Finiteness: An algorithm must terminate after a finite no of steps.

Definiteness: The steps of the algorithm must be precisely defined (or) unambiguously specified.

Generality: an algorithm must be generic enough to solve all problems of a particular class.

Effectiveness: The Operations of the algorithm must be basic enough to be put down on pencil & paper. They should not be too complex.

I/O: the algorithm must have certain initial & precise inputs & outputs that Have be generated both at its intermediate and final steps.

// An algorithm doesnot enforce a language (or) mode for its expression but only demands adherence to its properties.

⇒ add a pinch of salt & pepper' 'fry until it turns golden brown' 'pinch & golden brown' are subject to ambiguity.

⇒ Alg represented using pictorial repretns such as flow charts.

→ encoded in a programming lang for implementation on a computer is called program.

* Dev't of an algorithm:

Steps involved in the dev of alg:

1. problem statement
2. Model formulation

list.

(3) Algorithm design
(4) Algorithm correctness
(5) Implementation
(6) Algorithm Analysis
(7) program Testing
(8) Documentation.

(1) Once a clear statement of problem is done,
(2) Model for the solution is formulated
(3) Design the Alg based on the solution Model that is formulated
(4) * here we can see the role of DS.
* The right choice of data structure needs to be made at the design stage itself.
* data structure Influences the efficiency of the algorithm.
(4) Once correctness of the algorithm is checked, and the algorithm Implemented.
(5) Most imp is measuring the Performance of the alg is done. (i,e termed as algorithm Analysis".

(It can be seen how the use of appropriate data structures results in better performance).

(6) program tested and Dev ends with proper documentation.

Efficiency of Alg.

Performance of the alg can be measured on the scales of time and space.

→ The problem which performs its task in the minimum possible time

→ alg (or) consumes (or) needs limited memory space to execution

$= n) = n$.

Frequency count (or) Stepcount:

We can calculate Time complexity ~~either~~ by using

(1) Frequency count

(2) ~~step count~~.

(1) → It specifies how many times a statement is executed.

1. For comments, declarations, —— S.C = 0.
2. Return, Assignments ——— S.C = 1
   - ↳ We can return value
   - Assign.
3. Ignore lower order exponents when higher order exponents are present.

   ⟶ Ignore

   eg: $3n^4 + 4n^3 + 10n^2 + n + 100$

   $3n^4 = O(n^4)$.
4. Ignore constant multiples.

eg:

```
int sum(int a[], int n)
{
    s=0;  ——— Assignment S.C — 1
    for(i=0; i<n; i++)
        s = s+a[i];
    return s; ——→ 1.
}
```

Let n = 3, i=0,

3 → 1

1<3 — 2

2<3 — 3

3<3 — false but condition checked.

u → (n+1)

Body will be executed 'n' times,

→ 1
→ n+1
+ n
+ 1
2n+3 → Time complexity = O(n)

→ Higher order Exponential (b)

Asymptotic Notations:

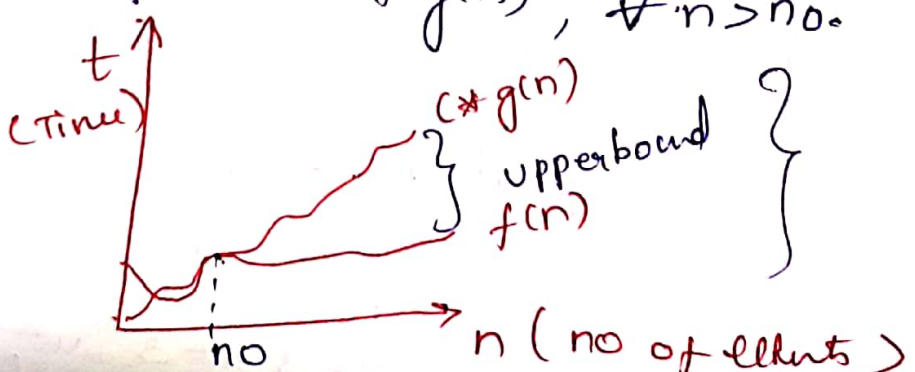
We can use these Notations we can calculate Time complexities of an alg.

1. Big oh Notation
2. Big Omega Notation
3. Theta Notation.

Big oh: Mainly represents upperbound of Algs runtime,
→ TO calculate max amount of time taken by the alg.
→ Worst Case T.C.

Def: Let $f(n), g(n)$ be two Non-Negative functions, then $f(n) = O(g(n))$ if there Exists two positive constants c, no, Such that

$$f(n) \leq c * g(n), \quad \forall n > n_0.$$



$f(n) = 3n+2$, $g(n)=n$.

$f(n) = O(g(n))$

$f(n) \le c * g(n), \forall\ n > n_0$.

