Data Structures and Algorithms

(CS210/CS210A)

Lecture 1:

- An **overview** and **motivation** for the course
- some **concrete** examples.

The website of the course

moodle.cse.iitk.ac.in

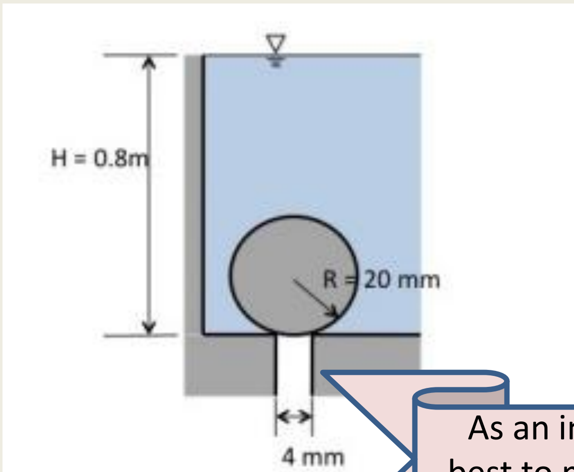
▼CSE

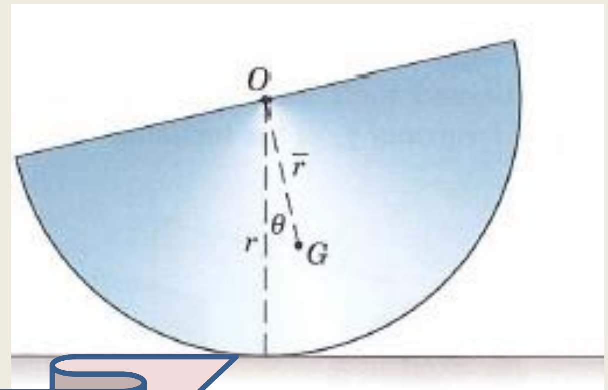
CS210: Data Structures and Algorithms

(guest login allowed)

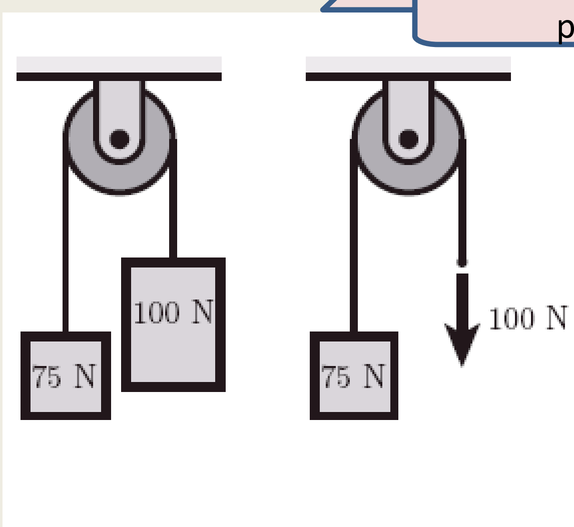
This course will be taught **differently.**

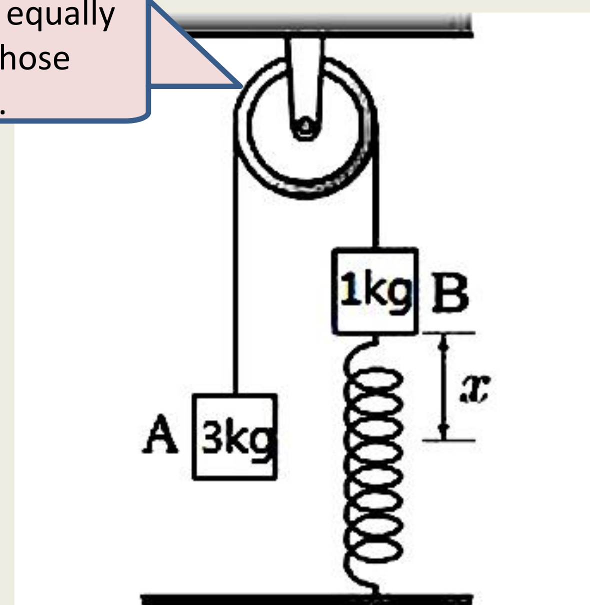
Those were the golden moments…





As an instructor, I shall try my best to make this course equally enjoyable as solving those physics problems.





