A pair of glasses with a dark frame and light-colored lenses is resting on a piece of white paper. The background is a soft, out-of-focus yellow and orange gradient.

# Data Structures

## Chapter 1

1. Recursion

**2. Performance Analysis**

- Space Complexity
- Time Complexity
- Step Count

3. Asymptotic Analysis

# Performance Analysis

---

- The program we write should
    1. meet the specification.
    2. work correctly.
    3. be documented properly.
    4. run effectively
    5. be readable.
    6. **use the storage effectively – space**
    7. **run timely – time**
- } space & time complexity

The **space complexity** of a program is the amount of **memory** that it needs to run to completion.

The **time complexity** of a program is the amount of computer **time** that it needs to run to completion.

# Performance Analysis

---

- **Space complexity:**
- Fixed space requirements :  $\mathbf{c}$ 
  - that do not depend on input size, simple or fixed-size variables
- Variable space requirements :  $S_p(I)$ 
  - that depend on the instance  $I$ , stack, variable

The total space requirement for the program  $P$ :

$$S(P) = \mathbf{c} + S_p(I)$$

where  $\mathbf{c}$  is a constant for fixed space and variable space for the instance  $I$ .

We are concerned about only  $S_p(I)$ , but **not  $\mathbf{c}$ . Why?**

Because we usually **compare** the algorithms of the programs.

# Performance Analysis

- Space complexity:  $S(P) = c + S_p(I)$
- Example:  $S_{\text{sum}}(n) = ?$

Program sum

```
float sum(float list[], int n) {  
    float total = 0;  
    for (int i=0; i<n; i++)  
        total += list[i];  
    return total;  
}
```

$S_{\text{sum}}(n) = 0$  since the C/C++ passes list[] by its address.

# Performance Analysis

- Space complexity:  $S(P) = c + S_p(I)$
- Example:  $S_{\text{rsum}}(n) = ?$

Program rsum

```
float rsum(float list[], int n) {  
    if (n)  
        return rsum(list, n-1) + list[n-1];  
    return 0;  
}
```

Program rsum

```
float rsum(float list[], int n) {  
    if (n)  
        return rsum(list, n-1) + list[n];  
    return 0;  
}
```

# Performance Analysis

- Space complexity:  $S(P) = c + S_p(I)$
- Example:  $S_{\text{rsum}}(n) = ?$

Program rsum

```
float rsum(float list[], int n) {  
    if (n)  
        return rsum(list, n-1) + list[n-1];  
    return 0;  
}
```

The variable space requirement are for **two** parameters and **one** return address are saved in the system stack **per recursive call**:

$$\text{sizeof}(n) + \text{list[]} \text{ address} + \text{return address} = 12$$

← assuming 32 bit address

# Performance Analysis

- Space complexity:  $S(P) = c + S_p(I)$
- Example:  $S_{\text{rsum}}(n=\text{MAX\_SIZE}) = ?$

Program rsum

```
float rsum(float list[], int n) {  
    if (n)  
        return rsum(list, n-1) + list[n-1];  
    return 0;  
}
```

The variable space requirement are for **two** parameters and **one** return address are saved in the system stack **per recursive call**:

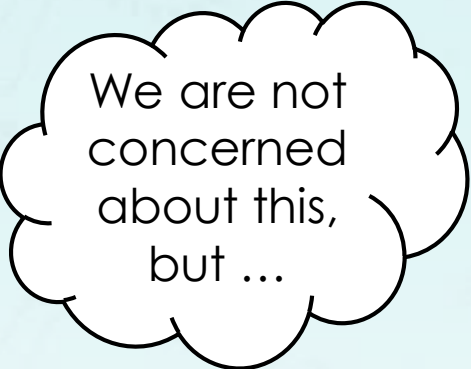
$$\text{sizeof}(n) + \text{list[]} \text{ address} + \text{return address} = 12$$