Assume $c = 5$ | Let $n = 4$

$3n+2 \le c*n$

$n=2$: $8 \le 4$ False

$n=4$: $14 \le 20$ True

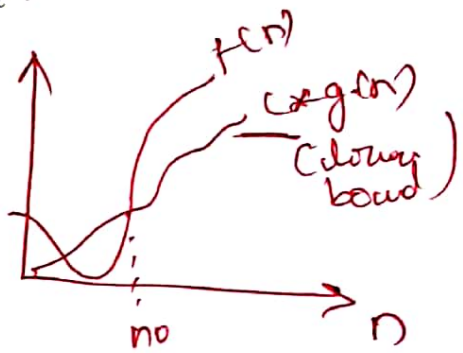
finally, $n_0 > 2$ should be greater.

Big Omega (Ω): to represents lower bound of alg.

→ to calculate min amount of time.

→ B.C Tic.

$f(n) \ge c*g(n)$.



$f(n) = 3n+2$, $g(n) = n$

$f(n) \ge c*g(n)$.

$n=1$: $3n+2 \ge c*n$ → $5 \ge 1$

$n_0 = 2$: $8 \ge 4$

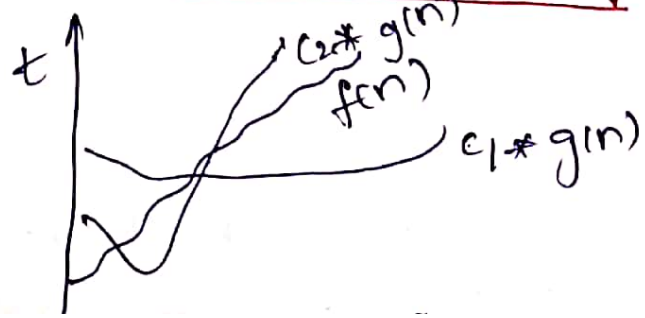
$n_0 \ge 1$

Let $c=1$

Θ Notation: average bound of an alg.

$f(n) = \Theta(g(n))$, if there Exist 3 +ve constants c_1, c_2, n_0. such that

$$c_1 * g(n) \le f(n) \le c_2 * g(n)$$



- $16n^3+78n^2+12n \rightarrow O(n3)$
  $16n^3 \rightarrow$
- $34n+90 \rightarrow O(n)$
- $56 \rightarrow$ constant $O(1)$.

Polynomial Notation:

polynomial algorithms include quadratic algorithms $O(n^2)$, cubic algorithms $O(n^3)$

represents an algorithm whose performance is directly proportional to the square of the size of the dataset.

Exponential Notation:

The exponential Notation $O(2^n)$ describes an algorithm whose growth doubles with each addition to the data set.

n	n^2	2^n
$n=1$,	$1^2=1$	$2^1=2$
$=2$,	$2^2=4$	$2^2=4$
$=3$	$3^2=9$	$2^3=8$
$=4$	$4^2=16$	$2^4=16$
$=5$	$5^2=25$	$25=32$
$=6$	$6^2=36$	$2^6=64$ ↑
$=7$	$7^2=49$	$2^7=128$

Average, Best, Worst case complexities:

Time complexity of an algorithm is dependent on parameter associated with the input/output instances of the problem.

eg:

I/p_1: -1, -23, -11, -2, -4, -6, -7, -8, -9, 2

I/p_2: 1, -7, 11, -2, -6, -8, -7, -9

Search for the first occuring even Number in the list.

* In case of input, it takes 10 comparisons to find all

* In case of Input 2, the first element is posive element.

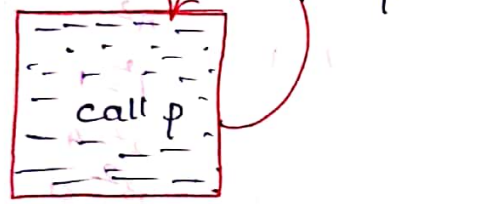
16n:

(This justifies the statement that the running time of algorithms are not just dependent on the size of the i/p but also on its nature.

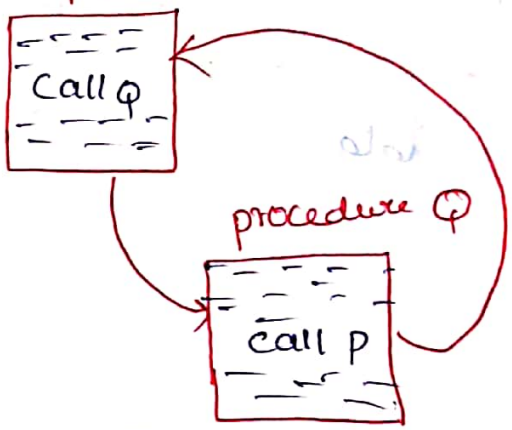
Analyzing Recursive Algorithms:

If 'p' is a procedure containing a call statement to itself. (or) to another procedure that results in a call to itself, the the procedure P is said to be a recursive procedure.

Inge



Direct Recursion



Indirect Recursion.

Eg: Recursion is used to solve Towers of Hanoi problems.