Prerequisite of this course

- A good command on Programming in C
  - Programs involving arrays
  - Recursion
  - Linked lists (preferred)

- **Fascination for solving Puzzles**

Salient features of the course

- **Every concept** — **<u>We</u>** shall **<u>re-invent</u>** in the class itself.
- **Solving each problem** — Through discussion **in the class**.

solution will emerge naturally if we ask

right set of questions

and then try to find their **answers**.

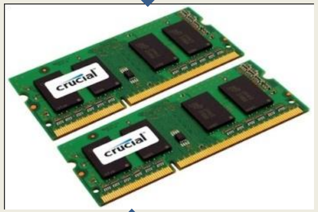
… so that finally it is a concept/solution derived by <u>you</u>
and not a concept from some scientist/book/teacher.

Isn't that nice ☺

Let us open a desktop/laptop

A processor (CPU)

speed = few GHz
(a few **nanoseconds** to execute an instruction)



Internal memory (RAM)

size = a few GB (Stores a billion bytes/words)
speed = a few GHz(a few **nanoseconds** to read a byte/word)

External Memory (Hard Disk Drive)

size = a few tera bytes
speed : seek time = **miliseconds**
transfer rate= around **billion** bits per second

A simplifying assumption
(for the rest of the lecture)

It takes around a few **nanoseconds** to execute an instruction.

*(This assumption is **well supported** by the modern day computers)*

EFFICIENT ALGORITHMS

What is an algorithm ?

Definition:

A **finite sequence** of **well defined instructions** required to solve a given computational problem.

A prime objective of the course:

Design of **efficient** algorithms

WHY SHOULD WE CARE FOR EFFICIENT ALGORITHMS

WE HAVE PROCESSORS RUNNING AT GIGAHERTZ?

Revisiting problems from ESC101

Problem 1:
Fibonacci numbers

Fibonacci numbers

$F(0) = 0;$
$F(1) = 1;$
$F(n) = F(n-1) + F(n-2)$ for all $n > 1;$

$$F(n) \approx a \cdot b^n$$

An easy exercise : Using induction or otherwise, show that

$$F(n) > 2^{\frac{n-2}{2}}$$

Algorithms you must have implemented for computing $F(n)$:

- **Iterative**
- **recursive**

Iterative Algorithm for $F(n)$

```
IFib(n)
if n=0 return 0;
    else if n=1 return 1;
        else {
                a ← 0;  b← 1;
                For(i=2 to n) do
                {    temp ← b;
                     b← a+b;
                     a← temp;
                }
             }
  return b;
```

Recursive algorithm for $F(n)$

Rfib(n)

```
{   if n=0 return 0;
      else if n=1 return 1;
            else return(Rfib(n-1) + Rfib(n-2))
}
```

Homework 1

(compulsory)

Write a **C** program for the following problem:

Input: a number n

n : **long long int** (64 bit integer).

Output: **F(n) mod 2014**

Time Taken	Largest n for Rfib	Largest n for IFib
1 minute		
10 minutes		
60 minutes		

Problem 2:
Subset-sum problem

Input: An array **A** storing n numbers, and a number s

A											
A	12	3	46	34	19	101	208	120	219	115	220

Output: Determine if there is a subset of numbers from **A** whose sum is s.

The fastest existing algorithm till date : $2^{n/2}$ instructions

- Time for $n = 100$ **At least an year**
- Time for $n = 120$ **At least 1000 years**

on the fastest existing computer.

