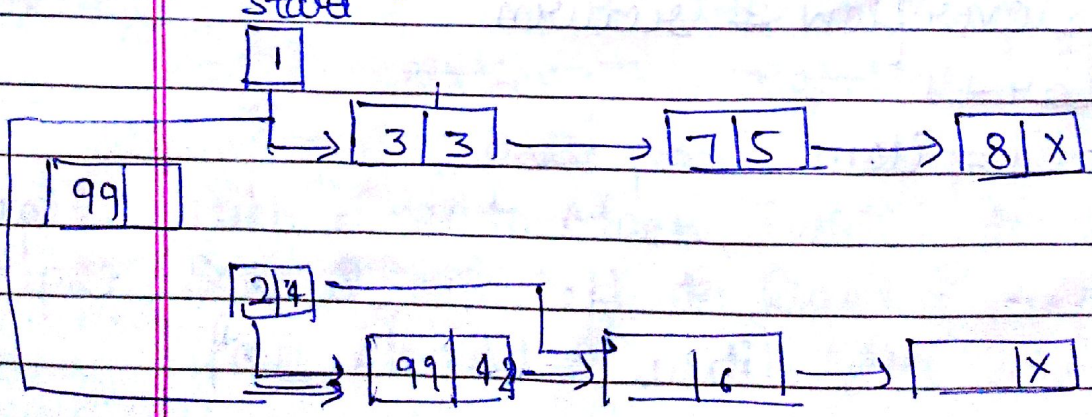
(2) Remove the first node from the avail list using the variable new to keep track of the location of the new node this stmt can be implemented by

New = Avail

Avail = LinK [Avail]

(3) Copy new information into new node this can be implemented as

info [new] = item



ALGORITHM:- [Overflow] If Avail = NULL then write overflow and exit

Step 2:- [Remove first node from avail list]
Set New := Avail, Avail = Link [new]

Step 3:- Set info [new] = item [copies new data]

Step 4:- Set Link [new] = Start [new node now points to original first node]

Step 5 & Set Start = new [changes start to new points to original new node]

Step 6 Exit

Insertion at middle

Step 1:- [Overflow] If avail = NULL then write overflow and exit

Step 2 [Remove first node from list]
Set New := Avail and Avail = Link [Avail]

Step 3:- Set Info [New] = item

Step 4 If LOC = NULL then [Insert at first node]
Set Link [new] = Start and start = new
else [Insert after node with location]

Step 5:- Set LINK [New] = LINK [LOC] & LINK [LOC] = new

Step 6:- Exit

Deletion at middle:-

Step 1:- If LOCP = NULL then START = Link [start]
else Set LINK [LOCP] = LINK [LOC]

Step 2:- [Returned Deleted node to avail]
Set LINK [LOC] := Avail & Avail LOC

Step 3 Exit

Sorting :-

Bubble Sort :-

eg :-

4	3	3	3	3	1
3	4	4	4	1	3
5	5	5	1	4	4
10	10	1	5	5	5
1	1	10	10	10	10

ALGORITHM :-

1. [Initialization] Read a[i], n
   Set i = 1
2. Repeat Step 3, 4 while J <= N-1
3. Repeat Step 4 while i <= N
4. If a[i] > a[i+1]
   then swap = a[i]
   a[i] = a[i+1]
   a[i+1] = swap
5. end of 5 step, 4 step & 3 step
6. Print the Stmt
7. Exit

Insertion Sort

eg

10	9	8	7	6	5
9	10	9	8	7	6
8	8	10	9	8	7
11	11	11	10	9	8
7	7	7	11	10	9
6	6	6	6	11	10
5	5	5	5	5	11

ALGORITHM :- Set A[0] = $-\infty$

2. Repeat Step 3 to 5 for K = 2, 3, ... N
3. Set TEMP = A[K] and PTR := K-1

④ Repeat while TEMP < A[PTR]
(a) Set A[PTR+1] = A[PTR]
(b) Set PTR := PTR-1
[End of loop]
⑤ Set A[PTR+1] = TEMP
[End of step 2 loop]
⑥ Return

Selection Sort :-
① Repeat Step 2 and 3 for K = 1, 2, --- N-1
② Call MIN(A, K, N, LOC)
③ [Interchange A[K] and A[LOC]]
Set TEMP = A[K]; A[K] = A[LOC] and A[LOC] = TEM
End of Step 1 loop
④ Exit

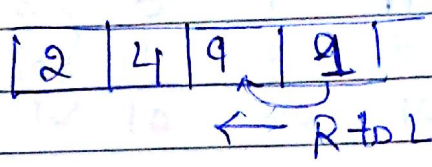
→ ALGORITHM :- It is a well define computational procedure that take some value or set of value as input and produces some values or set of values as output
An algorithm is a sequence of computational steps that transforms the input into the output

