1/2/23
UNIT - II

FUNDAMENTALS OF THE ANALYSIS OF ALGORITHM EFFICIENCY

Best, average and worst case analysis - Linear Search - ⓧ Amortized efficiency - Mathematical analysis of non recursive algorithm - Mathematical ⓧ analysis of recursive algorithm - Example : Fibonacci number.

Best, average and worst case analysis

Best - case analysis

The process of identifying the best case input of size n for an algorithm, (that takes the lowest no. of operations during execution) and counting the no. of times the basic operation is executed.

It gives the lower bound of running time for an algorithm.

worst - case analysis

The process of identifying the worst case input of size n for an algorithm, (that takes the highest no. of operations during execution) and counting the no. of times the basic operation is executed.

It gives the higher bound of running time for an algorithm.

average - case analysis

The process of identifying ~~a~~ all possible inputs of size n and there probability of execution and counting the no. of times the basic operation is executed in each case.

Linear search

Explanation

Algorithm

Analysis

$C_{best}(n) = 1$ $C_{avg}(n) = \frac{n+1}{2}$

$C_{worst}(n) = n$

2/2/23

Amortized efficiency

In some situation a single operation may be expensive (takes more time) but the total time for an entire sequence for n such operation is always considerably better than the worst case efficiency of that single operation carried out n times.

Amortized efficiency applies not to ~~each~~ a single run alg but to a sequence of operation performed on the same set of data.

The high cost of such ~~as~~ the worst case can be amortized over the entire sequence. Eg: hash table, sets and splay trees.

Mathematical analysis for non recursive algorithm

General plan for analysing the efficiency of non recursive algorithm

1. decide on a parameter (or) more than one parameter indicating input size.

2. Identify the algorithm basic operation (mostly it should be located in the inner most loop)

3. check whether the number of times the basic operation is executed depends only on i/p size n or on other parameters too. in such case calculate

the best case, worst & average case efficiencies for each parameter individually.

4. Set up a sum expressing the no. of times the basic operation is executed.

5. Using standard formula and some manipulation rules, either find a closed form for the count. or establish it Order of growth.

Eg: $\frac{1}{2}n^2$ Eg: $\in \Theta(n^2)$

Sum manipulation rules

1. $\sum_{i=1}^{n} ca_i = c\sum_{i=1}^{n} a_i$

2. $\sum_{i=1}^{n} (a_i \pm b_i) = \sum_{i=1}^{n} a_i \pm \sum_{i=1}^{n} b_i$

Summation Formula

1. $\sum_{i=l}^{u} 1 = u-l+1$ where l & u are some lower ~~[illegible]~~ and upperlimit and $l \leq u$.

Eg: $\sum_{i=5}^{12} 1 = 12-5+1 = 8$ $\sum_{i=6}^{11} 1 = 11-6+1 = 6$

$\sum_{i=0}^{10} 1 = 10-0+1 = 11$

2. $\sum_{i=0}^{n} i = \sum_{i=1}^{n} 1 = 1+2+3+\dots+n$

$= \frac{n(n+1)}{2} = \frac{1}{2}n^2 + \frac{1}{2}n \approx \frac{1}{2}n^2 \in \Theta(n^2)$

3/2/23

Example algorithms

1. Algorithm to find the ~~[illegible]~~ largest element in the list of n numbers.

```
# Alg MaxElement(A[0..n-1])
// determines the maximum element in a list
```

// i/p : An array A[0..n-1] with n real numbers

// o/p : The maximum element of the list.

```
max = A[0]
for i to n-1 do
        if max < A[i]
            max = A[i].
return max
```

Eg

0	1	2	3
3	2	10	7

max = 3

i = 1 Compare max, 2 → max > 2 ? True
i = 2 compare max, $\overset{10}{8}$ → max >10 ? False
i = 3 compare max > 7 ? True
i = 4

Explanation

The algorithm starts with the assumption that the 1st element is the largest one.

Then every Subsequence is compared with the largest element, max.