Problem 3:
Sorting

Input: An array **A** storing **n** numbers.

Output: Sorted **A**

A fact:

A significant fraction of the code of all the software is for sorting or searching only.

To sort **10 million** numbers on the present day computers

- **Selection sort** will take at least a few hours.
- **Merge sort** will take only a few seconds.
- **Quick sort** will take ??? .

How to design efficient algorithm for a problem ?

Design of algorithms and data structures is also an Art

Requires:

- **Creativity**
- **Hard work**
- **Practice**
- **Perseverance** (most important)

Summary of Algorithms

- There are many practically relevant problems for which there does not exist any efficient algorithm till date ☹. (How to deal with them ?)

- Efficient algorithms are <u>important for theoretical as well as practical</u> purposes.

- <u>Algorithm design is an art</u> which demands a lot of creativity, intuition, and perseverance.

- More and more applications in real life require efficient algorithms
  - Search engines like **Google** exploits many clever algorithms.

THE DATA STRUCTURES

An Example

Given: a telephone directory storing telephone no. of **hundred million** persons.

Aim: to answer a sequence of **queries** of the form

"what is the phone number of a given person ?".

Solution 1 :

Keep the directory in an array.

do **sequential search** for each query.

Time per query: around **1/10th** of a **second**

Solution 2:

Keep the directory in an array, and **sort it** according to names,

do **binary search** for each query.

Time per query: less than **100 nanoseconds**

Aim of a data structure ?

To store/organize a given data in the memory of computer so that each subsequent operation (query/update) can be performed quickly ?

Range-Minima Problem

A Motivating example

to realize the importance of data structures

Range-Minima Problem

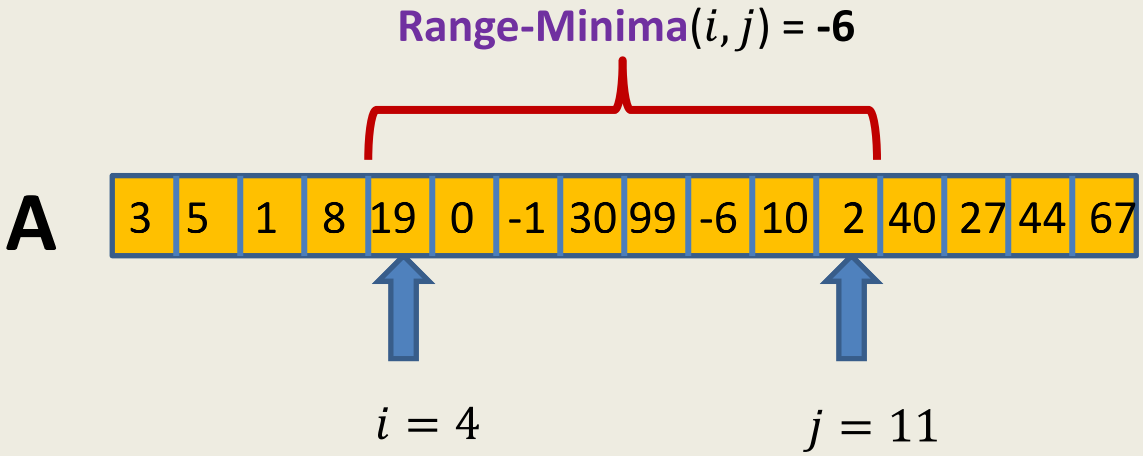
Given: an array **A** storing n numbers,

Aim: a data structure to answer a sequence of queries of the following type

Range-minima(i, j) : report the smallest element from **A**[i],…,**A**[j]

Let n = one **million**.