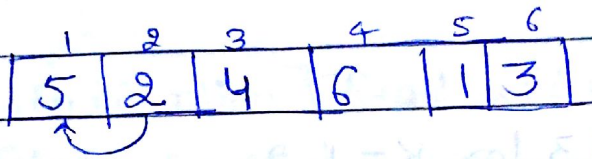
Factor to decide the Best Algorithm :-
① No of item to be Sorted
② Input values are already sorted
③ Architechture of the computer and the kind of storage devices to be used
④ An algorithm is set to be correct for every input instance, it hold with correct output which Solv
Computational problem
⑤ An incorrect algorithm might not fault at all or Input
some instances or it might end with an incorrect answers

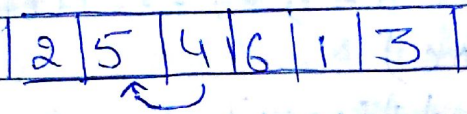
Passion
Page No.
Date

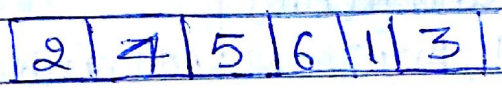
→ Data structure : It is a way to store organised data in order to facilate access and modification

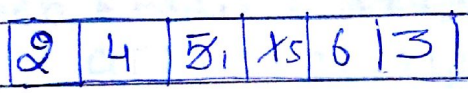
Insertion Sort : Easiest Sorted method

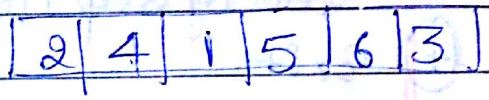


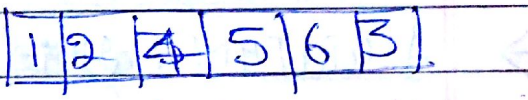












eg 31 41 59 26 41 58

31 41 26 59 41 58
31 26 41 59 41 58
26 31 41 59 41 58
26 31 41 41 59 58
26 31 41 41 58 59

Algorithm:
1) for j = 2 to A length
2) Key = A[j]
3) // Insert A[j] into sorted sequence A[1...j-1]
4) i = j-1
5) while i > 0 and A[i] > Key

6) $A[i+1] = A[i]$

7) $i = i-1$

8) $A[i+1] = key$

Analysis :-

Cost	times
C_1	n
C_2	$n-1$
C_4	$n-1$
C_5	$\sum_{j=2}^{n} t_j$
C_6	$\sum_{j=2}^{n} t_j - 1$
C_7	$\sum_{j=2}^{n} t_j - 1$
C_8	$n-1$

$$T(n) = C_1 n + C_2(n-1) + C_4(n-1) + C_5 \sum_{j=2}^{n} t_j + C_6 \sum_{j=2}^{n} (t_j - 1) + C_7 \sum_{j=2}^{n} (t_j - 1) + C_8(n-1)$$

$$= C_1 n + C_2 n - C_2 + C_4 n - C_4 + C_5(n-1) + C_8(n-1)$$

$$= (n)(C_1 + C_2 + C_4 + C_5 + C_8) - (C_2 + C_4 + C_5 + C_8)$$

$\neq$ complexity.

This running time as an + b for constants a & b that depend upon statement cost C_I is a linear function of N in the reverse order. If we array is, the reverse order. If we must compare each element array[j] with

each element in the entered subscripted a[1] to a[j-1] and so $t_j = j$ for 2, 3, 4, ... n

$1+2+3+\cdots+n-1$

$C_5 = \frac{n(n+1)}{2} - 1$

C_6 $\sum_{j=2}^{n} (j-1) = \frac{n(n-1)}{2}$

$1+2+3+4+\cdots+(n-1)+n-n$

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

$= \frac{n(n+1)-2n}{2} = \frac{n^2+n-2n}{2} = \frac{n^2-n}{2}$

$= \frac{n(n-1)}{2}$

$T(n) = C_1 n + C_2(n-1) + C_4(n-1) + C_5\left(\frac{n(n+1)}{2} - 1\right)$

$+ C_6\left(\frac{n(n-1)}{2}\right) + C_7\left(\frac{n(n-1)}{2}\right) + C_8(n-1)$

$= n(C_1+C_2+C_4) - (C_2+C_4) + C_5\left(\frac{n^2+n-2}{2}\right) + \frac{C_6 n^2}{2} - \frac{C_6 n}{2}$

$+ \frac{C_7 n^2}{2} - \frac{C_7 n^2}{2} + C_8 n - C_8$

$= n^2\left(\frac{C_5}{2} + \frac{C_6}{2} - \frac{C_7}{2}\right) + n\left(C_1 + C_2 + C_4 + C_8 + \frac{C_5}{2} - \frac{C_6}{2}\right)$

