Performance Analysis

- **Motivation:**
  - Estimation of required resources such as memory space, computational time, and communication bandwidth.
  - Comparison of algorithms

- **Model of implementation:**
  - One-processor RAM (random-access machine) model.
  - Single operations, such arithmetic operations & comparison operation, take constant time.

- What algorithm should we choose?
  - **COST**:
    - Time Complexity
    - Space Complexity

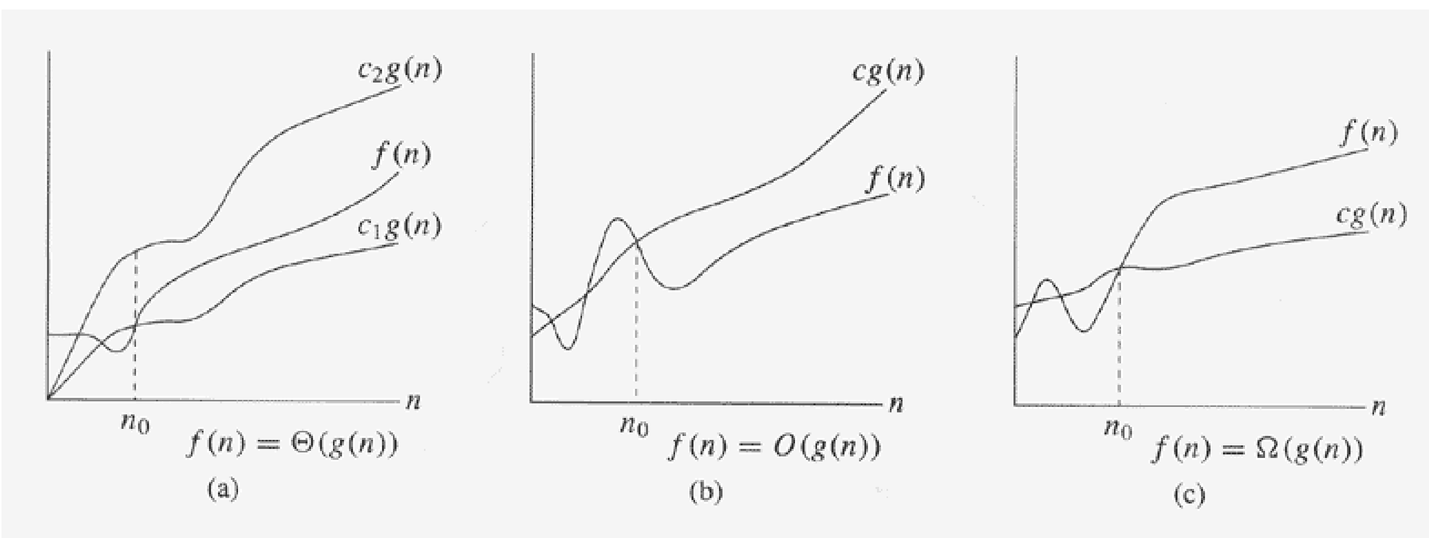
- **Time Complexity**:
  - The time complexity of a program is the time it needs to run to completion.

- **Space Complexity**:
  - The space complexity of a program is the amount of memory it needs to run to completion

- **Asymptotic Notation**:

    - **Objective:**
        - What is the rate of growth of a function?
        - What is a good way to tell a user how quickly or slowly an algorithm runs?

    - **Definition:**
        - A theoretical measure of the comparison of the execution of an algorithm, given the problem size n, which is usually the number of inputs.
        - To compare the rates of growth:
            - Big-O notation: Upper bound
            - Omega notation: lower bound
            - Theta notation: Exact notation



- **Big-O notation**:
  - Definition: $F(n)=O(g(n))$ iff there exist positive constants C and n_0 such that $F(n) \le Cg(n)$ when $n \ge n_0$.

    $g(n)$ is an **upper bound** of $F(n)$.

- **Example 1**:

  $F(n) = 3n+2$

  $F(n) = O(?)$

  For $2 \le n$ ====> $3n+2 \le 3n+n = 4n$

  ==> $F(n) = 3n+2 \le 4n$ ===> $F(n) = O(n)$.

