CS 203-Advanced Data Structures and Algorithms

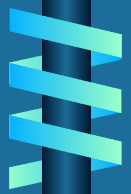
Chapter 1 : Introduction to algorithms and complexity

Chapter 2: Search algorithms

Chapter 3: Sorting Algorithms

Chapter 4: Graph Algorithms

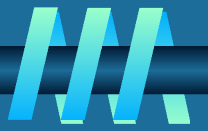
Chapter 5: Coping with NP-Completeness



Chapter 1- Introduction to Algorithms and Complexity

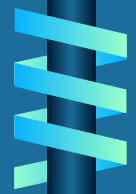
- Introduction to algorithm and efficiency
- Introduction to Data Structures
- Psudo code of an algorithm
- Time Complexity of an algorithm
- Best, Worst and Average Case times
- Time Complexity of recursive algorithms

Introduction

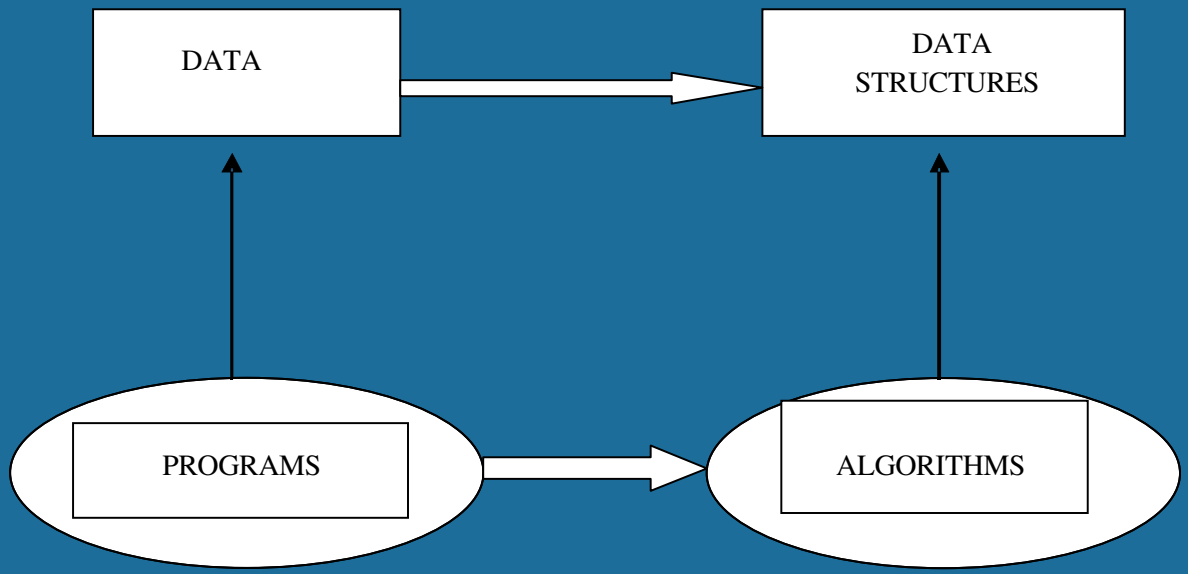


Data structures **involve study of efficiently storing data so that basic operations on the data can be performed efficiently.**

Algorithms **is a study of how to efficiently manipulate the data stored in data structures to solve a particular problem.**



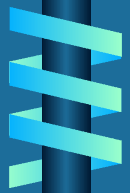
Data Structures and Algorithms



Data and Programs are machine dependant.
Data Structures and Algorithms are machine independent.
Data structures become data when implemented.
Algorithms become programs when implemented.

Some Well-known Computational Problems

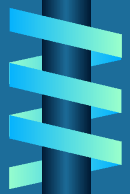
- **Sorting**
- **Searching**
- **Shortest paths in a graph**
- **Minimum spanning tree**
- **Primality testing**
- **Traveling salesman problem**
- **Knapsack problem**
- **Chess**
- **Towers of Hanoi**
- **Program termination**



What is an algorithm?