When a new element is greater than max, the max Value is Updated. Finally at the end of the list max has the largest Value.

Analysis

1. $\overset{\text{decide}}{\text{design}}$ on the parameter

Parameter is n which is nothing but the i/p size

The alg considers all the n elements in the list.

2. Identify the basic operation

The basic operation is comparison of the currently largest element and an element in the array.

3. check if analysis depend on n or any other parameter.

The algorithm compares all the n elements in the list irrespective of their position, value etc., so best case, average case & worst case will be equivalent.

Hence C(n) can be calculated.

4. Set up a summation based on n.

$$C(n) = \sum_{i=1}^{n-1} 1 = (n-1) - \cancel{1} + \cancel{1} = n-1$$

$$\in \Theta(n)$$

$$\boxed{C(n) = n-1 \quad \in \Theta(n)}$$

2. Uniqueness for an array.

The algorithm ~~sa~~ that check uniqueness of an array - whether all the elements in the array or distinct - no duplicate

```
Alg Uniqueness(A[0..n-1])
    // determines Uniqueness of an array
    // i/p : An array A[0..n-1] with n elements
    // o/p : Returns true if the array is unique, and
            returns false if the array has duplicate
            elements.

~~for (i = 0~~ for i = 0 to n-2 do
        for j = i+1 to n-1 do
            if A[i] = A[j]
                return False
    return True
```

4/2/23

Eg

0	1	2	3	4
10	20	30	35	50

n = 5

$i=0 \rightarrow 10 \rightarrow \overset{1}{20}, \overset{2}{30}, \overset{3}{35}, \overset{4}{50}$ Comparison 4

$i=1 \rightarrow 20 \rightarrow \overset{2}{30}, \overset{3}{35}, \overset{4}{50}$ 3

$i=2 \rightarrow 30 \rightarrow \overset{3}{35}, \overset{4}{50}$ 2

$i=3 \rightarrow 35 \rightarrow \overset{4}{50}$ 1

No duplicates = 4+3+2+1

= 10

⇒ max no. of comparisons

10	20	30	10

i = 0 10 → 20, 30, 10

return False.

Explanation

To check the uniqueness of an array the algorithm compares every element in the array with every other element in the array.

If any 2 elements are equals the algorithm stops and return false (the array is not unique).

If no 2 elements are equals, the algorithm executes complitely and finally returns True.

Analysis

1. $\overset{\text{decide}}{\text{design}}$ on an parameter

Parameter → i/p size n because the algorithm depends on

* n and * duplication in n elements.

2. Identify the basic operation

The basic operation is comparison within inner most loop.

3. check if analysis depends on n or any other parameters.

The algorithm execution depends on • n and where the duplication occurs.

so the best case i/p is when the 1st two elements are equals. - it takes one comparison.

Eg:

10	10	30	40

i = 0 Compare 10 with 10

match found

Array is not unique

No. of comparisons = 1

$C_{best}(n) = 1$

Worst Case analysis

The worst case i/p 1. when the last two elements are equals.

2. No two elements are equal.

3. In both the cases the max. no. of comparisons takes place. So $C_{worst}(n)$ has to be calculated.

4. Set up a summation based on n.

$$C(n) = \sum_{i=0}^{n-2}\left(\sum_{j=i+1}^{n-1} 1\right) = \sum_{i=0}^{n-2}\left(n-\cancel{1}-(i+1)+\cancel{1}\right)$$