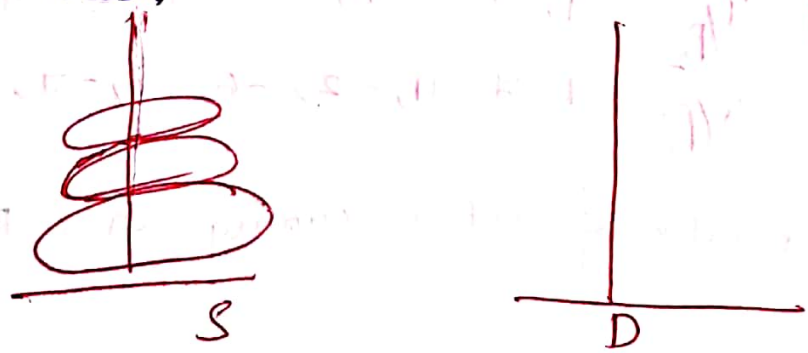
Towers of Hanoi problems:

* Tower of Hanoi is a Mathematical puzzle where we have three rods and 'n' disks.

* The objective of the puzzle is to move the entire stack to another rod.

Rules:

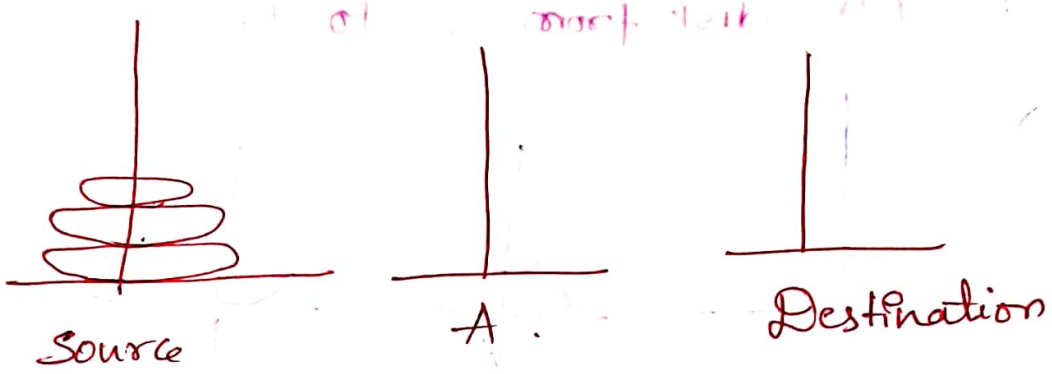
*



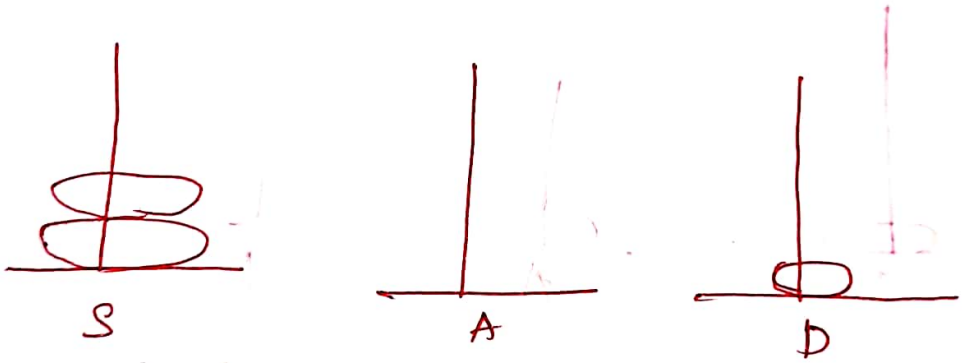
(1) : only one disk can be moved at a time.

(2) Larger disk can't be placed on the top of smaller disk.

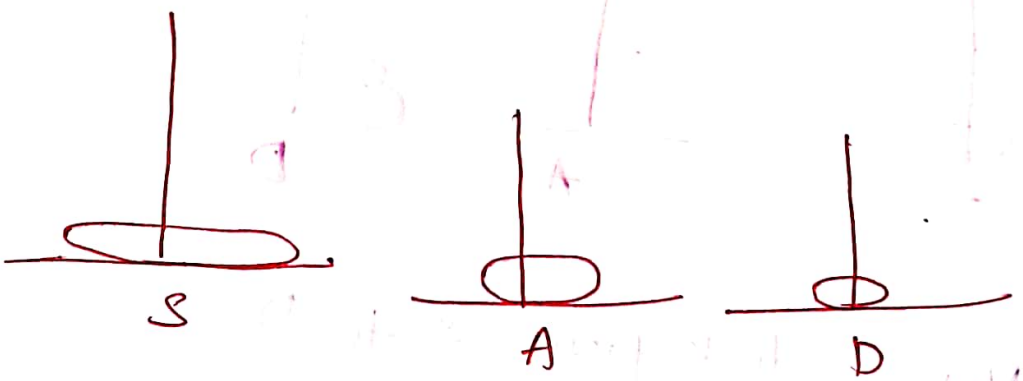
(3) We can use Auxiliary tower for temporary storage of disks.



