Data Structures and Algorithms

(ESO207)

Lecture 3:

- **Time complexity, Big "O" notation**
- Designing Efficient Algorithm
    - **Maximum sum subarray Problem**

Three algorithms

Algorithm for F(n)mod m	No. Instructions in RAM model
RFib(n,m)	$> 2^{(n-2)/2}$
IterFib(n,m)	$3n$
Clever_Algo_Fib(n,m)	$35 \log_2 (n-1) + 11$

- Which algorithm turned out to be the best $\boxed{\text{experimentally ?}}$

Lesson 1
from Assignment 1 ?

No. of instructions executed by algorithm in **RAM** model

Proportional to

Time taken by algorithm in real life

May be different for different input

Dependence on input

Time complexity of an algorithm

Definition:

the **worst case** number of instructions executed

as a **function** of the **input size** (or a parameter defining the input size)

The number of bits/bytes/words

Examples to illustrate input size :

Problem	Input size
Computing **F**(n)**mod** m for any positive integers n and m	$\log_2 n + \log_2 m$ bits
Whether an array storing j numbers (each stored in a **word**) is sorted ?	j words
Whether a $n \times m$ matrix of numbers (each stored in a **word**) contains "14" ?	nm words

Homework: What is the time complexity of **Rfib**, **IterFib**, **Clever-Algo-Fib** ?

Example:

Whether an array **A** storing j numbers (<u>each stored in a **word**</u>) is sorted ?

IsSorted(A)

{ $i \leftarrow 1$; — **1** time

flag ← true; — **1** time

while($i < \boldsymbol{j}$ and flag==true)

{

If (**A**[i]< **A**[$i-1$]) flag←false ;

$i \leftarrow i+1$;

} — $\boldsymbol{j-1}$ times in the <u>worst case</u>

Return flag ; — **1** time

}

Time complexity = $\boldsymbol{2j+1}$

Example:
Time complexity of matrix multiplication

Each element of the matrices occupies one **word** of RAM.

Matrix-mult(**C**[n, n],**D**[n, n])

{ **for** $i = 0$ to $\boldsymbol{n-1}$ — $\boldsymbol{n}$ times

{ **for** $j = 0$ to $\boldsymbol{n-1}$ — $\boldsymbol{n}$ times

{ **M**[i, j] ← 0;

for $k = 0$ to $\boldsymbol{n-1}$

{ **M**[i, j]← **M**[i, j] + **C**[i, k]*****D**[k, j];

}

} — $\boldsymbol{n+1}$ instructions

}

Return **M** — $\boldsymbol{1}$ time

}

Time complexity = $\boldsymbol{n^3 + n^2 + 1}$

Lesson 2 learnt from Assignment 1 ?

Algorithm for F(n)mod m	No. of Instructions
RFib(n,m)	$> 2^{(n-2)/2}$
IterFib(n,m)	$3n$
Clever_Algo_Fib(n,m)	$100\ \log_2(n-1) + 1000$

Question: What would have been the outcome if

No. of instructions of **Clever_Algo_Fib**(n,m) = $100\ \log_2(n-1) + 1000$

Answer: **Clever_Algo_Fib** would still be the fastest algorithm for large value of n.

COMPARING EFFICIENCY OF ALGORITHMS

Comparing **efficiency** of two algorithms

Let **A** and **B** be two algorithms to solve a given problem.

Algorithm **A** has time complexity : $2\,n^2 + 125$

Algorithm **B** has time complexity : $5\,n^2 + 67\,n + 400$

Question: Which algorithm is more efficient ?

Obviously **A** is more efficient than **B**

Comparing efficiency of two algorithms

Let **A** and **B** be two algorithms to solve a given problem.

Algorithm **A** has time complexity : $2\,n^2 + 125$

Algorithm **B** has time complexity : $50\,n + 125$

Question: Which one would you prefer based on the **efficiency** criteria ?

