

## Topic 8

# Introduction to Analysis of Algorithms

09/30/11

1-1

## Objectives

- To introduce the concept of analysing algorithms with respect to the time taken to have them executed
  - Purpose:
    - To see if an algorithm is feasible
    - To compare different algorithms for solving a problem
- (There will be much more on this later)

09/30/11

1-2

1-2

## Introduction to Analysis of Algorithms

- One aspect of software quality is the efficient use of **computer resources** :
  - CPU time
  - Memory usage
- We frequently want to analyse algorithms with respect to **execution time**
  - Called **time complexity** analysis
  - For example, to decide which sorting algorithm will take less time to run

09/30/11

1-3

1-3

## Time Complexity

- Analysis of time taken is based on:
  - Problem size (e.g. number of items to sort)
  - Key operations (e.g. comparison of two values)
- What we want to analyse is the **relationship** between
  - The size of the problem, **n**
  - And the time it takes to solve the problem, **t(n)**
    - Note that t(n) is a function of n, so it depends on the size of the problem

09/30/11

1-4

1-4

## Growth Functions

- This **t(n)** is called a **growth function**
- What does a growth function look like?
  - Example of a growth function for some algorithm:  
 **$t(n) = 15n^2 + 45n$** 
    - See the next slide to see how t(n) changes as n gets bigger!

09/30/11

1-5

1-5

## Example: $15n^2 + 45n$

| No. of items n | $15n^2$            | $45n$      | $15n^2 + 45n$      |
|----------------|--------------------|------------|--------------------|
| 1              | 15                 | 45         | 60                 |
| 2              | 60                 | 90         | 150                |
| 5              | 375                | 225        | 600                |
| 10             | 1,500              | 450        | 1,950              |
| 100            | 150,000            | 4,500      | 154,500            |
| 1,000          | 15,000,000         | 45,000     | 15,045,000         |
| 10,000         | 1,500,000,000      | 450,000    | 1,500,450,000      |
| 100,000        | 150,000,000,000    | 4,500,000  | 150,004,500,000    |
| 1,000,000      | 15,000,000,000,000 | 45,000,000 | 15,000,045,000,000 |

09/30/11

1-6

## Comparison of Terms in $15n^2 + 45n$

- When  $n$  is small, which term is larger?
- But, as  $n$  gets larger, note that the  $15n^2$  term grows more quickly than the  $45n$  term
- Also, the constants 15 and 45 become irrelevant as  $n$  increases
- We say that the  $n^2$  term is **dominant** in this expression

09/30/11

1-7

## Big-O Notation

- It is not usually necessary to know the exact growth function
- The key issue is the **asymptotic complexity** of the function : **how it grows as  $n$  increases**
  - This is determined by the **dominant term** in the growth function (the term that increases most quickly as  $n$  increases)
  - Constants and secondary terms become irrelevant as  $n$  increases

09/30/11

1-8

1-8

## Big-O Notation

- The asymptotic complexity of the function is referred to as the **order of the algorithm**, and is specified by using **Big-O notation**
  - **Example:  $O(n^2)$**  means that the time taken by the algorithm grows like the  $n^2$  function as  $n$  increases
  - **$O(1)$**  means constant time, regardless of the size of the problem

09/30/11

1-9

1-9

## Some Growth Functions and Their Asymptotic Complexities

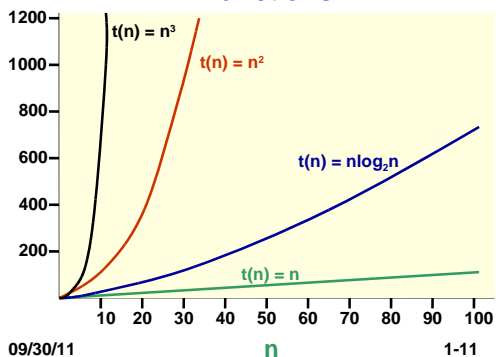
| Growth Function                | Order             |
|--------------------------------|-------------------|
| $t(n) = 17$                    | $O(1)$            |
| $t(n) = 20n - 4$               | $O(n)$            |
| $t(n) = 12n * \log_2 n + 100n$ | $O(n * \log_2 n)$ |
| $t(n) = 3n^2 + 5n - 2$         | $O(n^2)$          |
| $t(n) = 2^n + 18n^2 + 3n$      | $O(2^n)$          |

09/30/11

1-10

1-10

## Comparison of Some Typical Growth Functions



09/30/11

$n$

1-11

1-11

## Exercise: Asymptotic Complexities

| Growth Function             | Order |
|-----------------------------|-------|
| $t(n) = 5n^2 + 3n$          | ?     |
| $t(n) = n^3 + \log_2 n - 4$ | ?     |
| $t(n) = \log_2 n * 10n + 5$ | ?     |
| $t(n) = 3n^2 + 3n^3 + 3$    | ?     |
| $t(n) = 2^n + 18n^{100}$    | ?     |

09/30/11

1-12

1-12

## Determining Time Complexity

- Algorithms frequently contain sections of code that are executed over and over again, i.e. **loops**
- So, **analysing loop execution** is basic to determining time complexity

09/30/11

1-13

## Analysing Loop Execution

- A loop executes a certain number of times (say  $n$ ), so the time complexity of the loop is  $n$  times the time complexity of the body of the loop
- Example:** what is the time complexity of the following loop, in Big-O notation?

```
x = 0;
for (int i=0; i<n; i++)
    x = x + 1;
```

09/30/11

1-14

1-14

- Nested loops:** the body of the outer loop includes the inner loop
- Example:** what is the time complexity of the following loop, in Big-O notation?

```
for (int i=0; i<n; i++)
{
    x = x + 1;
    for (int j=0; j<n; j++)
        y = y - 1;
}
```

09/30/11

1-15

1-15

## More Loop Analysis Examples

```
x = 0;
for (int i=0; i<n; i=i+2)
{
    x = x + 1;
}
```

```
x = 0;
for (int i=1; i<n; i=i*2)
{
    x = x + 1;
}
```

09/30/11

1-16

1-16

## More Loop Analysis Examples

```
x = 0;
for (int i=0; i<n; i++)
    for (int j = i, j < n, j++)
    {
        x = x + 1;
    }
```

09/30/11

1-17

1-17

## Analysis of Stack Operations

- Stack operations are generally efficient, because they all work on only one end of the collection
- But which is more efficient: the array implementation or the linked list implementation?

09/30/11

1-18

## Analysis of Stack Operations

- $n$  is the number of items on the stack
- **push** operation for **ArrayStack**:
  - $O(1)$  if array is not full (why?)
    - What would it be if the array is full? (**worst case**)
- **push** operation for **LinkedStack**:
  - $O(1)$  (why?)
- **pop** operation for each?
- **peek** operation for each?