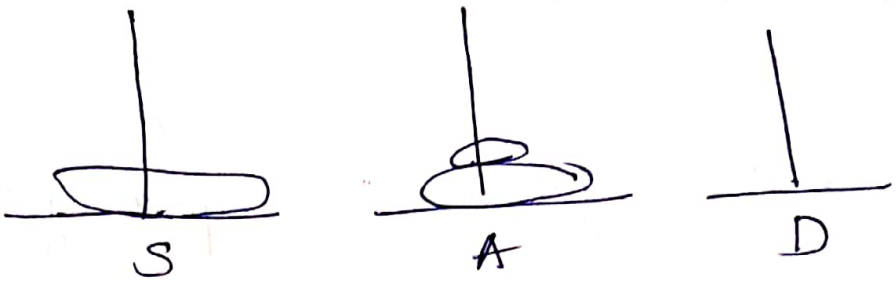
Step 1: Move disk from S to D



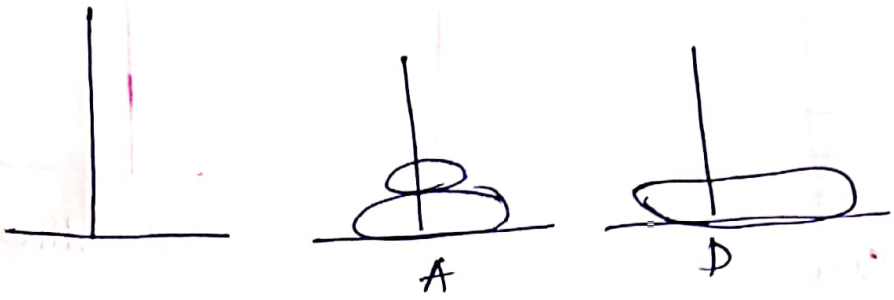
Step 2: Move Disk from S to A



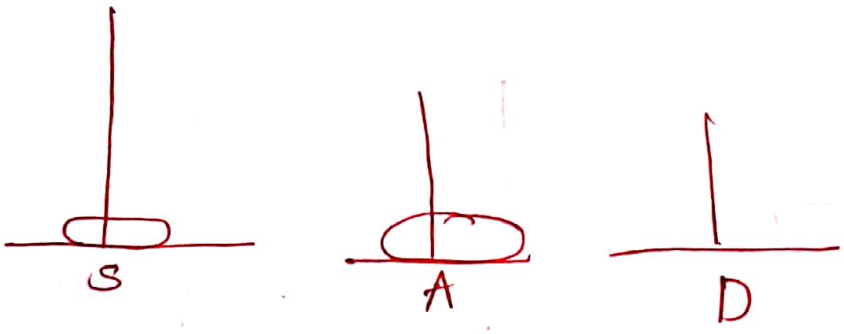
Step 3: Move disk from D to A.



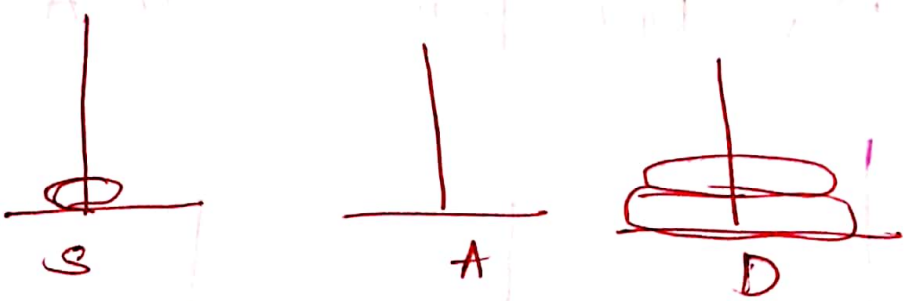
Step 4 Move disk from S to D



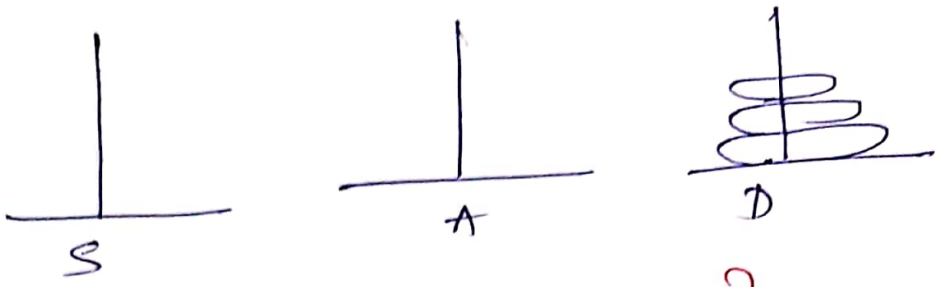
Step 5: Move disk from A to S.