- **Recipe, process, method, technique, procedure, routine,… with following requirements:**

2. **Finiteness**
   - **terminates after a finite number of steps**
3. **Definiteness**
   - **rigorously and unambiguously specified**
4. **Input**
   - **valid inputs are clearly specified**
5. **Output**
   - **can be proved to produce the correct output given a valid input**
6. **Effectiveness**
   - **steps are sufficiently simple and basic**



Euclid's Algorithm

Problem: Find gcd(m,n), the greatest common divisor of two nonnegative, not both zero integers m and n

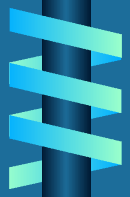
Examples: gcd(60,24) = 12, gcd(60,0) = 60, gcd(0,0) = ?

Euclid's algorithm is based on repeated application of equality

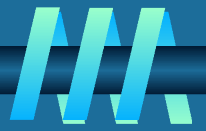
$$\gcd(m,n) = \gcd(n, m \bmod n)$$

until the second number becomes 0, which makes the problem trivial.

Example: gcd(60,24) = gcd(24,12) = gcd(12,0) = 12



Two descriptions of Euclid's algorithm



Step 1 **If $n = 0$, return m and stop; otherwise go to Step 2**

Step 2 **Divide m by n and assign the value fo the remainder to r**

Step 3 **Assign the value of n to m and the value of r to n. Go to Step 1.**

while $n \neq 0$ **do**

$r := m \bmod n$

$m := n$

$n := r$

endwhile

return m

Other methods for computing gcd(*m*,*n*)

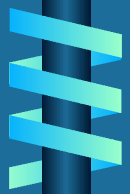
Consecutive integer checking algorithm

Step 1 Assign the value of min{*m*,*n*} to *t*

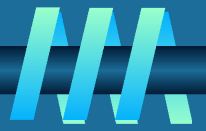
Step 2 Divide *m* by *t*. If the remainder is 0, go to Step 3; otherwise, go to Step 4

Step 3 Divide *n* by *t*. If the remainder is 0, return *t* and stop; otherwise, go to Step 4

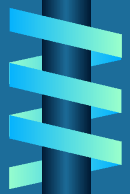
Step 4 Decrease *t* by 1 and go to Step 2



Algorithm design strategies



- **Brute force**
- **Divide and conquer**
- **Decrease and conquer**
- **Transform and conquer**
- **Greedy approach**
- **Dynamic programming**
- **Backtracking and Branch and bound**
- **Space and time tradeoffs**



Fundamental data structures

- list
  - array
  - linked list
  - string
- stack
- queue
- priority queue
- graph
- tree
- set and dictionary

Syntax of Pseudo Code

- **We will use Pascal like syntax for algorithms**

Declaration and Assignment:

```
var <variable>:<type>;
<variable> := <value>
```

While loop:

```
while( <condition>) do
        <bodyOfWhile>
endwhile
```

For loop:

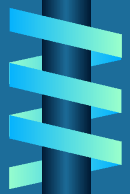
```
for <variable>:=<startValue> to <StopValue> do
      <bodyOfFor>
endfor
```

If Statement

```
if <condition> then
    <bodyOfIf>
endif
```

If-else statement

```
if <condition> then
     <bodyOfIf>
else
     <bodyOfElse>
endif
```



Algorithmic Efficiency

Efficiency **of an algorithm can be looked from the following angles:**

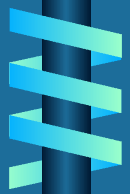
How long does the algorithm take to produce output (time complexity)

How much resources does it use (memory, processors etc) (memory complexity)

How easy is to implement.

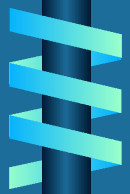
Does there exist a better algorithm?

- **Lower bounds**
- **Optimality**



Steps in Designing Algorithms

1. Understanding the problem
2. Ascertaining the Capabilities of Computational Device
3. Choosing between exact and Approximate Problem Solving
4. Deciding on Appropriate Data Structures
5. Methods of Specifying an Algorithm (Pseudo code)
6. Proving Correctness
7. Time complexity Analysis
8. Coding the algorithm



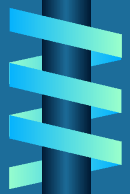
Time for execution of algorithms

Declaration takes no time.

Assignment and comparison take 1 unit of time.

Branching takes as much time as needed for the longer branch plus time for the comparison.

Looping takes as much time as (# of times body of the loop executes) * (time for the body of the loop + 1 time unit for comparison).