  Where $C=4$ and $n_0=2$

- **Example 2**:

  $F(n) = 6*2^n+n^2$

  $F(n) = O(?)$

  $n^2 \le 2^n$ is true only when $n \ge 4$

  ===> $6*2^n+n^2 \le 6*2^n+n^2 = 7*2^n$

  ===> $C=7$ and $n_0=4$ and $F(n) \le 7*2^n$

  ===> $F(n) = O(2^n)$

- **Theorem**:

If $F(n) = a_m n^m + a_{m-1} n^{m-1} + \ldots + a_1 n + a_0 = \sum_{i=0}^{m} a_i n^i$ Then $F(n) = O(n^m)$.

- **Note**:
  $O(\log(n)) < O(n) < O(n\log n) < O(n^2) < O(n^3) < O(2^n)$.

- **Omega notation**:
  - Definition:
    $F(n)=\Omega(g(n))$ iff there exist positive constants C and n_0 such that $F(n) \geq Cg(n)$ when $n \geq n_0$.

    g(n) is a lower bound of F(n).

  - Example 1:
    $F(n) = 3n+2$
    $F(n) = \Omega(?)$

    Since $2 \geq 0$ ===> $3n+2 \geq 3n$ for $n \geq 1$
    Note that the inequality holds also for $n \geq 0$, however, the definition of Ω requires $n_0 > 0$.

    ===> $C=3$, $n_0=1$, and $F(n) \geq 3n$.
    ===> $F(n) = \Omega(n)$.

  - Be Careful:
    Note also that $3n+2 \geq 1$ for $n \geq 1$
    ===> $F(n) = \Omega(1)$.

  - <u>Theorem</u>:

If $F(n) = a_m n^m + a_{m-1} n^{m-1} + \ldots + a_1 n + a_0 = \sum_{i=0}^{m} a_i n^i$

and $a_m > 0$ then $F(n) = \Omega(n^m)$.

- Theta notation:
  - Definition:
    $F(n)=\Theta(g(n))$ iff there exist positive constants C_1, C_2 and n_0 such that $C_1*g(n) \leq F(n) \leq C_2*g(n)$ when $n \geq n_0$.
    $g(n)$ is called an **exact bound** of $F(n)$.

  - Example 1:
    $F(n) = 3n+2$
    $F(n) = \Theta(?)$

    We have shown that $F(n) \leq 4n$ and $F(n) \geq 3n$

    ===> $3n \leq F(n) \leq 4n$
    ===> $C_1 = 3$, $C_2 = 4$ and $n_0 = 2$.
    ===> $F(n) = \Theta(n)$

  - Example:
    Show that the exact bound of $F(n) = \sum_{i=1}^{n} i^k = \Theta(n^{k+1})$

  - <u>Theorem</u>:
    If $F(n) = a_m n^m + a_{m-1} n^{m-1} + \ldots + a_1 n + a_0 = \sum_{i=0}^{m} a_i n^i$

    and $a_m > 0$ then $F(n) = \Theta(n^m)$.

- **Common Functions:**

If F is:	Say that F is:	If F is:	Say that F is:
$O(1)$	Constant	$O(n^r)$, $1<r<2$	Subquadratic
$O(\log n)$	Logarithmic	$O(n^2)$	Quadratic
$O(\log^c n)$, $c>=1$	Polylogarithmic	$O(n^3)$	Cubic
$O(n^r)$, $0<r<1$	Sublinear	$O(n^c)$, $c>=1$	Polynomial
$O(n)$	Linear	$O(r^n)$, $r>1$	Exponential

- Properties:
  - Let $T_1(n) = O(f(n))$ and $T_2(n) = O(g(n))$
  - The sum rule:
    - Definition:

      If $T(n) = T_1(n) + T_2(n)$
      then $T(n) = O(\max(f(n),g(n))$.

    - Example:

      $T(n) = n^3 + n^2$ ===> $T(n) = O(n^3)$.

  - The product rule:
    - Definition:

      If $T(n) = T_1(n) * T_2(n)$
      then $T(n) = O(f(n)*g(n))$.