Step 6: Move disk from A to D.



Step 7: Move disk from S to D



No of disks = 3

No of moves required = $2^n - 1$

eg ⇒ $n = 5$

$2^n - 1 = 32 - 1 \Rightarrow 31$ moves required.

Sequential Search:

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
#define MAX_SIZE 5
using namespace std;
int main()
{
  int arr_search[MAX_SIZE], i, element;
  cout<<"Linear search Example\n";
  cout<<"\n Enter"<<MAX_SIZE<<"Elements for
                                Searching:";
for(i=0; i<MAX_SIZE; i++)
    cin>>arr_search[i];
cout<<"\n you Data";
for(i=0; i<MAX_SIZE; i++)
  {
    cout<<"\t"<<arr_search[i]
  }
cout<<"\n Enter Element to search:";
  cin>>Element;
```

(5)

(5)

0<5,

to read

Take first element.

to print.

```
for(i=0; i<MAX-SIZE; i++)
{
    if(arr-search[i]==element)
    {
      cout<<" element"<<element<<"found :
            position :"<<i+1;
      break;
    }
}
  if(i==MAX-SIZE)
    cout<<" element"<<element<<"Not-found";
  getch();
}
```

Sample Input/output:

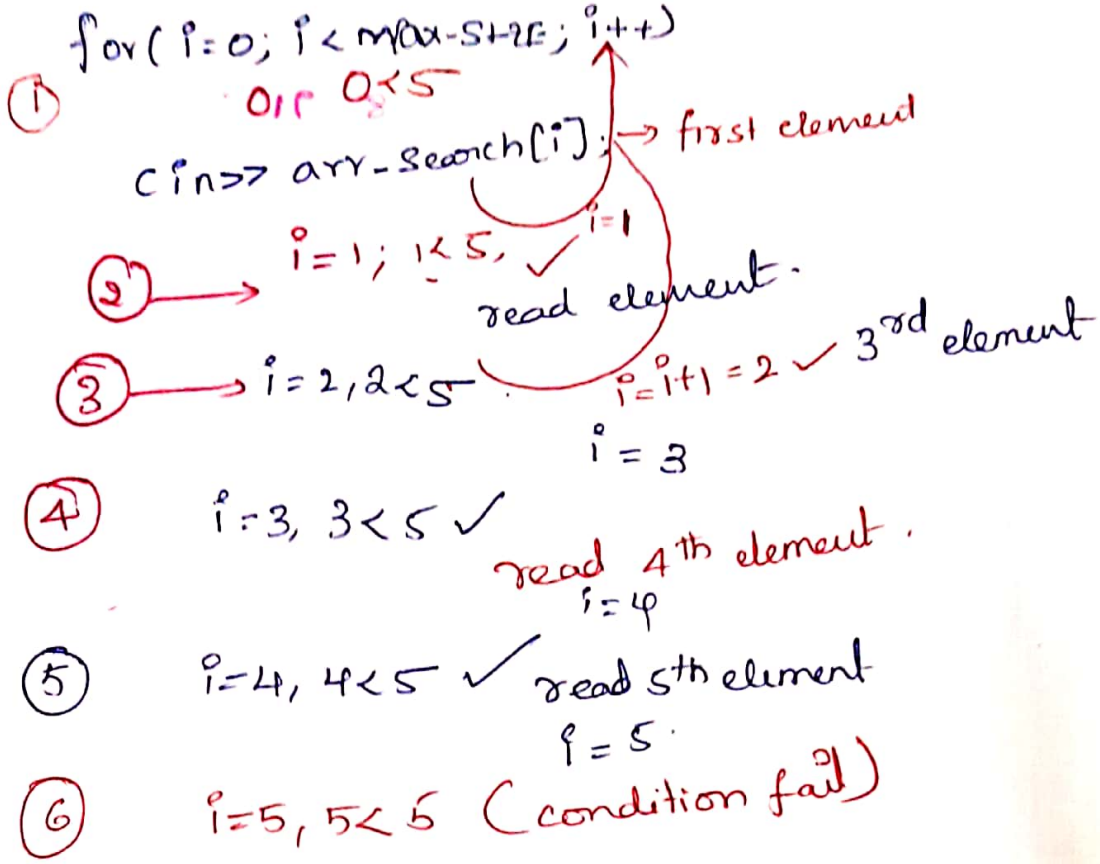
Linear Search Example.

Enter 5 elements for Searching := 20

40
80
79
68

Enter element to Search: 79.

Element found position: 4.



Next it execute cout<<

```
// for(i=0; i< MAX-SIZE; i++)
   {
      cout<<" IF"<< arr-search[i];
   }
```

To print each element.

```
// cout<< enter element"
```

Search element.