$$= \sum_{i=0}^{n-2}(n-i-1) = \sum_{i=0}^{n-2}(n-1)-i$$

$$= n-1\sum_{i=0}^{n-2}1 - \sum_{i=1}^{n-2} i$$

$$= n-1 \cdot (n-2-0+1) - \frac{(n-2)(n-2+1)}{2}$$

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

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

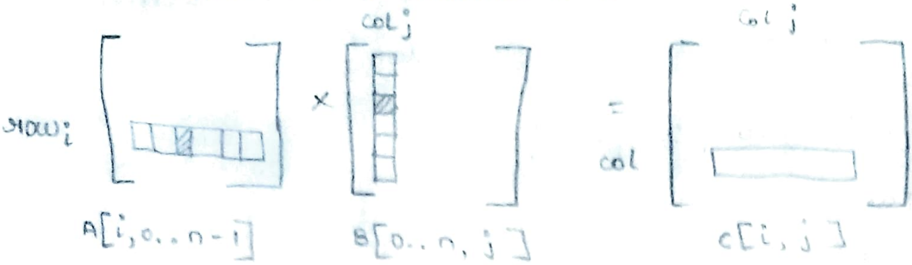
$$= (n-1)\left(\frac{n}{2}\right) = \frac{(n-1)n}{2} = \frac{n(n-1)}{2}$$

$$\approx \frac{1}{2}n^2 \in \Theta(n^2)$$

Eg 3 3. Multiplication of matrices

2 nxn matrix

Product → a scalar matrix with nxn values.



$$C[i,j] = A[i,0] * B[0,j] + A[i,1] * B[1,j] + \dots + A[i,n-1] * B[n-1,j]$$

```
Alg MatrixMultiplication (A[0..n-1, 0..n-1], B[0..n-1, 0..n-1])
// determines the product of 2 nxn matrix
// i/p: A[0..n-1, 0..n-1], B[0..n-1, 0..n-1]
// o/p: product matrix C[0..n-1, 0..n-1]

 for i = 0 to n-1 do
     for j = 0 to n-1 do
         C[i, j] = 0
         for k = 0 to n-1 do
             C[i, j] = C[i, j] + A[i, k] * B[k, j]
 return C
```

7/2/23

Analysis

1. ~~design~~ decide on an parameter

Parameter → i/p size n.

2. Identify the basic operation.

basic operation is multiplication & addition.

→ in the innermost loop.

3. check if analysis depends on n (or) any other parameters

The analysis only depends on the parameter n & hence best, worst & average case efficiency need not be analyse separately.

4. set up a summation.

To find C(n) the number of times ~~a~~ the addition & multiplication ~~do~~ will be executed has to be computed.

*

$$C(n) = \underset{\substack{\text{No. of}\\ \text{addition}}}{\underset{\downarrow}{C_a(n)}} + \underset{\substack{\text{No. of}\\ \text{multiplication.}}}{\underset{\downarrow}{C_m(n)}}$$

$$C_a(n) = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\sum_{k=0}^{n-1} 1 = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} n-\cancel{1}-0+\cancel{1}$$

$$= \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} n = n\sum_{i=0}^{n-1} (n-\cancel{1}-0+\cancel{1})$$

$$= n^2\sum_{i=0}^{n-1} 1 = n^2(n-\cancel{1}-0+\cancel{1}) = n^3$$

$$C_m(n) = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\sum_{k=0}^{n-1} 1 = n^3$$

$$c(n) = n^3 + n^3 = 2n^3 \approx n^3 \in \Theta(n^3)$$

The overall execution time can be calculated considering the time taken for addition and multiplication and no. of addition & multiplication.

$$T(n) = \underset{\substack{\text{Time taken}\\ \text{for 1addition}}}{\underset{\downarrow}{t_a(n)}}\, C_a(n) + \underset{\substack{\text{Time taken for}\\ \text{1 multiplication}}}{\underset{\downarrow}{t_m(n)}}\, C_m(n).$$

Finally, $\boxed{T(n) = t_a\, C_a(n) + t_m\, C_m(n)}$

$$c(n) \in \Theta(n^3)$$

4. Algorithm to find the no. of bits in the binary representation of decimal number.

```
Alg Binary count (n)
// determines the no. of bits in binary representation of a
decimal number.
    // I/P : A decimal number n.
    // O/P : The no. of bits.
    count = 1
    while n>1
        $n = \lfloor \frac{n}{2} \rfloor$
        count = count +1
    return count
```

Eg $n = 9$

$count = 1$

$9 > 1$

$n = \lfloor \frac{9}{2} \rfloor = 4$ count = 2

$4 > 1 \rightarrow n = \lfloor \frac{4}{2} \rfloor = 2$ count 3

$2 > 1 \rightarrow n = \lfloor \frac{2}{2} \rfloor = 1$ Count = 4

$1 > 1 \longrightarrow$ False

Analysis

1. The parameter size is 1 because no. of i/p.s is equal to 1.

So let the parameter be n.

