Data Structures and Algorithms

(CS210A)

Semester I – **2014-15**

Lecture 3:

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

Which algorithm turned out to be the best ?

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)	$27 \log_2 (n-1) + 6$

Assignment 1 - part 1 is over.

Lesson 1 learnt from Assignment 1 ?

Time taken by algorithm in real life = **Proportional to** No. of instructions executed by algorithm in **RAM** model

Inferences:

- The difference in the time of individual instructions (**+,*,if,…**) is irrelevant.
- **RAM** model of computation is a very **accurate** model for measuring efficiency of algorithms.

Time complexity of an algorithm

Definition:

The time complexity of an algorithm is the **worst case** number of instructions executed as a **function** of the **input size** (or a parameter defining the input size).

Example: the time complexity of searching for a '0' in a matrix **M**$[n, n]$ is at most n^2+ c for some constant c.

Example:

Time complexity of matrix multiplication

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

{ **for** i = 0 to $n-1$ — n times

{ **for** j=0 to $n-1$ — n times

{ **M**[i,j] ← 0;

for k=0 to $n-1$

{ **M**[i,j]← **M**[i,j] + **C**[i,k]***D**[k,j]; — $n+1$ instructions

}

}

}

Return **M** — 1 time

}

Time complexity = $n^3 + n^2 + 1$

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)	$27 \log_2 (n-1) + 6$

Question: What would have been the outcome if

No. of instructions of **Clever_Algo_Fib**(n,m) = $100 \log_2 (n-1) + 60$

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** has time complexity : $n^2 + 125$

A judgment question for you !

Algorithm **A** for a given problem 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 <u>asymptotically large input size</u>.

Some Observations

Algorithm **A** for a given problem has time complexity **f**(n)= $5n^2 + n + 1250$

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) = $10n^{1.5} + 20n + 2000$

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** govern the time complexity asymptotically.

ORDER NOTATIONS

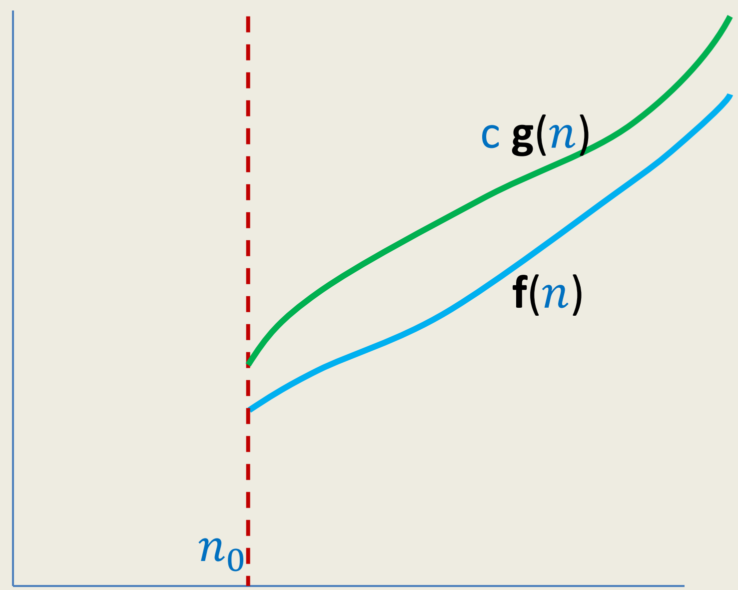
a neat and precise way to describe
Time Complexity

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

- $20\,n^2$ = **O**(n^2)
- $100\,n + 60$ = **O**(n^2)
- $100\,n + 60$ = **O**(n)
- n^2 = **O**$(n^{2.5})$
- 2000 = **O**(1)

c =20, $n_0 = 1$

c =1, $n_0 = 160$

c =160, $n_0 = 1$

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×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. There is **no formula** for designing efficient algorithms.
2. Almost every new problem demands a **fresh** approach.

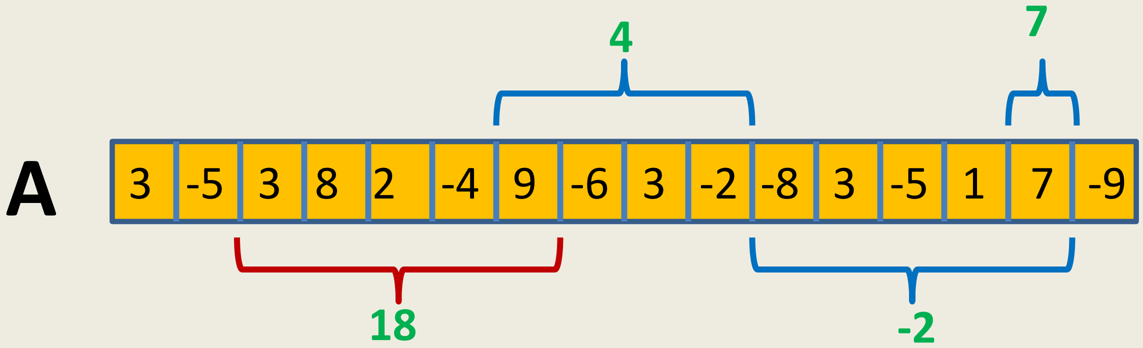
3. Designing an efficient algorithm or data structure requires
   1. Ability to make **key observations**.
   2. Ability to ask **right kind of questions**.
   3. A **positive attitude** and …
   4. a lot of **perseverance**.

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;
}
```

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

compute_sum(A, i,j)

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

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…☺