```
// for( i=0; i<MAX-SIZE; i++)
   {
      if (arr-search[i]==element)
   {
```

Binary Search

0	1	2	3	4	5	6	7	8	9
10	69	23	45						

0	1	2	3	4	5	6	7	8	9
2	17	36	69	72	88	89	96	98	99

low = 0 high = 9

{Search Element = 89}

(1) Calculate Mid.

$$Mid = \frac{low + high}{2}$$

$$= \frac{0+9}{2}$$

$$= \frac{9}{2} = 4.5 \simeq 4$$

(2)

0	1	2	3	4	5	6	7	8	9
2	17	36	69	72	88	89	96	98	99

Mid = 72

Search Element = 89.

72 < 89 → key to be Searched.

↓ Mid value.

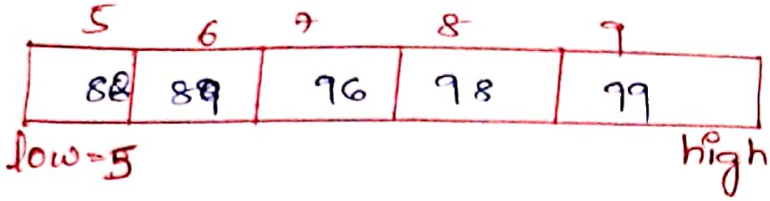
{Note: Element is greater than the Mid value
So we can skip left part of the data}

0	1	2	3	4	5	6	7	8	9
2	17	36	69	72	88	89	96	98	99

Mid

Skip.

(3) Consider New data.



④ calculate Mid.

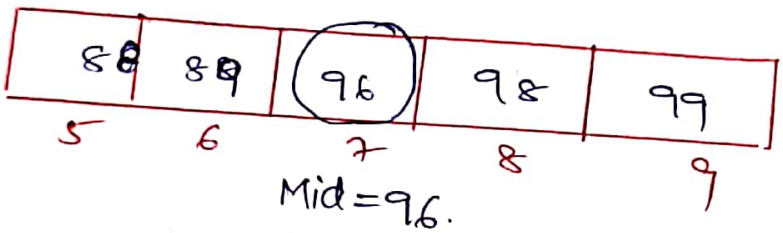
$$Mid = \frac{low + high}{2}$$

$$= \frac{5+9}{2}$$

$$= \frac{14}{2}$$

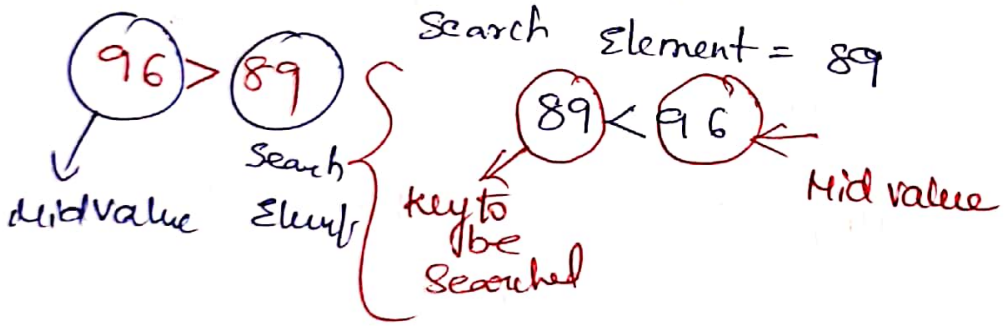
$$Mid = 7.$$

⑤

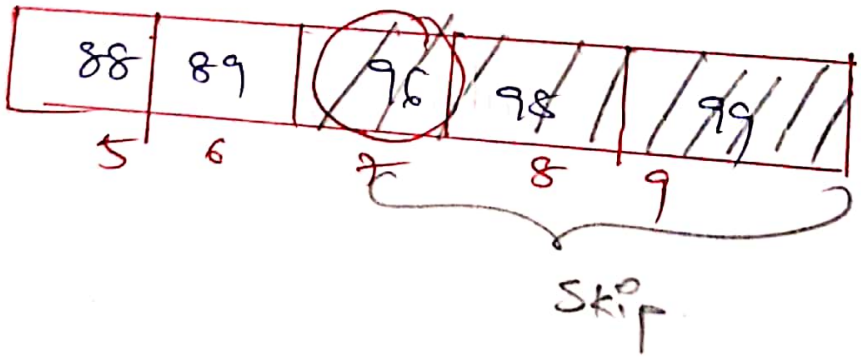


Mid = 96.

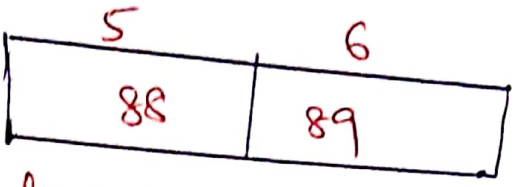
Search Element = 89



Note: Mid value is greater than the Search Element we can skip right part. of the Mid.



(6)



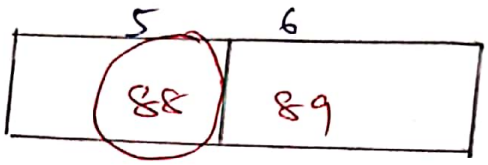
low = 5 high = 6

Calculate Mid = $\frac{low + high}{2}$

$= \frac{5+6}{2}$

Mid $= \frac{11}{2} = 5.5 \approx 5$

(7)

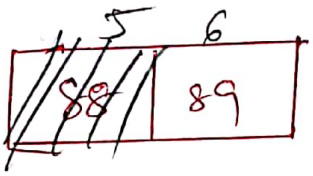


Mid = 88

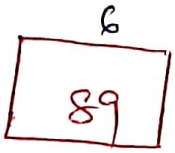
Mid (88) < Search Elemt

88 < 89 (Skip the left part.

(8)



(9)



l = 6 high = 6.

When low == high, we can say that Element found.

eg:

Fibonacci Search:

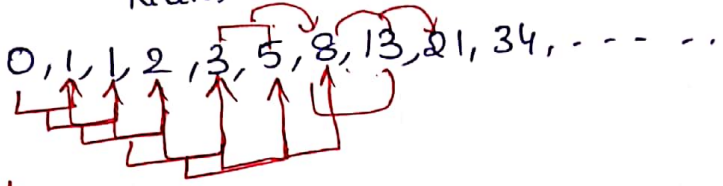
Febnocci Search is a comprison-based technique. that uses Febnacci Numbers to search an element in a Sorted Array.

Fibonacci Numbers:

The first two numbers are '0' and '1' and each Subsequent numbers in the series is equal to the sum of the previous two numbers.

$$\begin{cases} F(n) = n & \text{when } n <= 1 \\ F(n) = F(n-1) + F(n-2) & \text{when } n > 1 \end{cases}$$

Fibonacci Numbers are



Here $1 = 0+1$

$2 = 1+1$

$3 = 1+2$

$5 = 2+3$

$8 = 3+5$

$13 = 8+5$

Step1: Find the smallest number >= n. Let the number be * fibm Let the two Fibonacci numbers preceding it be m_1, m_2.

* Imp.

Step2: While the array has elements.

① Compare x with last element of the range covered by m_2.

* Else if (x) less than the element move the 3 Fibonacci variables two Fibonacci down, indicating elimination of approximately rear two-third of the remaining array.

* Else (x) is greater than the element, move the three Fibonacci variables one Fibonacci down. It indicates elimination of the front one-third of the remaining array. (Reset offset to Index.)