No. of queries = **10 millions**



Range-Minima Problem

Applications:

- **Computational geometry**
- **String matching**
- **As an efficient subroutine in a variety of algorithms**

(we shall discuss these problems sometime in this course or the next level course CS345)

Range-Minima Problem

Solution 1:

Answer each query in a brute force manner using **A** itself.

```
Range-minima-trivial(i,j)
{    temp ← i+1;
     min ← A[i];
     While(temp <= j)
     {   if (min > A[temp])
              min ← A[temp];
         temp← temp+1;
     }
     return min
}
```

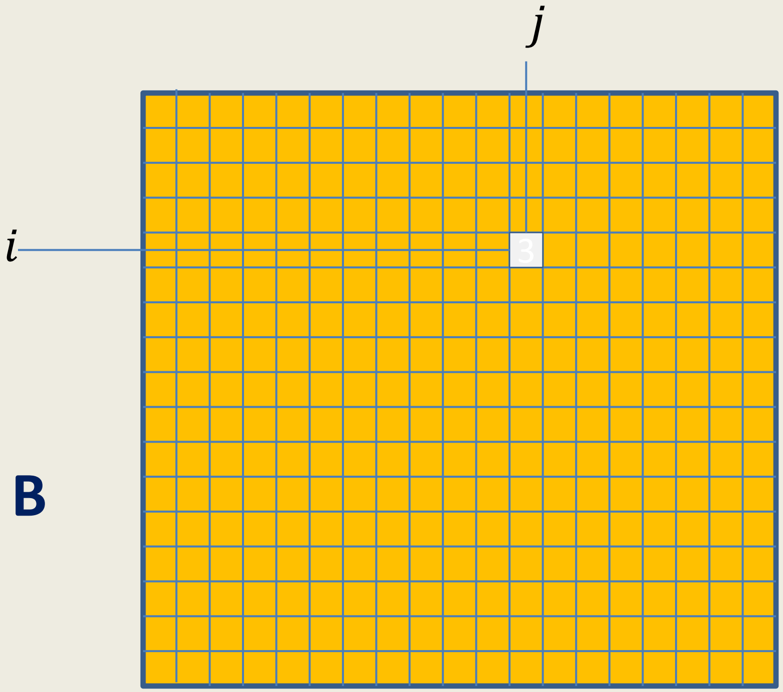
Time taken to answer a query: few **milliseconds**

Time for answering **all** queries:
a few hours

Range-Minima Problem

Solution 2:

Compute and store answer for each possible query in a $n \times n$ matrix **B**.



B$[i][j]$ stores the smallest element from **A**$[i]$,…,**A**$[j]$

Space : roughly n^2 words.

Solution 2 is Theoretically efficient but practically impossible

Size of **B** is **too large** to be kept in RAM. So we shall have to keep most of it in the **Hard disk drive.** Hence it will take a few **milliseconds per query**.

Range-Minima Problem

Question: Does there exist a data structure for Range-minima which is

- **Compact**

  (nearly **the same size** as the input array A)
- **Can answer each query efficiently ?**

  (a few **nanoseconds** per query)

Homework 2: Ponder over the above question.

(we shall solve it soon)

Range-1-Query

Determining if a rectangle has at least one 1 ?

- **Data structure**: a few tables.
- **Query time**: a few nanoseconds.

0	0	0	1	1	0	1	0
0	1	1	0	1	0	0	1
1	0	1	0	0	1	1	1
1	0	0	0	0	1	0	0
0	1	0	1	0	0	0	0
0	0	1	0	1	0	0	1
1	0	0	0	0	0	0	1
0	0	1	1	1	0	0	1

Any idea about when this result was derived ?

in a Conference in 2015

Data structures to be covered in this course

Elementary Data Structures

- Array
- List
- Stack
- Queue

Hierarchical Data Structures

- Binary Heap
- Binary Search Trees

Augmented Data Structures

- Look forward to working with all of you to make this course enjoyable.

- This course will be light in contents (no formulas)

  But it will be very demanding too.

- In case of any difficulty during the course,

  just drop me an email without any delay.

  I shall be happy to help ☺