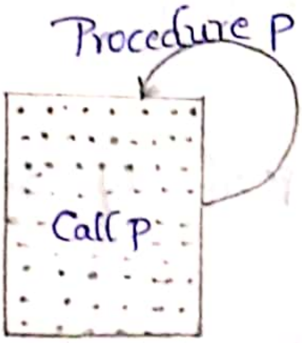
① How are recursive program analyzed?

Ans:- Recursion is an important concept and many algorithms can be best described in terms of recursion.

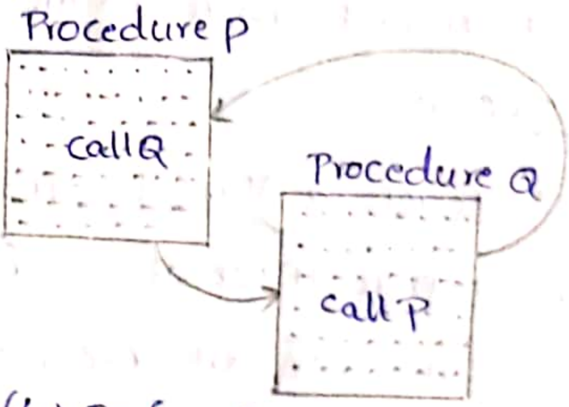
Recursive Procedures:

If P is a procedure containing a call statement to itself or to another procedure that results in a call to itself, then the Procedure P is said to be a "Recursive Procedure". In the former case it is termed "direct recursion" and in the latter case it is termed "Indirect recursion".

Extending concept to programming can yield program functions or Programs themselves that are recursively defined. In such cases they are reffered to as recursive functions and recursive Programs respectively.



(a) Direct recursion



(b) Indirect ~~Pro~~ recursion

In order that recursively defined function may not run into an infinite loop it is essential that following Properties are satisfied by an recursive Procedure:-

(i) There must be criteria, one or more, called base criteria or simply base case(s), where Procedure does not call itself either directly or indirectly.

(ii) Each time the Procedure calls itself directly or indirectly, it must be closer to the base criteria.

Example:-

1) Recursion Process is used to solve Towers of hanoi Problem.

2) And it is also used to compute factorial of a number n.

② Write about the following searching techniques with example?

- Binary Search
- Fibonacci search.

Ans:- Binary Search:-

Binary search is an efficient algorithm for finding an item from a sorted list of items.

→ It works by repeatedly dividing in half portion of list that could contain item, until you have narrowed down possible locations to just one.

Process:-

→ Search a sorted array by repeatedly dividing the search interval in half.

→ Begin with an interval covering whole array. If value of search key is less than item in middle of interval, narrow interval to lower half, otherwise to upper half.

→ Repeatedly check until value is found or interval is empty.

$$\boxed{\text{Mid value} = \frac{\text{low index} + \text{high index}}{2}}$$

Example :-

Sorted list :- 7, 34, 56, 67, 72, 87, 92, 94, 111, 115

Search element :- 92

0	1	2	3	4	5	6	7	8	9
7	34	56	67	72	87	92	94	111	115

low = 0 high = 9

① Calculate Mid

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

$$= \frac{0+9}{2} = \frac{9}{2} = 4.5 \approx 4$$

②

7	34	56	67	72	87	92	94	111	115
0	1	2	3	4	5	6	7	8	9

Mid = 72

Search element = 92

72 < 92 → (search element)

[Element is greater than Mid value so we can skip left Part of data]

~~7~~	~~34~~	~~56~~	~~67~~	~~72~~	87	92	94	111	115

Mid

(Skip)

③ Consider New data

5	6	7	8	9
87	92	94	111	115

low = 5 high = 9

④ Calculate Mid

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

$$= \frac{5+9}{2} = \frac{14}{2} = 7 \quad \boxed{\text{Mid} = 7}$$

(5)

87	92	94	111	115

Mid value = 94

92 < 94

Key value Mid value

[Mid value is greater than search element we can skip right Part of Mid]

87	92	94	111	115

skip

(6)

5	6
87	92

low=5 high=6

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

$= \frac{5+6}{2} = \frac{11}{2} = 5.5 \approx 5$

Mid = 5

(7)

5	6
87	8 92

Mid Value = 87

88<92

(so skip left Part)

(8)

5	6
87	92

(9)

6
92

low=6 high=6

When low=high, we can say that element is found.

so element found at Position with index value 6.

Fibonacci Search:-

Fibonacci search is a Comparision-based technique. It uses Fibonacci numbers to search an element in a sorted array.

Process:-

Step 1:-

Find the smallest number >= n, Let the number be fibm. Let the two fibonacci numbers Preceding it be m_1, m_2.

Step-2:- 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-thired of the remaining array.