(3) Since there might be a single element remaining for comparison. Check if M_1 is 1. If yes, compare x with that remaining element. If match return index.

Eg: Consider (n) of elements

(1) 16, 25, 39, 47, 86, 92, 98, 99.

(2) 10, 20, 30, 40, 50.

Sol:

0	1	2	3	4
10	20	30	40	50

Search Ele = 20

} n = no of elements
n = 5

Now, write Fibonacci series upto 'n'

0 1 1 2 3 5

Smallest Fibonacci number >= n

0 1 1 2 3 (5) ← fibm = 5

m_2 m_1

$m_1 = 3$

$m_2 = 2$

offset = 0.

{ preceding no's are $m_1 + m_2$

calculate Index $i = \min(\text{offset} + m_2, n)$

fibm = 5

$m_1 = 3$

$m_2 = 2$

$i = \min(0+2, 5)$

$= \min(2, 5)$

$i = 2$

Now. compare a[2] with search element.

x = 20; a[2] = 30

'20', x is Less than, Indexed Elment

Move two Fibonacci Variables down, it means.

0 1 1 2 3 5 } First case

m_2 m_1 M

Two varibles down means

Now M = 2

0	1	1	2	3	5

fibM = 2

$m_1 = 1$

$m_2 = 1$

Here offset Same

calculate Index

$i = \min(\text{offset} + m_2, n)$

$= \min(0+1, 5)$

$= \min(1, 5)$

$i = 1$, $a[i] = a[1] = 20$.

$x = 20$, $a[i] = 20$.
nothing but element found at index 'i'

(ii) 0 1 2 3 4 5 6 7
16, 25, 39, 47, 86, 92, 98, 99.

$n = 8$, Search Ele = 147

Fibonacci sequence.

0 1 1 2 3 5 8 13

Find Smallest Fibonacinum >= n.

fibm = 13
$m_1 = 8$
$m_2 = 5$

$i = \min(\text{offset} + m_2, n)$
$= \min(0 + 5, 13)$
$= \min(5, 13)$
$i = 5$
$a[i] = a[5] = 92$, $x = 47$.

'47' is smaller than a[i], so down '2'

Variables.

0 1 1 2 3 5 8 13
m_2 m_1 fibM

0 1 1 2 3 5 8 13

$m_2 = 2$ $m_1 = 3$ $M = 5$

$i = \min(\text{offset} + m_2, n)$
$= \min(0 + 2, 13)$
$= \min(2, 13)$
$i = 2 \Rightarrow a[2] = 39$

$x = 47$, $a[2] = 39$.

47(x) is greater than a[2], down only one variable.

0 1 1 2 3 5

$m_2 = 1$ $m_1 = 2$ $H = 3$

(offset = Index) (reset)

$i = 2$, offset = 2

$i = \min(\text{offset} + m_2, n)$

$= \min(2 + 1, 13)$

$= \min(3, 13)$

$i = 3 \Rightarrow a[3] = 47$,

$x = 47$, $a[3] = 47$.

Element found at '3'rd Index.

x

Algorithm:-

1. Select an element pivot from the array elements
2. Rearrange the elements in the array in such a way that elements that are less than the pivot appear before the pivot and all the elements greater than the pivot element come after it.
3. After such a partitioning, the pivot is placed in its final position. This is called partition operation.
4. Recursively sort the two sub-arrays thus obtained (one with sub list of values smaller than that of the pivot element and the other having higher value elements.

Technique:-

1. Set the index of the first element in the array to loc and left variables. Also, set the index of the last elements of the array to the right variable.

   loc = 0, left = 0, right = n-1