    - Example:

      $T(n) = n*n^*n$ ===> $T(n) = O(n^3)$.

  - The scalar rule:

- Definition:

  If $T(n) = T_1(n) * K$ where K is a constant
  then $T(n) = O(f(n))$.

- Example:

  $T(n) = n^2 * \frac{1}{2}$ ===> $T(n) = O(n^2)$.

- Be Careful:
  - Which is better $F(n) = 5* n^3$ or $G(n) = 95*n^2$?

$$\frac{\mathbf{F(n)}}{\mathbf{G(n)}} = \frac{\mathbf{5n^3}}{\mathbf{95n^2}} = \frac{\mathbf{1}}{\mathbf{19}}\mathbf{n}$$

  - Case 1: $\frac{\mathbf{n}}{\mathbf{19}} < \mathbf{1} \Rightarrow \mathbf{n} < \mathbf{19}$

    ===> $5* n^3 < 95*n^2$ F(n) is better .

  - Case 2: $\frac{\mathbf{n}}{\mathbf{19}} > \mathbf{1} \Rightarrow \mathbf{n} > \mathbf{19}$

    ===> $5* n^3 > 95*n^2$ G(n) is better .

- **Time Complexity of a Program:**

  - Comments: no time
  - Declaration: no time
  - Expressions and assignment statements: 1 time unit
  - Iteration statements:
    -

      ```
      For i =< exp1 >   to  < exp2 >  do
      Begin
              Statements.
      End
      ```

      The number of iterations of the loop times the time of Statements

    - While <exp> do: Similar to For loop.

- **Space Complexity of a Program:**

  - The total number of memory locations used in the declaration part
  - The memory needed for execution (Recursive programs)

- **Examples:**
  - Ex. 1: Fibonacci numbers:

Procedure FIB:	0
integer n, fn1, fb2, fn;	0
integer i;	0
Begin	1
read (n);	1
If n<2 Then	1
Print(n)	1
else begin	1
fn1=1; fn2=0;	1
For i=2 to n do	n
begin	n-1
fn = fn1 + fn2;	n-1
fn2 = fn1;	n-1
fn1 = fn;	n-1
end;	n-1
end	1
print(fn);	1
end;	1

Let $T(n)$ be the time complexity of FIB:

Total: $T(n) = 6n+8$ ====> $T(n) = O(n)$

- Ex. 2:

	Procedure BUBBLE(integer a: array);	0
	integer i,j,temp;	0
	begin	0
1	For i=1 to n-1 do	n-1
2	For j=n downto i+1 do	n-i
3	begin	1
4	If aj-1 > aj then	1
5	begin	1
6	temp = aj-1;	1
7	aj-1 = aj;	1
8	aj = temp;	1
9	end;	1
10	end;	1
	end;	

Let T(n) be the time complexity of BUBBLE:

- Line 3, 4, 5, 6, 7, 8, 9, 10
  O(max(1, 1, 1, 1, 1, 1, 1, 1) = O(1)
- move up line 2: O((n-i)*1) = O(n-i)
- move up line 1: $\sum_{i=1}^{n}(n-i) = \frac{(n-1)n}{2} = \frac{n^2}{2} - \frac{n}{2} = O(n^2)$

- Ex. 3:
  - Compute the time complexity of the following program:

```
Procedure Mystery (int n)
    For i=1 to n-1 do
        For j = i+1 to n do
            For k = 1 to j do
                x =x +1;
```

- Ex. 4:
  - Compute the time complexity of the following program fragment:

```
sum  = 0;
for i=1 to n do
    for j=1 to i do
        k = n**2
        while k>0 do
                sum=sum+1;
                k = k div 2;
```

- Ex. 5:
  - Compute the time complexity of the following program:

```
Procedure Puzzle (int n)
    For i=1 to n do
        For j = 1 to 10 do
            For k = n to n+5 do
                x =x +1;
```

- Ex. 6:
  - Compute the time complexity of the following program:

Procedure Foo (int n)

    For i=1 to $\sqrt{n}$ do

        For j = 1 to $\sqrt{n}$ do

            For k = 1 to $\sqrt{n}$ -j+1do

                x =x +1;