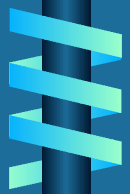


Examples

```
1. for i := 1 to n do
     A[i] := 5; ----- 1 time unit
   endfor

Time for the algorithm = n*(1+1) = 2n


2. for i := 1 to n do
      A[i] := 5;
   endfor
   for j := 1 to n-1 do
       A[j] = 7;
   endfor
Time for the algorithm = n*(1+1) + (n-1) * (1+1)=
   4n-2
```

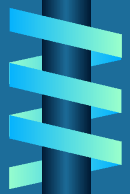


Examples

3. Find if there are any repeated elements in the array A.

```
for i := 1 to n do
   for j := 1 to n do
       if (A[i] = A[j] and i <> j) then
         found := true;
       endif
   endfor
endfor
```

Time Taken = $n * [(n*(2+1)) + 1] = 3n^2 + n$



Examples

4. Find if the key is in the array A.

```
var i: integer;
var found : boolean;
i:=1;
found := false;
while (i <= n and !found)  do
 if (A[i] = key) then
     found := true;
 else
      i := i+1;
 endif
endwhile
```

Time for the algorithm = Cannot be predetermined since the number of times the loop executes depends on the data.

Time Complexity

In terms of growth, the actual time expression does not matter. Only the rate of growth matters.

To get the rate of growth, drop all terms except the term that has the highest degree in n. Also drop the coefficient of the highest degree term.

For Example 1, the time complexity was

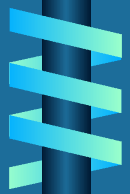
`2n, but the rate or growth is n`

For Example 2, the time complexity was

`4n-2, but the rate or growth is n`

For Example 3, the time complexity was

$3n^2+n$ but the rate of growth is n^2



Examples

Additional Examples:

Actual Time	Time Complexity
$2n + 5$	n
$3n^2 + 9n + 1$	n^2
$n\log n + \log n + 3n$	$n\log n$
$n^2 + \log n + n^3$	n^3
$2^n + n$	2^n
$n\log m + m\log n$	$n\log m + m\log n$

Order of Time Complexities

Increasing order of time complexities:

1 } constant time

logn } logarithimic

} **sublinear**

$n^{.7}$

n } linear

nlogn

$nlog^2n$

n^2 } quadratic

n^2logn

n^3 } cubic

.

.

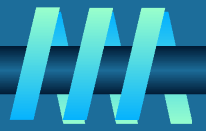
2^n } exponential

n! } factorial

Time Complexity Growth

Time	33n	46nlogn	$13n^2$	$3.4n^3$	2^n
n=10	.00033 sec	.0015sec	.0013sec	.0034sec	.001 sec
n=100	.003sec	.03sec	.13sec	3.4sec	$2*10^{24}$ hours
n=1000	.033sec	.45sec	13sec	.94hours	10^{282} years
n=10,000	.33sec	6.1sec	22min	39days	10^{3000} years
n=100,000	3.3sec	1.3sec	1.5days	108 years	$4*10^4$ centuries
Max size for 1min	1,800,000	82,000	2200	260	26

Best Average and Worst Case Time



Sometimes it is not possible to find the exact time complexity since it may depend on actual data. We saw an instance of this in Example 4.

Therefore, We define three types of time complexities:

***Best Case Ω( )* is the time taken by shortest execution of the algorithm.**

***Average Case Θ( )* is the time taken by algorithm on an average.**

***Worst case O( )* is the time taken by the longest execution of the algorithm.**

Best Worst and Average case time

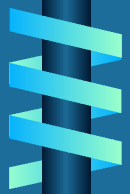
Find if the key is in the array A.

```
var i: integer;
var found : boolean;
i:=1;
found := false;
while (i <= n and !found)  do
 if (A[i] = key) then
     found := true;
 else
  i := i+1;
 endif
endwhile
```

Best Case = Ω(1)

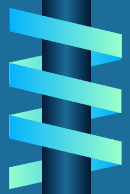
Worst case = O(n)

Average Case Θ(???)