2. Start from the element pointed by right and scan the array from right to left, comparing each element on the way with the element pointed by the variable loc.

   i.e, a[loc] should be less than a[right].

   → If this is the case, then simply continue comparing until right becomes equal to loc. Once right = loc, it means the pivot has been placed in its correct position.

   → However, if at any point, we have a[loc] > a[right], then interchange the two values and jump to step 3.

   → set loc = right.

from the element pointed by left and scan the array left to right, comparing each element on the way with the element pointed by loc.

i.e a[loc] should be greater than a[left].

If that is the case, then simply continue comparing until left becomes equal to loc. Once left = loc it means the pivot has been placed in its correct position.

→ if at any point, we have a[loc] < a[left], then interchange the two values and jump to step 2.

→ set loc = left.

Eg:- 27 10 36 18 25 45 n=6

0 1 2 3 4 5

loc
left

right

→ Initially Scan from right to left

a[loc] < a[right]

27 10 36 18 25 45 ← decrease the value of right

loc
left

right

→ a[loc] > a[right], interchange two values, loc = right.

25 10 36 18 27 45

left →

right
loc

→ from left to right (a[loc] > a[left]) → increment the value of left

25 10 36 18 27 45

left →

right
loc

→

25 10 36 18 27 45

left

right
loc.

& a[loc] < a[left], interchang values set loc = left

25 10 27 18 36 45
left loc right

↓

Scan from right to left, Since a[loc] < a[right], decrement the value of right.

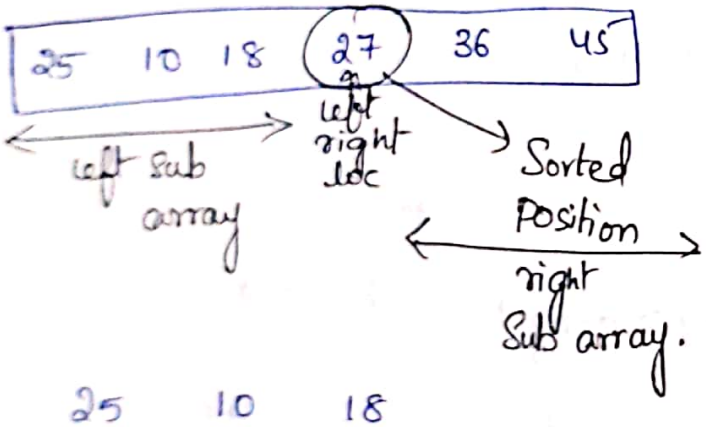
25 10 27 18 36 45
left loc right

↓

& a[loc] > a[right], interchange two values, loc = right

25 10 18 27 36 45
left loc right

Start Scanning from left to right, & a[loc] ≥ a[left], increment value to left



25 10 18
left loc right

a[loc] > a[right], ~~interchange~~ decrement.
Interchange

18 10 25
left right loc

→ a[left] < a[loc]

18 10 25
left right loc
→ increment.

a[left] < a[loc]

18 10 (25) ← left, right, loc — Sorted position

left sub array.

(15)

10	18	25	27	

18 10 25

left loc, right

← a[loc] < a[right], Enter change.

10 18

left, loc, right

a[left] > a[right loc]

10 > 18, Increment to left

(10) (18) ← Sorted position.

left loc right — left loc right

36	45

left loc, right

← a[loc] < a[right]

36 < 45

decrement right

(36) ← Sorted position.

left loc right

10	18	25	27	36	

(45) ← left, right loc

10	18	25	27	36	45

elements are Sorted.

① a[loc] < a[right]
decrement the value of right

a[loc] < a[left]
interchange the values

② a[loc] > a[right]
interchange values

a[loc] > a[left]
Increment the value of left

Quicksort.