Lecture 03

Complexity of Algorithms

- **An *algorithm* is a set of instructions that a computer will follow**
  - **examples**

- **Solutions to most modern problems require complex algorithms**
  - **Examples**

- **Efficiency of an algorithm can be measured in two ways**
  - **Time efficiency**
  - **Space Efficiency**

- **Sometimes we have to sacrifice one to get the other**

- **Algorithm execution time depends on many factors**
  - **Processor, compiler, language, data size, memory management etc..**

- **Lets assume standard Model of Computation**
  - **uni-processor, RAM, Sequential instructions etc..**

- **Input size plays a crucial part in algorithm analysis, and we will describe performance of an algorithm using input size n**
  - **Example: how long does it take to reverse an array of size n?**

<u>Example: Bubble Sort</u>

```
for i = 1 to n-1

  for j = 0 to n-i-1 do

     if (A[j]>A[j+1]) swap (A[j], A[j+1]);
```

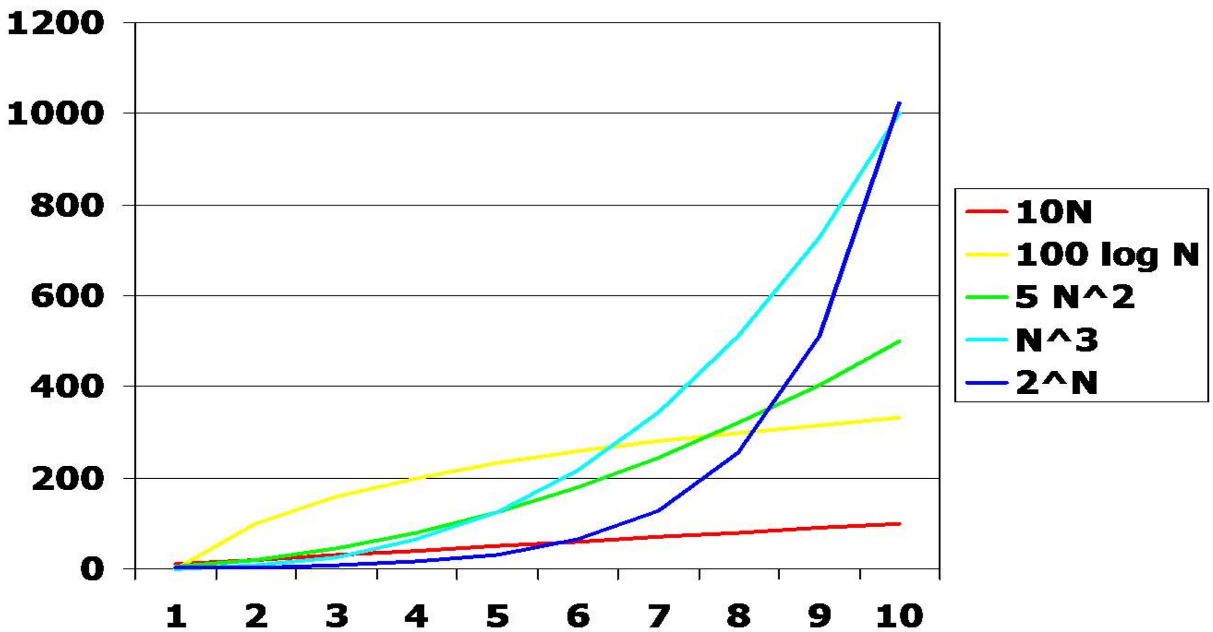
- **Let $T_P(n)$ be the performance of an algorithm P as a function of n. Find $T_P(n)$ for bubble sort**

- **Lets count the operations (counting is one of the skills we have to learn)**

Exercises

- **Find $T_P(n)$ when algorithm P is**
  - **Finding the minimum in an arbitrary array**
  - **Finding the max in a sorted array**
  - **Finding duplicates in an arbitrary array**
  - **Finding all permutations of n items**
  - **Finding the shortest distance between two cities**

- Now that we can describe the performance of an algorithm as a function of input size n, we will attempt to describe performance of standard algorithm using some known functions



- Now we develop a notation to describe the performance of an algorithm

- We can obtain upper bounds, lower bounds and absolute bounds for time efficiency of an algorithm

- We shall discuss Big-O, Big-Ω, Big-θ and Little-o notations that can be used to get bounds for performance of an algorithm
  - We will only discuss big O in this course

- **<u>Formal Definition</u>: Given a function T:N→ N that describes the running time of an algorithm on an input of size N, we say**
- **T(n) = *O*(f(n)) if**
  - **there are positive constants c and n_0 such that $T(n) \leq c \times f(n)$ when $n \geq n_0$.**
  - **c is called the *constant factor*.**
  - **The n_0 constant says that at some point, $c \times f(N)$ is always bigger than T(n)**
  - **So we have an upper bound for T(n)**

- **Space complexity is also an interesting metric for assessing the efficiency of a program.**
  - **How much space is used by my program during runtime?**
  - **Eg: Object [ ] A = new Object[N];**

- **We can get some measurement of how much space will be used by the program by looking at the code**
  - **Expressed in big O, big Omega .. notations**

- **But we need to look at some of the Java API's to get a true use of memory during execution of a program**
- **Class Runtime is available from Java API**
- **Interfaces with the environment current application is running**
- **Several interesting methods (see more on API)**
  - **<u>availableProcessors</u>()**
    Returns the number of processors available to the Java virtual machine.
  - **<u>gc</u>() Runs the garbage collector.**
  - **<u>maxMemory</u>()**
    Returns the maximum amount of memory that the Java virtual machine will attempt to use.

- **totalMemory()**
  Returns the total amount of memory in the Java virtual machine.

Summary

- **Runtime of an algorithm depends on many factors**
- **However, an asymptotic analysis of the algorithm can be obtained using the size of the input data n**
- **Complexity can be discussed in terms of**
  - **Time**
  - **Space**