Answer : **A** is more efficient than **B** for $n < 25$

B is more efficient than **A** for $n > 25$

Time complexity is **really** an issue only when the input is of large size

Rule 1

Compare the **time complexities** of two algorithms for **asymptotically large value** of input size only

Comparing efficiency of two algorithms

Algorithm **B** with time complexity $50\,n + 125$

is certainly more efficient than

Algorithm **A** with time complexity : $2n^2 + 125$

A judgment question for you !

Algorithm **A** has time complexity **f**(n)= $5\ n^2$ + n + 1250

Researchers have designed two new algorithms B and C

- Algorithm **B** has time complexity **g**(n) = n^2 + 10
- Algorithm **C** has time complexity **h**(n) = $10\ n^{1.5}$ + $20\ n$ + 2000

Which of **B** and **C** is an improvement over **A** in the **true sense** ?

$$\lim_{n\to\infty} \frac{\mathbf{g}(n)}{\mathbf{f}(n)} = 1/5$$

$$\lim_{n\to\infty} \frac{\mathbf{h}(n)}{\mathbf{f}(n)} = 0$$

C is an improvement over **A** in the true sense.

Rule 2

An algorithm **X** is superior to another algorithm **Y** if
the **ratio** of time complexity of **X** and time complexity of **Y**
approaches 0 for asymptotically large input size.

Some Observations

Algorithm **A** has time complexity **f**(n)= $5\,n^2 + n + 1250$

Researchers have designed two new algorithms B and C

- Algorithm **B** has time complexity **g**(n) = $n^2 + 10$
- Algorithm **C** has time complexity **h**(n) = $10\,n^{1.5} + 20\,n + 2000$

Algorithm **C** is **the most efficient** of all.

Observation 1:

multiplicative or additive **Constants** do **not** play any role**.**

Observation 2:

The highest order term governs the time complexity asymptotically.

ORDER NOTATIONS

A **mathematical** **way**
to capture the **intuitions** developed till now.
(reflect upon it yourself)

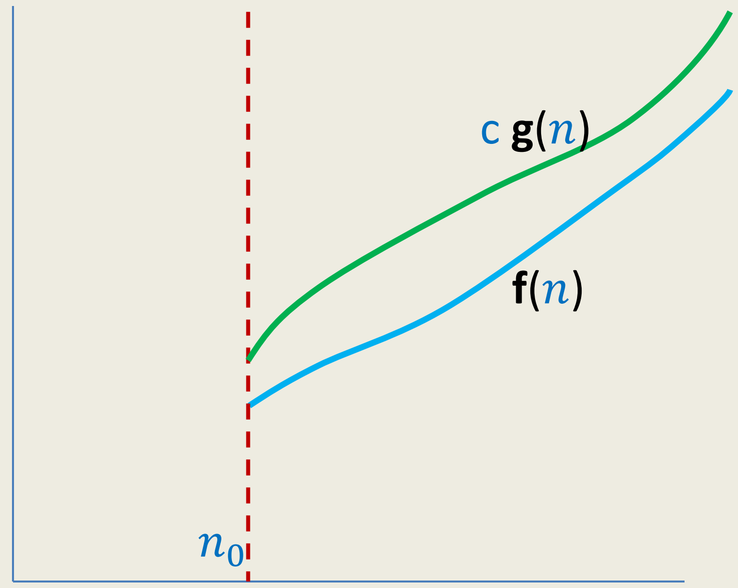
Order notation

Definition: Let **f**(n) and **g**(n) be any two increasing functions of n.

f(n) is said to be <u>of the order of</u> **g**(n)

if there exist constants c and n_0 such that

$$f(n) \le c\, g(n) \quad \text{for all } n > n_0$$



If **f**(n) is <u>of the order of</u> **g**(n),
we write **f**(n) = **O**(**g**(n))

Order notation : Examples

- $20\,n^2 \quad = \mathbf{O}(n^2)$ — $c = 20,\ n_0 = 1$
- ~~$100\,n + 60 \quad = \mathbf{O}(n^2)$~~ Loose — $c = 1,\ n_0 = 160$
- $100\,n + 60 \quad = \mathbf{O}(n)$ — $c = 160,\ n_0 = 1$
- ~~$10\,n^2 = \mathbf{O}(n^{2.5})$~~ Loose
- $2000 = \mathbf{O}(1)$

While analyzing time complexity of an algorithm accurately, our aim should be to choose the **g**(n) which is not loose.
Later in the course, we shall refine & extend this notion suitably.

