16-05-2023

CSC 213

Introduction To Algorithm

- Program = algorithm + Data structure

- Solutions to many problems in life such as finding the shorter path, sorting etc may involve the use of many algorithms. We all use algorithms in our daily life such as preparing for an exam, preparing a meal. To know which of the algorithms use better, you have to analyse the algorithm based on some factors such as time complexity and space complexity.

- Algorithm can be defined as the set of steps to accomplish a task. The formal definition of algorithm is as follows:- It is a finite sequence of steps or instructions to solve a problem (or to solve a computational probleml.

To design a better program, algorithms are required. Algorithms are written after which programs are then written

Difference between algorithm and Program

Algorithm	Program
1) Required at the design phase.	1) Required at the implementation phase.
2) The person writing the algorithm should have domain knowledge.	2) The person should be a Programmer.
3) Written in natural language such as English. Language	3) Written in any programming Language.
4) We analyse algorithm	4) We test a program

Rules for constructing an Algorithm

a) Input

b) Output

c) Definiteness

d) Finiteness

e) Effectiveness

f) Comment session

Ways to represent an algorithm

1) Natural Language

2) Actual Programming language

3) Flowchart

4) Pseudocode

28-05-2023

CSC 213

Introduction To Algorithm

- Time complexity
- Space complexity

computational complexity.

- The Turing machines of Turing.

* Theorem 1.1

* Sort:- Selection sort, Exchange sort, Insertion sort

Assignment

i) Write a program to store a set of 25 numbers in an array, arrange and print the numbers in ascending order using selection sort, Exchange sort and insertion sort.

The Big "O" Notation

The two differentiable functions in this type of asymptotic nations are $f(n)$ and $g(n)$. $f(n)$ grows with the same rate as lower than $g(n)$.

The following are the conditions the big O notation:

$$f(n) \leq c \cdot g(n), \quad n \geq n_0$$

$$c > 0,$$

$$n_0 \geq 1$$

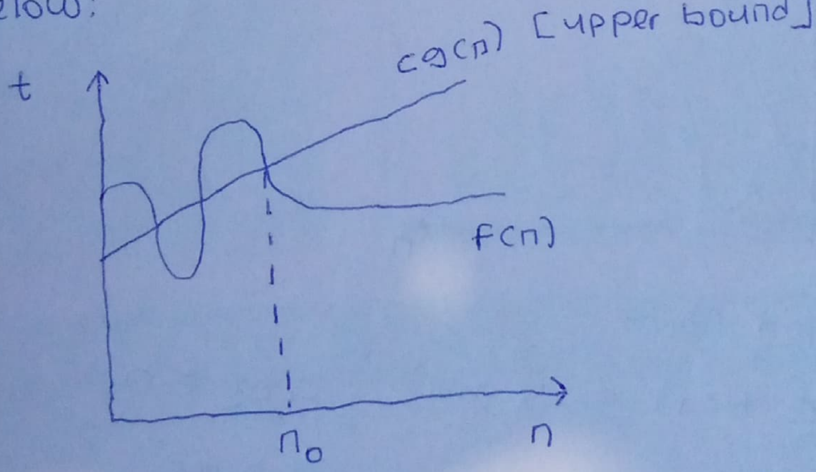
$$f(n) = O g(n)$$

This is logically broken down as follows;

$$f(n) = \theta(g(n)) \quad \text{or}$$

$$f(n) = o(g(n)) \rightarrow \text{"slower"}$$

The graphical graphical representation of big O notation is as shown below:



$g(n)$ is an asymptotic upper bound for $f(n)$.

For example: $f(n) = 3n + 2$; $g(n) = n$.

The Formula is $f(n) = O g(n)$

where $f(n) \leq c g(n)$; $c > 0$

$n_0 \geq 1$

$$3n + 2 \leq cn$$

Assuming $c = 4$

$3n + 2 \leq 4n$

$2 \leq 4n - 3n$

$2 \leq n$

$n \geq 2$

The Theta Notation

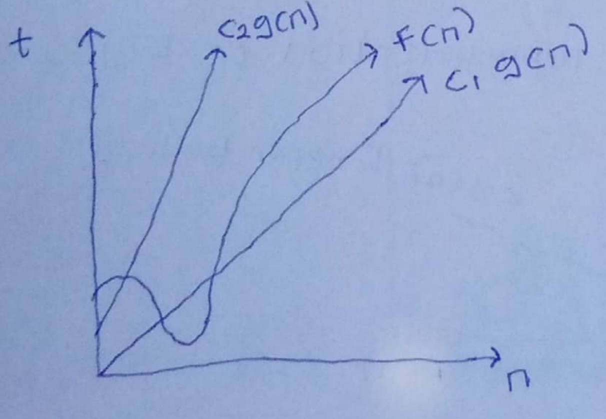
Again we choose $f(n)$ and $g(n)$ as two differentiable functions and say that they have the same growth rate if:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = c, \quad 0 < c < \infty$$

formally stated as ::

$$f(n) = \theta(g(n))$$

Graphically representation:



θ notation here:

$c_1 g(n) \leq f(n) \leq c_2 g(n)$; $c_1, c_2 > 0$

$n \geq n_0$

$n_0 \geq 1$ atleast 1 should be there

For example: $f(n) = 3n + 2$, $g(n) = n$

For mula for θ notation: $c_1 g(n) \leq f(n) \leq c_2 g(n)$

$f(n) \geq c_1 g(n)$, $f(n) \leq c_2 g(n)$

$3n + 2 \geq 4n$

Assume $c_1 = 1$, $c_2 = 4$

$n \geq 2$

$3n + 2 \geq 1n$

$2 \geq n - 3n$

$n \geq -1$

06-06-2023

CSC 213

(Ω)

The Big Omega Notation

The two differentiable functions are f(n) and g(n) where f(n) grows with the same rate or faster than g(n).