2. The basic operation is

The Comparison in the while loop condition

because the program continuous or stop the execution based on values of n and it is checked in the while condition $(n>1)$.

No. of execution of the basic operation = No. of iterations of the body of the loop +1 .

Since the value of n is halved every time in the iterations, the no. of times the loop is executed is equal to $\lfloor \log_2 n \rfloor$ (based on the basic efficiency classes).

So the total no. of comparisons is equal to

$\log_2 n + 1$

$C(n)$ = no. of executions of loop condition $n>1$

= no. of iterations +1

$$\boxed{C(n) = \lfloor \log_2 n \rfloor + 1}$$

$$\boxed{C(n) \in \Theta(\log n)}$$

8/2/23

Mathematical analysis for recursive algorithm.

General plan for analyzing the time efficiency of recursive algorithm.

1. decide on a parameter ~~indicator~~ indicating the iput size.

2. Identify the algorithms basic operation in the innermost loop.

3. check whether the no. of times the basic operation is executed can differ on different i/ps of the same size. If it can differ the worst, best & average case efficiencies must be investigated separately.

4. set up a recurrence relation and an

appropriate initial condition for computing the no. of times the basic operation is executed.

5. Solve the recurrence relation (or) determine its order of growth.

Recurrence ~~algorithm~~ relation ⓧ

It is an equation which represents a sequence based on ~~same~~ some rules. It expresses the nth term of a sequence as a function of the k preceeding terms where $k < n$.

Eg. $T(n) = T(n-1)+1$ → in factorial calculation,

initial Condition

In a sequence formed using recurrence relation, the 1st few values are needed to identify the beginning of the sequence & they dependent on initial condition.

Eg. $T(n)=0$ when $n = 0$.

↓

In factorial calculation because no multiplication

Example algorithms

1. Factorial of a number.

```
Alg Factorial (n)
// computes n! recursively.
// I/p : A positive integers n.
// o/p : n!
```

```
if n = 0
    return 1
else
    return Factorial (n-1) * n ;
```

Analysis

1. decide on a parameter

The parameter is n - the value of n is used for analysis.

2. Identify the basic operation.

The basic operation is multiplication.

3. The analysis depends on n only so best, worst and avg case efficiencies need not be done separately.

4. Set up a recurrence

The recursive statement $F(n) = F(n-1) * n$ and the initial condition in algorithm is $F(0) = 1$

Thus the no. of multiplications can be calculated using the following recurrence & the initial condition.

$M(n) = M(n-1) + 1$

$M(0) = 0$ (No. of multiplication when n=0 is 0]

9/2/23

5. Recurrence can be solved using method of backward substitution

$M(n) = M(n-1) + 1 \longrightarrow ①$

Sub. $M(n-1) = M(n-2) + 1$ in ①

$M(n) = M(n-2) + 1 + 1 \longrightarrow ②$

Similarly Sub. $M(n-2) = M(n-3) + 1$ in ②

$M(n) = M(n-3) + 1 + 1 + 1$

$$M(n) = M(n-3) + 3. \longrightarrow \textcircled{1}$$

Similarly if we proceed

$$M(n) = M(n-n) + n$$

$$M(n) = M(0) + n$$

Applying the initial condition,

$$M(n) = 0 + n$$

$$M(n) = n \in \Theta(n)$$

Hence n multiplication are done to calculate the factorial of n

As n increases the no. of multiplication increases gradually & reaches very high values as products.