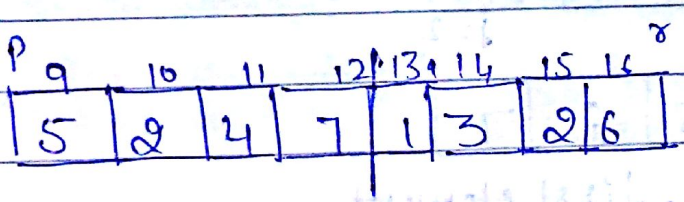
$-(C_2 + C_4 + C_5 + C_8)$

$ax^2 + bx + c$

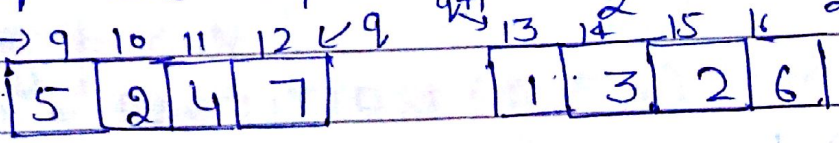
We can express this worst case running time as ax^2+bx+c for constant a, b & c that again depend upon statement cost C05 It is a quadratic function of n

Divide and Conquer approach:
Divide
Conquer
Combine

Some Algorithm are recurssive in structure to solve a given problem they call themselves reccursively one or more time to deal with closely related some programs
These Algorithm follows a divide and conquer approach They break a program into several such programs that are similar to original problem but small in size. Solve the sub problem reccursively then combine this solution to create a solution to original program



$q = \lfloor (p+r)/2 \rfloor = \frac{9+16}{2} = \frac{25}{2} = \lfloor 12.5 \rfloor = 12$



$\lfloor \frac{(p+q)}{2} \rfloor = \frac{9+12}{2} = \frac{21}{2} = \lfloor 10.5 \rfloor = 10$

p→ 9 10 ←q q+1 11 12

| 5 | 2 | | 4 | 7 |

$\lfloor \frac{9+10}{2} \rfloor = \lfloor \frac{19}{2} \rfloor = \lfloor 9.5 \rfloor = 9$

9 10

| 5 | | 2 |

$\lfloor \frac{11+12}{2} \rfloor = \lfloor \frac{23}{2} \rfloor = \lfloor 12.5 \rfloor = 12$

| 5 | 2 | 4 | 7 |

$\lfloor \frac{13+16}{2} \rfloor = \lfloor \frac{29}{2} \rfloor = \lfloor 14.5 \rfloor = 14$

| 5 | 2 | 4 | 7 | 1 | 3 | 2 | 6 |

| 5 | 2 | 4 | 7 | 1 | 3 | 2 | 6 |

| 2 | 5 | 4 | 7 | 1 | 3 | 2 | 6 |

| 2 | 4 | 5 | 7 | ∞ |

i=1

| 1 | 2 | 3 | 6 | ∞ |

j=1 ↓ j=2

| 1 | 2 | 2 | 3 | 4 | 5 | 6 | 7 |

Algorithm: Merge sort (A, p, r) → last element
↑ array, ↑ first element

1) if $p < r$
2) $q = \lfloor \frac{(p+r)}{2} \rfloor$
3) MERGE SORT (A, p, q)
4) MERGE SORT (A, q+1, r)

Combine 5) MERGE (A, P, q, r)

1.)	$n_1 = q - p + 1$	$\Rightarrow (12 - 9 + 1) = 4$
2)	$n_2 = r - q$	$= (16 - 12) = 4$
3)	Let $L(1 \cdots n_1 + 1)$ and $R[1 \cdots n_2 + 1]$ be new array	
4)	for ($i = 1$ to n_1)	($i = 1$ to 4)
5)	$L[i] = A[p + i - 1]$	$A[9 + 1 - 1] = 9$
6)	for ($j = 1$ to n_2)	($J = 1$ to 4)
7)	$R[j] = A[q + j]$	$A[12 + 1] = A[13]$
8)	$L[n_1 + 1] = \infty$	$L[4 + 1] = 5$
9)	$R[n_2 + 1] = \infty$	$R[4 + 1] = 5$
10)	$i = 1$	$i = 1$
11)	$j = 1$	$j = 1$
12)	for ($K = p$ to r	$K = 9$ to 16
13)	A if $L[i] \leq R[j]$	$L[1] \leq R[1]$
14)	$A[K] = L[i]$	$A[P] = L[1]$
15)	$i = i + 1$	$i = 1 + 1 = 2$
16)	else $A[K] = R[j]$	$A[9] = R[1]$
17)	$j = j + 1$	$J = 1 + 1 = 2$

Quick sort: Application of Divide and Conquer

i	p, 2	3	4	5	6	7	8 r
	2	8	7	1	3	5	6	4

Algorithm: QuickSort(A, P, r)

1.) If $p < r$
2) q = PARTITION (A, P, r)
3) QuickSort (A, P, q)
4) QuickSort (A, q+1, r)
5) PARTITION (A, p, r)
1. $x = A[r]$ // $x = 4$

i = p-1; ⇒ i=0

for j = p to r-1

if A[j] ≤ x

i = i+1

Exchange A[i] with A[j]

Exchange A[i+1] with A[r]

return i+1

```
for(j = 1; j<7; j++)
{
  if (A[j] ≤ x)
  {
    i = i+1
    Swap A[i] & A[j]
  }
}
```