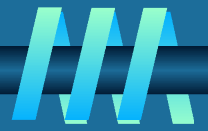


Best, Worst and Average case time

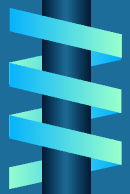
- **Sometimes the three cases may be the same!!**
- **Consider the problem of finding the largest element in an array.**
- **We do this by first assuming that the first element is the largest and then having an index i that moves from 2 to n, each time changing the largest if needed.**



Best, Worst, and Average Times



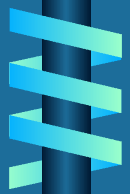
5	4	2	7	9	7	largest= 5
	↓ i=2					
5	4	2	7	9	7	largest =5
		↓ i=3				
5	4	2	7	9	7	largest =5
			↓ i=4			
5	4	2	7	9	7	largest= 7
				↓ i= 5		
5	4	2	7	9	7	largest= 9
					i =6↓	
5	4	2	7	9	7	largest =9



Best, Worst, and Average Times

```
var i: integer;
var largest : integer;
largest := A[1];
i :=2;
while (i <=n) do
   if (A[i] > largest) then
     largest := A[i];
   endif
  i:=i+1;
endwhile
```

$$T(n) = \Sigma_{1<i\leq n} 3 = 3n$$



Useful summation formulas and rules

$\Sigma_{l \le i \le u} 1 = 1+1+\ldots+1 = u - l + 1$

In particular, $\Sigma_{1 \le i \le u} 1 = n - 1 + 1 = n \in \Theta(n)$

$\Sigma_{1 \le i \le n} i = 1+2+\ldots+n = n(n+1)/2 \approx n^2/2 \in \Theta(n^2)$

$\Sigma_{1 \le i \le n} i^2 = 1^2+2^2+\ldots+n^2 = n(n+1)(2n+1)/6 \approx n^3/3 \in \Theta(n^3)$

$\Sigma_{0 \le i \le n} a^i = 1 + a + \ldots + a^n = (a^{n+1} - 1)/(a - 1)$ for any $a \neq 1$

In particular, $\Sigma_{0 \le i \le n} 2^i = 2^0 + 2^1 + \ldots + 2^n = 2^{n+1} - 1 \in \Theta(2^n)$

$\Sigma(a_i \pm b_i) = \Sigma a_i \pm \Sigma b_i \quad \Sigma c a_i = c \Sigma a_i \quad \Sigma_{l \le i \le u} a_i = \Sigma_{l \le i \le m} a_i + \Sigma_{m+1 \le i \le u} a_i$

L'Hôpital's rule and Stirling's formula

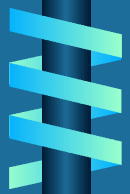
L'Hôpital's rule: If $\lim_{n\to\infty} f(n) = \lim_{n\to\infty} g(n) = \infty$ and the derivatives f', g' exist, then

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{f'(n)}{g'(n)}$$

Stirling's formula: $n! \approx (2\pi n)^{1/2} (n/e)^n$

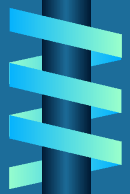
Example: $\log n$ vs. n

Example: 2^n vs. $n!$



Time Analysis of non-recursive code

- **Write an algorithm for finding if an array of size n has a unique (non-repeated) element. Find the time complexity of the algorithm using sigma notation.**



Example : Counting binary digits

```
procedure binary(n):integer
 var count;
 begin
      count := 1;
      while (n> 1) do
         count := count + 1;
         n :=      ⌊n/2⌋
      endwhile
     binary:=count;
end
```

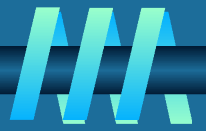
The floor of n/2 is used.

The time cannot be investigated the way the previous examples

Analysis of Recursive Algorithms

- **Decide on a parameter indicating an input's size.**
- **Set up a recurrence relation with an appropriate initial condition expressing the number of times the basic op. is executed.**
- **Solve the recurrence (or, at the very least, establish its solution's order of growth) by backward substitutions or another method.**

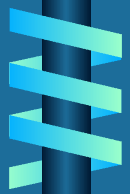
Example 3: Counting #bits



```
procedure binRec(n):integer
   if (n = 1)
         binRec := 1; stop
   else
        binRec := 1+ binRec([n/2])
   endif
end
```

$T(n) = 1 + T(n/2)$

$T(1) = 1;$

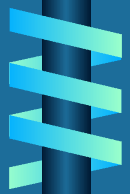


Recursive evaluation of *n*!