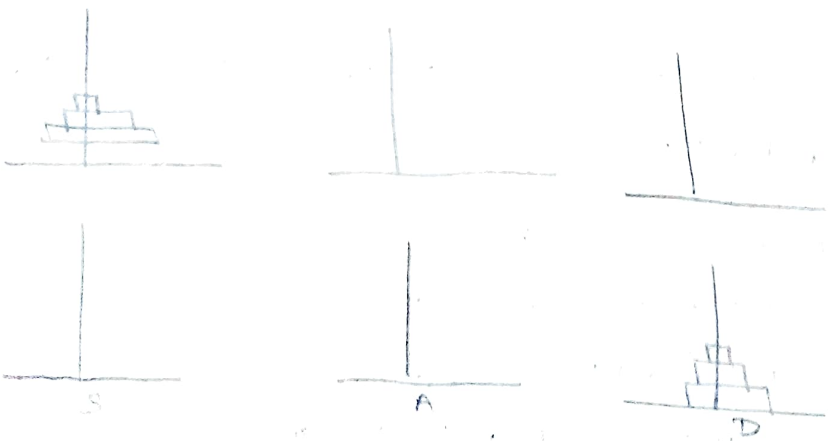
2. Tower of Hanoi puzzle

Problem, goal, solution, rules.

Problem

Given three towers -(source, auxillary & destination) and n no. of disks the aim of the problem is to move all the n disk from the source tower to the destination tower through the auxillary tower.

goal.



rules

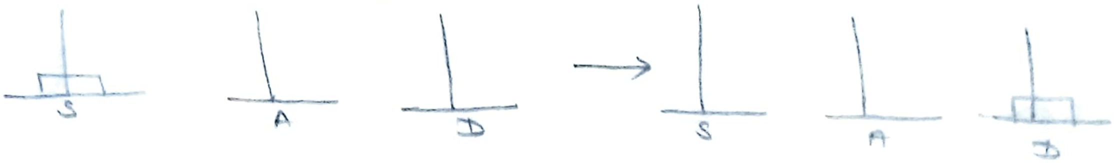
1. Only one disk must be moved at a time.

2. The no. of disk can be one or more than one.

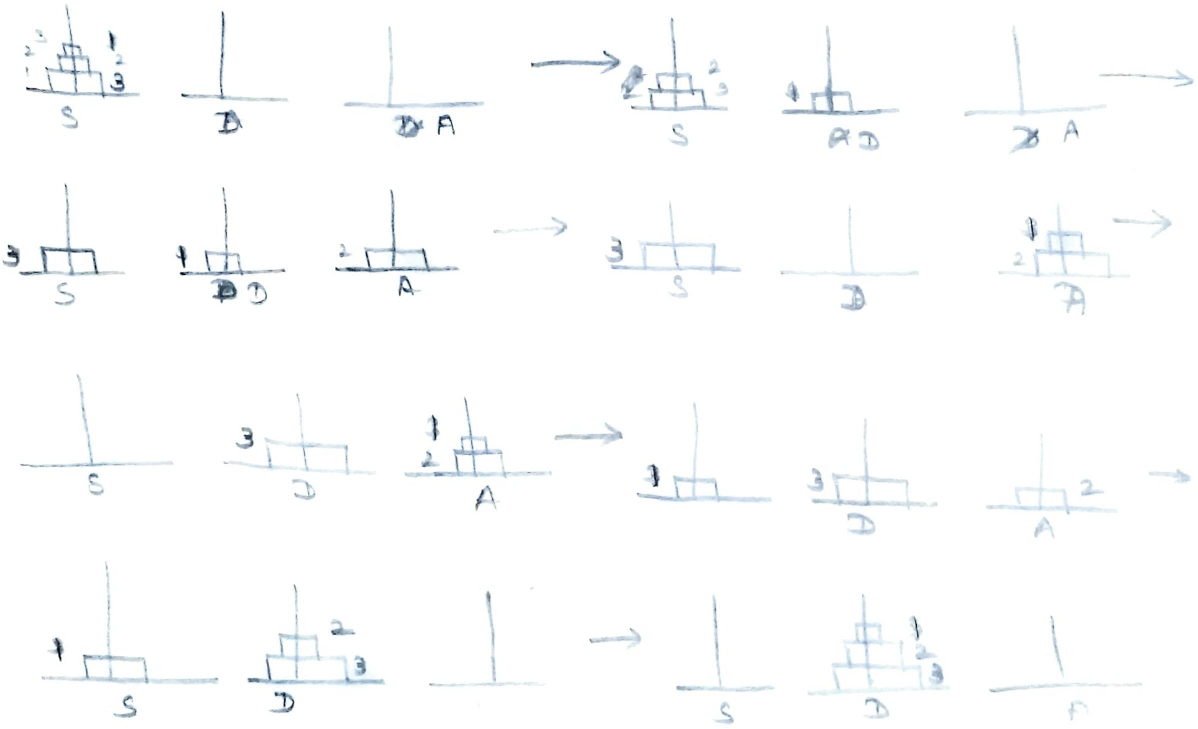
3. The smaller disks must be placed over the larger one.