It is represented as:: $f(n) \geq c\,g(n);\ n \geq n_0$

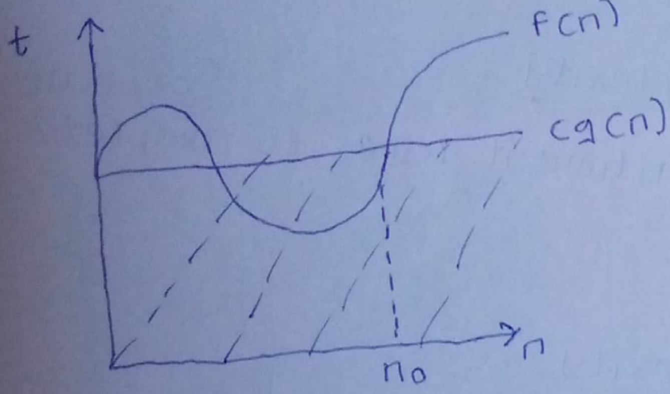
$c > 0,$

$n_0 \geq 1$

Ω notation is denoted by:

$F(n) = \Omega\, g(n)$

Graphical representation::



This is completely lower bounded.

G(n) is an asymptotic lower bound for F(n).

For example: $F(n) = 3n + 2,\ g(n) = n.$

where $f(n) \geq (g(n))$

$\therefore\ 3n + 2 \geq cn$, assume $c = 1$

$3n + 2 \geq n,\ n_0 \geq 1$

$3n + 2 = \Omega(n)$

This is the formula for calculating the time complexity of the big omega notation:- $f(n) = \Omega\, g(n)$.

Performance Analysis of an Algorithm

This can be measured in terms of:

1) Space complexity.
2) Time complexity.

An algorithm is said to be efficient and fast if it takes less time to execute and consume less memory space.

Space Complexity

This refers to the amount of memory space required by an algorithm during course of execution

Time complexity

"execute"

This refers to how much time it takes to complete a program.

Space Complexity

The algorithm generally require space for::

1) Instruction space
2) Data space
3) Environment space.

1) Instruction space depends on ~~how~~ the number of lines taken to execute the program.
2) Data space refers to all the space required to store the constant and variable values.
3) Environment space refers to the space required to store the environment information needed to resume the suspended functions.

13-06-2023 CSC 213

Space Complexity

The space complexity can be calculated in two ways:- based on the program. The program may be constant program or linear program. The two ways are:-

a) Constant space complexity

b) Linear space complexity.

a) Constant space complexity :- This means that a fixed space will be there. e.g.

```
int square (int a) {
    return a * a
}
```

We are using 1 variable, 1 integer value and only 1 input. Here algorithm required fixed amount of space for all input value. So this space complexity is constant.

- Why constant?
- This is because we are using fixed amount of space for all input values.

b) Linear space complexity :- Here, the space is varying. Space needed for algorithm is based on:-

- Size of variable "n" = 1 word.
- Array a values = n word
- Loop variable i = 1 word
- Sum variable s = 1 word

one variable

If you have 2 variables, each takes 1 word.

```
int sum (int A[], int n) {  - - - - - - - - =0        Linear space complexity
    int sum = 0, i;          - - - - - - -   1 word
    for (i = 0, i < n; i++)
        sum = sum + A[i];    - - - - - - -   1 word
        return sum;          - - - - - - -   1 word
}                            * variable n is taking 1 word.
```

Linear space complexity = 1 + 1 + 1 + n
= (n + 3) words

For any algorithm, memory is required for the following purposes:-

1) To store program instructions.
2) To store constant values.
3) To store variable values.
4) For few other things like function call, jumping statement.

* Auxiliary space :- This is the temporary space (excluding the input size) allocated by the algorithm to solve a problem with respect to input size.

- Space complexity includes both auxiliary space and the space used by the input.

i.e. Space complexity = input size + auxilliary space.

Space complexity

Example:

Algorithm: Addition of 2 numbers.

```
function add (n1, n2) {
    sum = n1 + n2
    return sum
}
```

$n_1 \rightarrow 4$ bytes, $n_2 \rightarrow 4$ bytes

sum $\rightarrow 4$ bytes

Auxilliary space = 4 bytes

Total = 16 bytes (constant)

Time Complexity

This refers to the total amount of time required by an algorithm to complete its execution. we have:-

1) Constant Time Complexity.
2) Linear Time Complexity

1) **Constant Time Complexity**:- If a program requires a fixed amount of time for all input values.