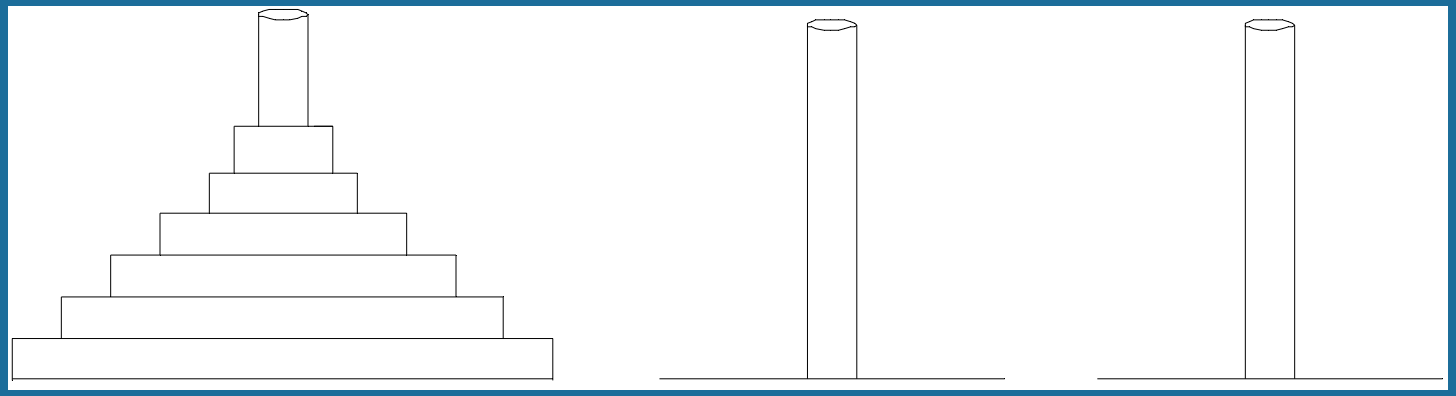
Definition: $n! = 1 * 2 * \ldots * (n-1) * n$ for $n \geq 1$ and $0! = 1$

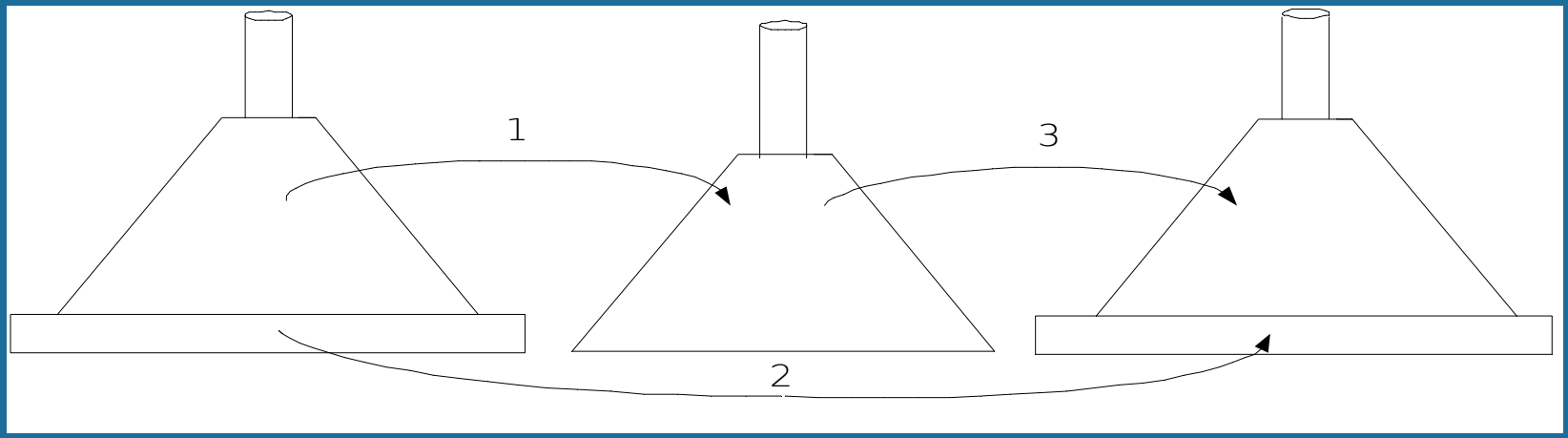
Recursive definition of *n*!: $F(n) = F(n-1) * n$ for $n \geq 1$ and $F(0) = 1$

Write the algorithm and perform an analysis for the computing the factorial of n.