Solution

when n=1, no. of moves = 1



when n=3



1. The moves 1st disk from source to destination.

2. move the 2nd disk from S to A

3. move 1st disk from D to A.

4. move the 3rd disk from S to D.

5. move the 1st disk from A to S.

6. move the 2nd disk from A to D.

7. move the 1st disk from S to D.

```
if n=1
    move disk 1 from S to D
else
    move n-1 disk from S to A.
    move 1 disk from S to D,
    move n-1 disk from A to D,
```

11/2/23

Analysis

1. parameter is n - the analysis depends on the number of disk to be moved.

2. Basic operation.

The no. of moves to be moved from one tower to another.

3. The analysis depend on only n. So best, worst, average case analysis not needed separately.

4. Set up the recurrence relation.

$M(n) = M(n-1) + 1 + M(n-1)$ for $n > 1$

$M(1) = 1$ ⟶ initial condition.

$$\boxed{\begin{array}{l} M(n) = 2M(n-1) + 1 \\ M(1) = 1 \end{array}}$$

5. solving the recurrence relation - using backward substitution method.

$M(n) = 2M(n-1) + 1 \rightarrow ①$ for $n > 1$

$M(1) = 1$

$M(n-1) = 2M(n-2) + 1 \rightarrow ②$

$M(n-2) = 2M(n-3) + 1 \rightarrow ③$

sub ② in ①

$M(n) = 2(2M(n-2)+1)+1 = 2^2 M(n-2) + 2 + 1 \rightarrow ④$

sub ③ in ④

$M(n) = 2^2(2M(n-3)+1) + 2 + 1 = 2^3 M(n-3) + 2^2 + 2 + 1$

$M(n) = 2^i M(n-i) + 2^{i-1} + 2^{i-2} + \ldots + 2^2 + 2 + 1$

$$\boxed{\sum_{i=0}^{k-1} 2^i = 2^k - 1} \qquad \boxed{M(n) = 2^i M(n-i) + 2^i - 1}$$

using the initial condition

$M(1) = 1$

Let $i = n-1$

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

$$\boxed{M(n) = 2^{n} - 1}$$

$$\boxed{M(n) \in \Theta(2^{n})}$$

The algorithm grows exponentially

The no. of moves increases faster even for smaller values of n.

So exponential algorithm - which may be due to recursion has to be designed & used carefully

when $n=1$, no. of moves $= 2^{n} - 1 = 2^{1} - 1 = 1$

when $n=2$, no. of moves $= 2^{n} - 1 = 2^{2} - 1 = 3$

when $n=3$, no. of moves $= 2^{n} - 1 = 2^{3} - 1 = 7$

when $n=4$, no. of moves $= 2^{n} - 1 = 2^{4} - 1 = 15$

3. Binary digits counting

Algo to count the no. of digits in binary representation of a decimal number.

```
Alg BinCount (n)
//Determines the no. of bits in the binary value of a decimal number.
// I/P : A positive decimal integer (n)
// O/P : The no. of bits
```

