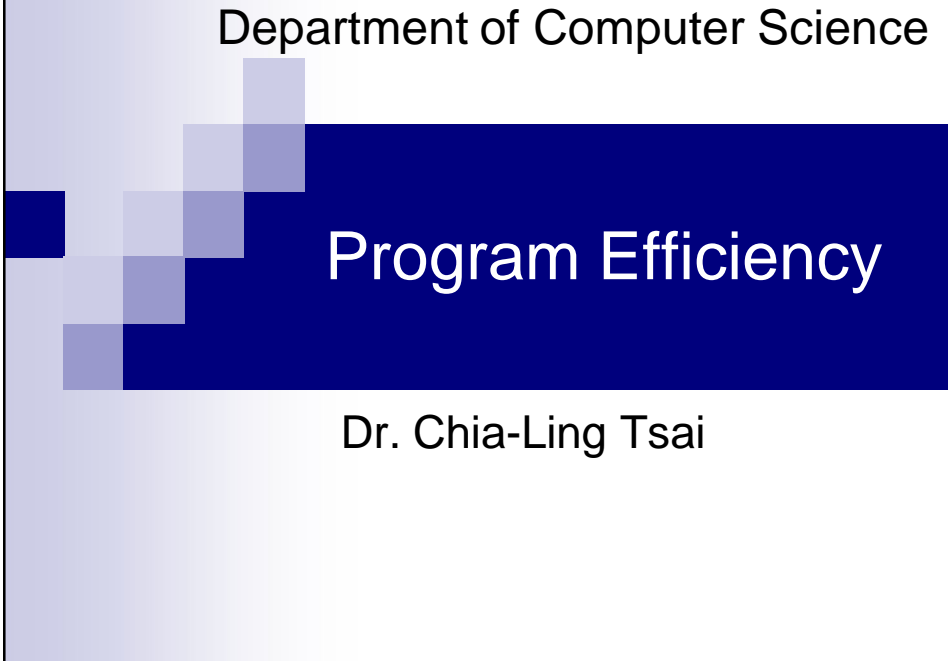



Department of Computer Science



# Program Efficiency

Dr. Chia-Ling Tsai



## Outline

- Motivation
- Algorithm analysis
- Complexity classes
- Sorting

## Motivation

- There are many ways to solve the same problem
- The speed matters if the program is needed to process megabytes of data
- Users are getting less patient

## Algorithm Analysis

- The program efficiency often refers to *time efficiency*.
  - Task-dependent
- Performance is measured by roughly how many statements are executed
- Assume all statements of the same complexity
  - Not true in reality!

## Statements for consideration

- Statements to be considered in the analysis that takes time to execute:
  - Variable declarations and assignments
  - Evaluating mathematical and logical expressions
  - Accessing or modifying an individual element of an array

## Counting

```
statement1. }  
statement2. } 3  
statement3. }
```

## Counting

```
for (N times) {  
    for (N times) {  
        statement1.  
    }  
}
```

```
for (N times) {  
    statement2.  
    statement3.  
}
```

On the order of  $N^2 == O(N^2)$

## Example: Range

- Task: computing the statistical range of numbers in an array.
- Definition 1: the largest difference of all number pairs in the array

```
public static int range(int[] numbers)  
{  
    int maxDiff = 0;  
    for (int i=0; i<numbers.length; i++) {  
        for (int j=0; j<numbers.length; j++) {  
            int diff=Math.abs(numbers[j]-numbers[i]);  
            maxDiff=Math.max(maxDiff, diff);  
        }  
    }  
    return maxDiff;  
}
```

## Tweaking

- Every pair is examined twice—wasting of computation
- Only examine pairs  $(i,j)$  where  $j>i$