Examples:

```
int sum (int a, int b) {
    return a + b;
}
```

2) **Linear Time Complexity**:- If input values are increasing the time complexity will change.

e.g.

assignment statement	1 step.
Loop conditions for "n" times.	n + 1
Body of loop	n steps

```
int sum (int A[], int n) {
    int sum = 0
    for (i = 0, i<n; i++)
        sum = sum + A[i]
return sum
```

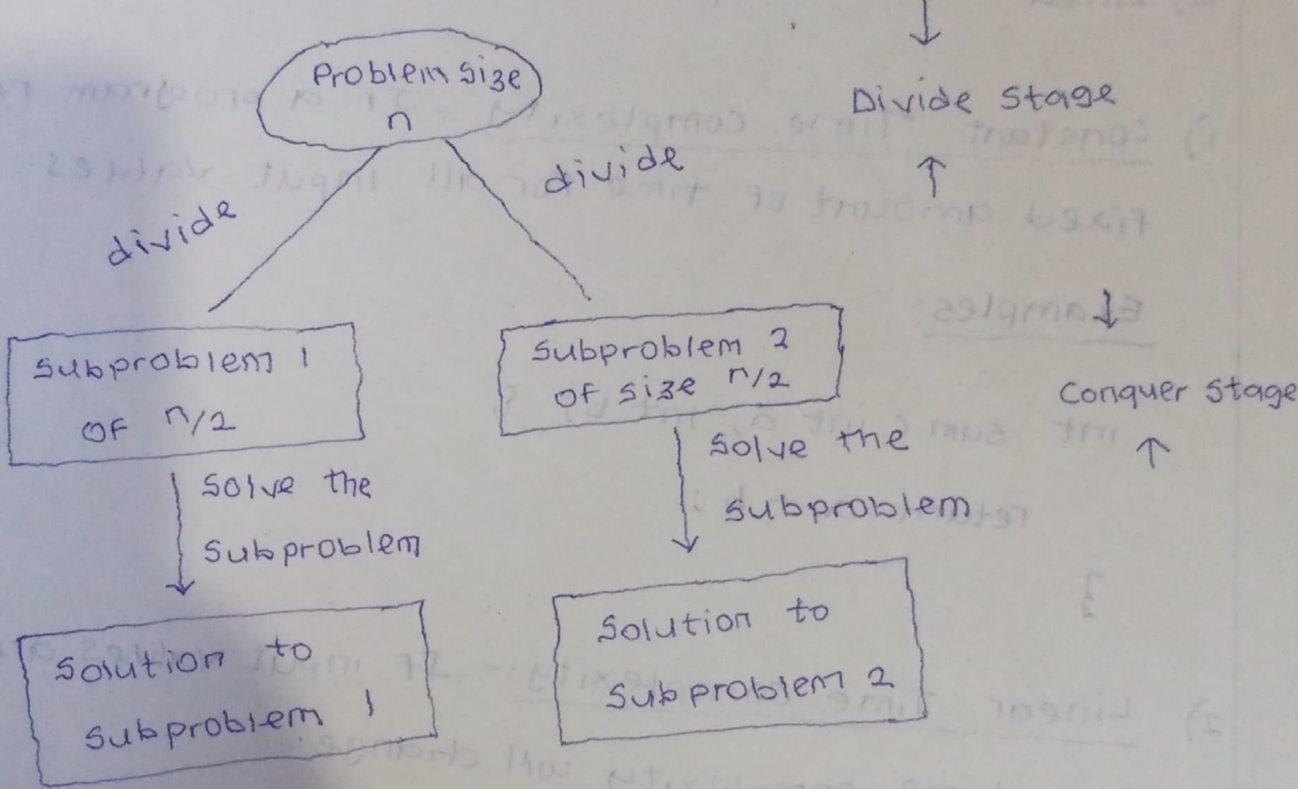
Divide and conquer

Step 1 :- Divide problems into smaller parts.

Step 2 :- Independently solve the parts.

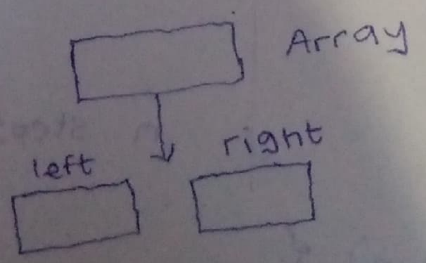
Step 3 :- Combine these solutions to get the overall solution.

This can be explained diagrammatically below:-



Ideas To solve Divide And conquer Problem

1) Divide the array into two halves and recursive solve left and right halves.



Then merge the two halves.

Note:- This is the technique for merge sort.

Divide And conquer

2) Partition array into small items and large items, recursively sort the two sets. (This is the technique for quick sort).

"Applications"

Examples for Divide And conquer

1) searching e.g. Binary search
2) Sorting. e.g. merge sort, Quick sort
3) Tree Transversal
4) Matrix ~~Manipulation~~ Multiplication
5) strassen's algorithm.