Pseudo code

```
if $n = 1$
    return 1
else
    return Bincount $\left\lfloor \frac{n}{2} \right\rfloor + 1$
```

$n = 8 \rightarrow$ Bincount (8)

$8 = 1$ Bincount $\left\lfloor \frac{8}{2} \right\rfloor + 1 \leftarrow 1$

$\frac{8}{2} = 4 \;\alpha\;$ Bincount $\left\lfloor \frac{4}{2} \right\rfloor + 1 + 1$

$2 = 1 \;\alpha\;$ Bincount $\left\lfloor \frac{2}{2} \right\rfloor + 1$ $= 4$

$1 = 1$

$\downarrow$

return 1

3 addition takes place

Analysis

1. parameter $\longrightarrow n$

2. Basic operation is addition. Since add contributes the ~~more~~ max in calculating the count.

3. B, W, A case not needed separately.

4. Recurrence relation

To calculate the no. ~~ao~~ of add used in recursive call

The following recursive call can be used.

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

The initial condition for this algorithm is

when $n = 1 \longrightarrow A(1) = 0$ (no

Since solving $A\left(\left\lfloor \frac{n}{2} \right\rfloor\right)$ is difficult task for all values of n, smoothness rule is applied. so that n is in terms of power of 2.

Let $n = 2^k$

The recurrence relation will be

$$A(2^k) = A\left\lfloor \frac{2^k}{2} \right\rfloor + 1$$

$$A(2^k) = A(2^{k-1}) + 1$$

$$A(2^0) = 0$$

Now backward substitution method can be applied

$$A(2^k) = A(2^{k-1}) + 1$$

$$A(2^{k-1}) = A(2^{k-2}) + 1$$

$$A(2^{k-2}) = A(2^{k-3}) + 1$$

so $A(2^k) = A(2^{k-1}) + 1$

$$= A(2^{k-2}) + 1 + 1 = A(2^{k-2}) + 2$$

$$= [A(2^{k-3}) + 1] + 2 = A(2^{k-3}) + 3$$

$$\vdots$$

$$= A(2^{k-k}) + k$$

$$= A(2^0) + k$$

$$= 0 + k$$

$$= k$$

$$A(2^k) = k$$

we sub $n = 2^k$

$$\Rightarrow k = \log_2 n$$

$$\boxed{A(n) = \log_2 n \in \Theta(\log n)}$$

17/2/23
Application of Fibonacci Series

In Cryptographic algorithm.

4 ways of finding the n^{th} fibonacci number.

1. Iterative algorithm 2. recursive algorithm

3. using mathematical formula 4. using matrix exponentiation

1. Iterative algorithm

```
Alg Fib(n)
// Generates the nth number in the fib. Series using iterative algorithm
// i/p : n
// o/p : The nth value ,

F[0] = 0
F[1] = 1
for i = 2 to n
    F[i] = F[i-1] + F[i-2]
return F[i].
```

The basic operation in this algorithm is addition - one addition per iteration. So total no. of addition = $C(n) = n-1$ which $\in \Theta(n)$

18/2/23
drawback

In this method, instead of storing ~~the value~~ 0th to n-1 th value in array, which will unnecessarily occupy the space, the previous 2 value alone can be stored and used in calculation.

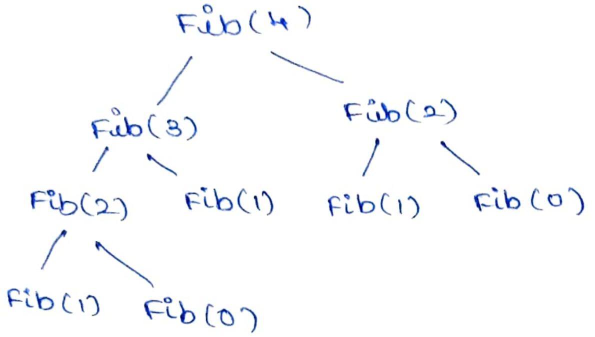
Eg. To find F[8] * it is enough to store F[6] and F[7] (F[0] to F[5] need not be stored)