* Else 'x' is greater than element, Move three fibonacci variables one fibonacci down. It indicates elimination of front one-third of remaining array. Reset offset to Index.

Step-3:- Since there might be a single element remaining for comparision. check if M1 is 1. If yes, compare x with that remaining element. If match return index.

Example:- Sorted Array :- 10, 20, 30, 40, 50

Search element :- 20

n, no:of elements = 5

Now write fibonacci series upto n

0 1 1 2 3 5

Smallest fibonacci number >= n

$0\ 1\ 1\ \underset{m_2}{2}\ \underset{m_1}{3}\ \boxed{5} \leftarrow$ fibm

fibm = 5
Preceding numbers m_1 & m_2

$m_1 = 3$

$m_2 = 2$

Offset = 0

Calculate Index $\boxed{i = \min(offet + m_2, n)}$

$fibm = 5$

$m_1 = 3$

$m_2 = 2$

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

$= \min(2,5)$

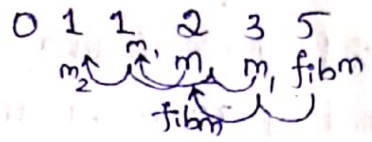
$i = 2$

Now Compare a[2] with search element

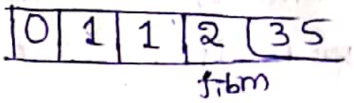
$x = 20;\ a[2] = 30$

20, x is less than indexed element

so, Move two fibonacci variables down, it means



∴ Now fibm = 2



$fibm = 2$

$m_1 = 1$

$m_2 = 1$

Calculate index $i = \min(offset + m_2, n)$

$= \min(0+1, 5)$ [in this case offset is Same]

$= \min(1,5)$

$i = 1$

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

$x = 20,\ a[1] = 20$

∴ Element found at the Index '1'.

(3) Write about algorithm for bubble sort with example?

Ans:- Bubble sort :-

Bubble sort is a very simple method that sorts array elements by repeatedly moving largest element to the highest index Position of array.

→ In bubble sorting, consecutive adjacent Pairs of elements in the array are compared with each other.

→ If element at lower index is greater than element at higher index, the two elements are interchanged.

→ The Process will continue till the list of unsorted elements exhausts.

→ This Procedure is called Bubble sorting because Elements bubble to top of list.

Technique :-

a) In Pass 1, [A_0] and [A_1] are Compared then [A1] is Compared with [A_2], [A_2] is compared with [A_3]...... Finally A[N-2] is Compared with A[N-1]. Pass1 involves N-1 Comparisions & Places biggest element at highest index of array.

b) In Pass2, [A_0] & [A_1] Compared, then [A1] is A_1 compared with [A_2], [A_2] is Compared with [A_3], so on. Finally A[N-3] is Compared with A[N-2]. Pass2, involves n-2 Comparisions and Places second biggest element at 2nd ~~big~~ highest index of array.

(c) In Pass 3, [A_0] & [A_1] Compared with [A_2] Then [A1] Compared with [A_2] so,on. Finally A[N-4] Compared with A[N-3]. Pass 3 involves n-3 Comparisions & places 3rd biggest element at third highest index of array.

(d) In Pass n-1, [A_0] & [A1] are Compared, so that [A_0]<[A_1]. After this step all elements of the array are arranged in ascending order.

Eg:- A[ ] = {34,42,12,56,23,8}

Pass1:-

a) Compare 34 and 42, since 34<42, no swapping

b) Compare 42 and 12, since 42>12, swapping requir

34,12,42,56,23,8

c) Compare 42 and 56, since 42<56 no swapping

34,12,42,56,23,8

d) Compare 56 and 23, since 56>23 swapping requir

34,12,42,23,56,8

e) compare 56 and 8, since 56>8 swapping required

34,12,42,23,8,56

Pass2

a) Compare 34 and 12, since 34>12, swapping required

12,34,42,23,8,56

b) Compare 34 and 42, since 34<42 no swapping

c) compare 42 and 23, since ~~23~~42>23 swap

12,34,23,42,8,56

d) Compare 42 and 8, since 42>8, swap required

12,34,23,8,42,56

Pass3:-

a) compare 12 and 34, since 12<34, no swapping

b) compare 34 and 23, since 34>23, swapping requir

12,23,34,8,42,56

c) Compare 34 and 8, since 34>8, swapping requir

12,23,~~34~~,8,34,42,56

Pass4:-

a) Compare 12 and 23, since 12<23, no swapping

b) Compare 23 and 8, since 23>8, swap required

12, 8, 23, 34, 42, 56

Passs5;-

Compare 12and 8 since 12>8, swapping required

8, 12, 23, 34, 42, 56.

Compare 12 and 23 since 12<23 no soap

Pass $\boxed{8, 12, 23, 34, 42, 56}$

↳ sorted list.