The Tower of Hanoi Puzzle

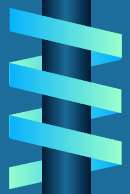




Recurrence for number of moves:

Solving recurrence for number of moves

$$M(n) = 2M(n-1) + 1, \quad M(1) = 1$$



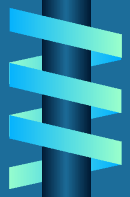
Exercises

1. Design an algorithm for computing $\lfloor \sqrt{(n)} \rfloor$. for any positive integer n. Besides assignment and comparison you can inly use four basic arithmetic operations.

2. Write an algorithm for each of the tasks below and find the worst case, best case and average case times for the following algorithms.
   a. Find if an array has repeated elements.
   b. Find the smallest element in an array.

3. Write an algorithm to find the second largest element in an array. Exactly how many comparisons are done in the worst case?

4. Write an algorithm to find the largest and smallest element in an array. In general this should take 2n comparisons if you wrote two independent algorithm. Try to write an algorithm that does this in 1.5n comparisons.



Exercises

5. List the following functions from lowest to highest order.

n 2^n $n\lg n$ $\ln n$ $n-n^3+7n^5$ $\lg n$ $n^{.5}$ e^n

$n^2+\lg n$ n^2 2^{n-1} $\lg\lg n^3$ $(\lg n)^2$ $n!$

n^{1+e} where $0 < e < 1$.

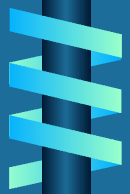
6. Describe a standard algorithm for finding the binary representation of a positive decimal integer

a) In English

b) In Pseudocode

7. Find gcd (31415, 14142) by applying Euclid's algorithm.

8. Prove that equality gcd(m,n) = gcd (n, m mod n) for every pair of positive integers m and n.



Exercises

9. Consider the following algorithm for finding the minimum distance between two closest elements in an array of numbers

```
procedure minDistance(A[1 .. n-1])
begin
    dmin := infinity;
    for i := 1 to n  do
        for j := 1 to n do
            if (i <> j) and |(A[i] – A[j])| < dmin
                dmin :=   |(A[i] – A[j])|
            endif
        endfor
    endfor
end
```

Make as many improvements as you can to the above algorithm.

Exercises

10. There are 22 pairs of gloves in a drawer. 5 pairs are red, 4 pairs are yellow and 2 pairs are green. You select the gloves in dark and can check them only after selection is made. What is the smallest number of gloves you need to select in order to have one pair of matching gloves, in the best case? in the worst case?

***11. You are facing a wall that stretches infinitely in both directions. There is a door in the wall, but you neither know how far away it is or which direction it is. You can see the door only when you get there. Design an algorithm that reaches you to the door by walking at most O(n) steps where n is the number of steps between you and the door.**

Exercises

12. Compute the following sums

a) Σ 1 , for i = 3 to i = n+1

b) Σ i(i+1) for i = 0 to i = n-1

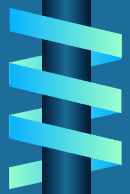
c) $\Sigma \Sigma$ i , j for i = 1 to n (outer) and j = 1 to n (inner)

d) Σ 1/(i (i+1)) for i = 1 to n

13. Find the order of growth for the following:

Σ $(i^2 + 1)^2$ for i = 0 to n-1

Σ $(i+1)\ 2^{i-1}$ for i = 1 to n



Exercises

14. Solve the following recurrence relations

x(n) = 3x(n-1) , n > 1 and x(1) = 4

x(n) = x(n-1) + n, n > 0, x(0) = 0

x(n) = x(n/2) + n , n > 1 x(1) = 1 [solve when n = 2^k]

x(n) = x(n/3) + 1, n > 1, x(1) = 1 [solve when n = 3^k]

15. Write a recursive algorithm for finding the nth fibonacci number and find the time complexity for finding the nth fibonacci.

16. Consider the following formula for finding the sum of first n cubes

S(n) = S(n-1) + n*n*n; , n >1

S(1) = 1

Write the recursive algorithm and find the time complexity.