Simple observations:

- If **f**(n) = **O**(**g**(n)) and **g**(n) = **O**(**h**(n)), then

  f(n) = **O**(**h**(n))

- If **f**(n) = **O**(**h**(n)) and **g**(n) = **O**(**h**(n)), then **f**(n) + **g**(n) = **?**(**h**(n))

These observations can be helpful for simplifying time complexity.

Prove these observation as Homeworks

A neat description of time complexity

- Algorithm **B** has time complexity **g**(n) = $n^2 + 10$

  Hence **g**(n) = **O**(n^2)

- Algorithm **C** has time complexity **h**(n) = $10\ n^{1.5} + 20\ n + 2000$

  Hence **h**(n) = **O**($n^{1.5}$)

- Algorithm for multiplying two $n \times n$ matrices has time complexity

  $n^3 + n^2 + 1$ = **O**(n^3)

Homeworks:

- **g**(n) = 2^n , **f**(n) = 3^n . Is **f**(n) = **O**(**g**(n)) ? Give proof.
- What is the time complexity of **selection sort** on an array storing n elements ?
- What is the time complexity of **Binary search** in a sorted array of n elements ?

HOW TO DESIGN EFFICIENT ALGORITHM ?

(This sentence captures precisely the goal of theoretical computer science)

Designing an efficient algorithm

Facts from the world of algorithms:

1. **No formula** for designing efficient algorithms.
2. Every new problem demands a **fresh** approach.

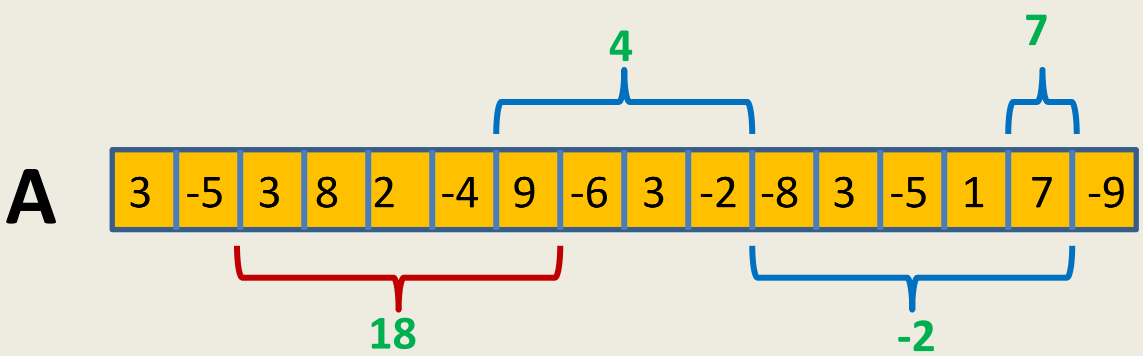
3. Designing an efficient algorithm or data structure requires
    1. Ability to make **key observations**.
    2. Ability to ask **right kind of questions**.
    3. A **positive attitude** and …
    4. a lot of **perseverance**.

We shall demonstrate the above facts during this course many times.

Max-sum subarray problem

Given an array **A** storing n numbers,

find its **subarray** the sum of whose elements is maximum.



Max-sum subarray problem: A trivial algorithm

```
A_trivial_algo(A)
{ max← A[0];
  For i=0 to n-1
     For j=i to n-1
          {    temp ← compute_sum(A,i,j);

               if max< temp  then max← temp;

          }
 return max;
}

compute_sum(A, i,j)
{   sum←A[i];
    For k=i+1 to j      sum← sum+A[k];
    return sum;
}
```

Homework: Prove that its time complexity is **O**(n^3)

Max-sum subarray problem:

Question: Can we design **O**(n) time algorithm for Max-sum subarray problem ?

Answer: Yes.

Think over it with a fresh mind ….
We shall design it together in the next class…☺