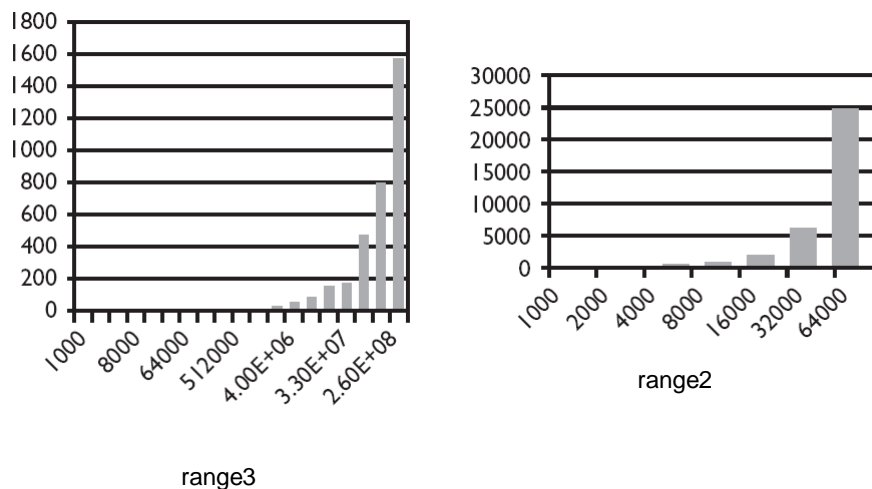
```
public static int range2(int[] numbers)
{
    int maxDiff = 0;
    for (int i=0; i<numbers.length; i++) {
        for (int j=i+1; j<numbers.length; j++) {
            int diff=Math.abs(numbers[j]-numbers[i]);
            maxDiff=Math.max(maxDiff, diff);
        }
    }
    return maxDiff;
}
```

## Redesign

- Definition 2: the difference between the lowest and the highest numbers in the array

```
public static int range3(int[] numbers)
{
    int max=numbers[0];
    int min=max;
    for (int i=1; i<numbers.length;i++) {
        if (numbers[i]>max) {
            max=numbers[i];
        } else if (numbers[i]<min) {
            min=numbers[i];
        }
    }
    return max-min;
}
```

## Runtime comparison



## Complexity classes

- $O(1)$  or constant time: runtime independent of input size
- $O(\log N)$  or logarithmic: typically dividing the problem space in half repeatedly until problem solved.
- $O(N)$  or linear: runtime directly proportional to  $N$
- $O(N \log N)$ ,  $O(N^2)$ ,  $O(N^3)$ ,  $O(2^N)$

## Searching—sequential

- What is the order of complexity for sequential search?

```
public static int sequential(int[] numbers, int target)
{
    for (int i=0; i<numbers.length; i++) {
        if (numbers[i]==target) {
            return i;
        }
    }
    return -1; //not found
}
```

## Searching—binary

Input:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
11	18	29	37	42	49	51	63	69	72	77	82	88	91	98

Repeat #1:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
11	18	29	37	42	49	51	63	69	72	77	82	88	91	98
↑							↑							↑
min							mid							max

observation: `numbers[7] < 77`

decision: search indexes 8 – 14

## (continue)

Repeat #2:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
11	18	29	37	42	49	51	63	69	72	77	82	88	91	98
								↑		↑				↑
								min		mid				max

observation: numbers[11] > 77  
 decision: search indexes 8 – 10

Repeat #3:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
11	18	29	37	42	49	51	63	69	72	77	82	88	91	98
								↑	↑	↑				
								min	mid	max				

observation: numbers[9] > 77  
 decision: search index 10

## (continue)

Repeat #4:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
11	18	29	37	42	49	51	63	69	72	77	82	88	91	98
										↑				
										min				
										mid				
										max				

```
public static int binary(int[] numbers, int target)
{
    int min=0;
    int max=numbers.length-1;

    while (min<=max) {
        int mid=(max+min)/2;
        if (numbers[mid]==target){
            return mid; // found it!
        } else if (numbers[mid]<target) {
            min=mid+1; // too small
        } else { //numbers[mid] > target
            max=mid-1; // too large
        }
    }
    return -1; // not found
}
```

What's the order of complexity?



## Summary

- Algorithm analysis
- Complexity classes
- Analysis on searching algorithms