2. Recursive algorithm

```
Alg Fib(n)
// compute nth Fib. no recursively
// i/p : An non negative integer n
// o/p : nth Fib. number
```

```
if n ≤ 1
    return n
else
    return Fib(n-1) + Fib(n-2)
```

A inefficiency algorithm many values are repeated calculated, as they are not stored earlier. This can be seen using following recursive tree for n = 4



Analysis

The recurrence to compute fib. no's is $F(n) = F(n-1) + F(n-2)$ with initial conditions $F(0) = 0$, $F(1) = 1$

Basic operation is addition.

To calculate the no. of addition the recurrence relation is $A(n) = A(n-1) + A(n-2) + 1 \Rightarrow$

$$A(n) - A(n-1) - A(n-2) = 1 \longrightarrow ①$$

Initial conditions, $A(0) = 0$, $A(1) = 0$.

Since, to solve eqn ① which is in inhomogeneous form (because RHS = 1), it has to be converted into a homogeneous eqn & can be solved using computed values.

$$A(n) - A(n-1) - A(n-2) - 1 = 0$$

Add & subtract 1

$$A(n) - A(n-1) - A(n-2) - 1 + 1 - 1 = 0$$

$$A(n) + 1 - A(n-1) - 1 - A(n-2) - 1 = 0$$

$$[A(n) + 1] - [A(n-1) + 1] - [A(n-2) + 1] = 0 \longrightarrow ②$$

Let $B(n) = A(n) + 1$

② $B(n) - B(n-1) - B(n-2) = 0 \longrightarrow ③$ $B(0) = 1$; $B(1) = 1$

F( )	0	1	2	3	4	5	6
	0	1	1	2	3	5	8
B( )		0	1	2	3	4	5

$A(n) = B(n) - 1 = F(n+1) - 1 \longrightarrow$ (4)

According to Binet's formula which was obtained by solving $F(n) - F(n-1) - F(n-2) = 0$

$\boxed{F(n) = \frac{1}{\sqrt{5}} (\phi^n - \hat{\phi}^n)}$ where $\phi = \frac{1+\sqrt{5}}{2} \approx 1.61803$

and $\hat{\phi} = \phi^{-1} = \frac{1-\sqrt{5}}{2} \approx 0.61803$.

$\hat{\phi}^n$ is negligible compared to ϕ^n as n increases so $\hat{\phi}^n$ can be ignored

$\boxed{F(n) = \frac{1}{\sqrt{5}} \phi^n} \longrightarrow$ (5)

Substitute binet's formula (5) in (4)

$A(n) = B(n) - 1 = \frac{1}{\sqrt{5}} \phi^{n+1} - 1$

$\boxed{A(n) \in \Theta(\phi^n)}$

If the no. of bits in n is considered as parameter to find the no. of addition then

$b = \log_2 n + 1$

$n = 2^b$

Hence $\boxed{A(b) = \Theta(\phi^{2^b})}$

3. using mathematical formula

we can use the direct formula to compute the nth fibnoacci, number

$F(n) = \frac{1}{\sqrt{5}} (\phi^n)$

5^n needs $n-1$ multiplication. so $A(n) = n-1$

$A(n) \in \Theta(n)$

During multiplication the intermediate values are irrational, so accurate approximation is needed. So that the round off will healed a correct result.

4. Using matrix ~~multip~~) exponentiation

The value of F(n) can be obtained from the following matrix representation,

$$\begin{bmatrix} F(n-1) & F(n) \\ F(n) & F(n+1) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^{n} \quad \text{for } n \geq 1$$

Let n=3 for 3 the matrix

$$\begin{bmatrix} F(2) & F(3) \\ F(3) & F(4) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^{3}$$

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 0+1 & 0+1 \\ 0+1 & 1+1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 0+1 & 1+1 \\ 0+2 & 1+2 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix}$$

$$A[n] = \Theta(\log n)$$