# Algorithm Analysis
## CS 213 Minors

Instructor: Prof. Ganesh Ramakrishnan

www.cse.iitb.ac.in/~cs213m

Ref: Chapter 2, DSAA by Weiss

(Developed on slides from Prof. Varsha Apte)

# Programming – what you have learnt so far

- Writing a program that works correctly
  - Various paradigms of writing such a program
- Programming constructs that make a program readable, elegant, etc
  - Recursion (elegant!)
  - Functions (convenient)

# …Programming – what you have learnt so far

- Invariants, assertions
- Proving program correctness
  - An essential “scientific” technique
- In Summary
  - CS 101 taught you the basics of programming in C++
  - CS 152 stressed on paradigms that should guide your programming style (**elegance**)

# What next?

- In this course, the biggest emphasis is on

# EFFICIENCY

Of a program.

- We no longer want to just write "*a program*" (which is elegant, and provably correct)
- We want to write an efficient program (which is also elegant and provably correct)
  - Never in your life should you lose sight of elegance!

Efficiency = M/c friendly  Elegance = Programmer friendly

# Efficiency…what is it?

- Once we mention any quality, as scientists and engineers, we must define *metrics* or a *rigorous mathematical definition* of how a quality can be measured
- How do we define efficiency?
  - In terms of resources used, and work done

Eg metrics

① Number of instructions of program executed as a function of input size

could be C++

② Number of weighted instructions of program executed as a function of input size

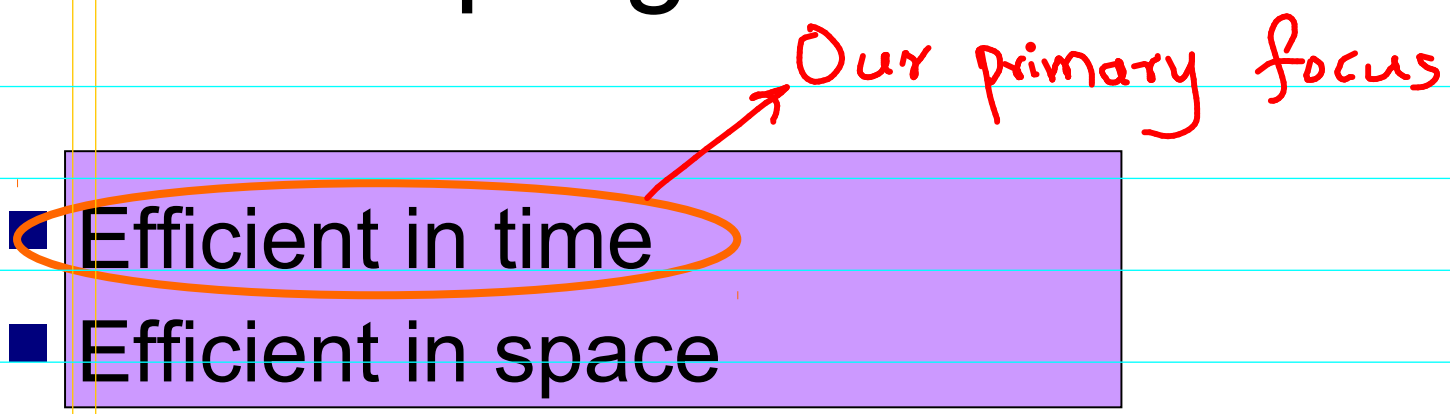
weighted by, say, the # of assembly language instructions corresponding to a single C language instruction. Eg: "*" is much more expensive than "+".

The # of assembly lang instructions could be a function of the processor (eg: A DSP processor has efficient multipliers)

③ Number of RAM, L1/L2 Cache, Hard disk accesses

④ N/W accesses

# Efficient programs




- Efficient in time
- Efficient in space
  - RAM, disk, etc
- Efficient in….?
  - Communication overhead (distributed programs)
  - etc

# Running time of programs Example: Search

- Consider a simple example:
  - Searching for an element in a sorted array
  - Given array A of some type T, and an element E of type T, set “found” to true

# Program code fragment

```
for (i=0; i < size; i++) {
            if (A[i] == num) {
              found = true;
              break;
            }
```

**Algorithm A**

*This code is not making use of the fact that array A is sorted. We will leave it this way for simplicity.*

**Algorithm B**

```
bsearch(vector<int> &a, int num, int begin,
int end) {

  int mid;
  bool found;
  mid  = (begin + end)/2;
  if (begin > end) {
        found = false;
  }
  else {
   if (a[mid] == num) {
      found=true;
    }
    else if (num < a[mid]) {
      bsearch(a,num, begin, mid-1);
    }
    else
      bsearch(a,num,mid+1, end);
  }
}
```

H/W: Compute # of instructions executed by programs A & B as a function of

(a) length of the list (i.e the initial value of end - begin)

(b) num