$$S_{\text{rsum}}(n) = 12 * n$$



# Performance Analysis

- **Time complexity:** The time taken by the program P:
  - $T(P) = \text{compile time } c + \text{execution time } T_p(n)$
- Similarly, we are concerned about only  $T_p(n)$ , but not  $c$ .
- **Example:**  $T_p(n) = c_a \text{ADD}(n) + c_s \text{SUB}(n) + c_l \text{LDA}(n) + c_{st} \text{STA}(n)$ 
  - where  $n$  – number of execution,  $c$  for constant time for operation
- **Program step:** a meaningful program segment whose execution time is independent of the instance characteristics.
- **Example:**
  - $a = 2;$
  - $a = 2 * b + 3 * c/d - e + f/g/a/b/c;$



We are not concerned about this, but ...

⇒ 1 step!!

⇒ 1 step!!



# Performance Analysis

- **Example:** How many **program steps** required?

Program	sum	$2n+3$
<pre>float sum(float list[], int n) {     float total = 0;     for (int i=0; i&lt;n; i++)         total += list[i];     return total; }</pre>		 1 n+1 n 1

# Performance Analysis

- **Example:** How many **program steps** required?

Program rsum	$2n+2$
<pre>float rsum(float list[], int n) {     if (n)         return rsum(list, n-1) + list[n-1];     return 0; }</pre>	$n+1$ $n$ $1$

# Performance Analysis

## ■ Comparison:

### Program sum

```
float sum(float list[], int n) {  
    float total = 0;  
    for (int i=0; i<n; i++)  
        total += list[i];  
    return total;  
}
```

### Program rsum

```
float rsum(float list[], int n) {  
    if (n)  
        return rsum(list, n-1) + list[n-1];  
    return 0;  
}
```

$$2n + 3 > 2n + 2$$

$$\text{sum} > \text{rsum}$$

$$T_{\text{iterative}} > T_{\text{recursive}}$$

# Performance Analysis

- **Example:** How many **program steps** required?

Program	sum of matrix
<pre>void add(int a[][MAX_SIZE], int b[][MAX_SIZE],         int c[][MAX_SIZE], int rows, int cols) {     for(int i=0; i&lt;rows; i++)         for(int j=0; j&lt;cols; j++)             c[i][j] = a[i][j] + b[i][j]; }</pre>	<pre>rows + 1 rows * (cols+1) rows * cols</pre>

step count = **2 rows\*cols + 2 rows + 1**

## Step Count Example 1:

- What is **the exact number of times** `sum++` executed?

	Step count
<pre>int sum = 0; for (int i = 1; i &lt;= n*n; i++)     for (int j = 1; j &lt;= i; j++)         sum++;</pre>	<pre>1 n * n + 1 2 + 3 + ... + n*n+1 ?</pre>

### Useful formulas:

$$1 + 2 + 3 + \dots + N = N(N+1)/2$$

$$1 + 2 + 4 + 8 + \dots + 2^n = 2^{n+1} - 1$$

## Step Count Example 2:

- What is **the exact number of times** `sum++` executed?

	Step count
<pre>int sum = 0; for (int i = 1; i &lt;= n; i++)     for (int j = n; j &gt;= i; j--)         sum++;</pre>	<pre>1 n + 1 (n+1) + (n) + (n-1) + ... + 2 ?</pre>

### Useful formulas:

$$1 + 2 + 3 + \dots + N = N(N+1)/2$$

$$1 + 2 + 4 + 8 + \dots + 2^n = 2^{n+1} - 1$$

## Step Count Example 3:

- What is **the exact number of times** `sum++` executed?

	Step count
<pre>int sum = 0; while (n &gt;= 1) {     sum++;     n /= 2; }</pre>	$n / 2^k = 1$

We have to find the smallest  $k$  such that  $n / 2^k = 1$

$$n / 2^k = 1$$

$$n = 2^k$$

$$\log(n) = \log(2^k)$$

$$\log(n) = k$$



## Step Count Example 4:

---

- Compute the following series:

a)  $1 + 2 + 3 + \dots + 9 + 10 =$

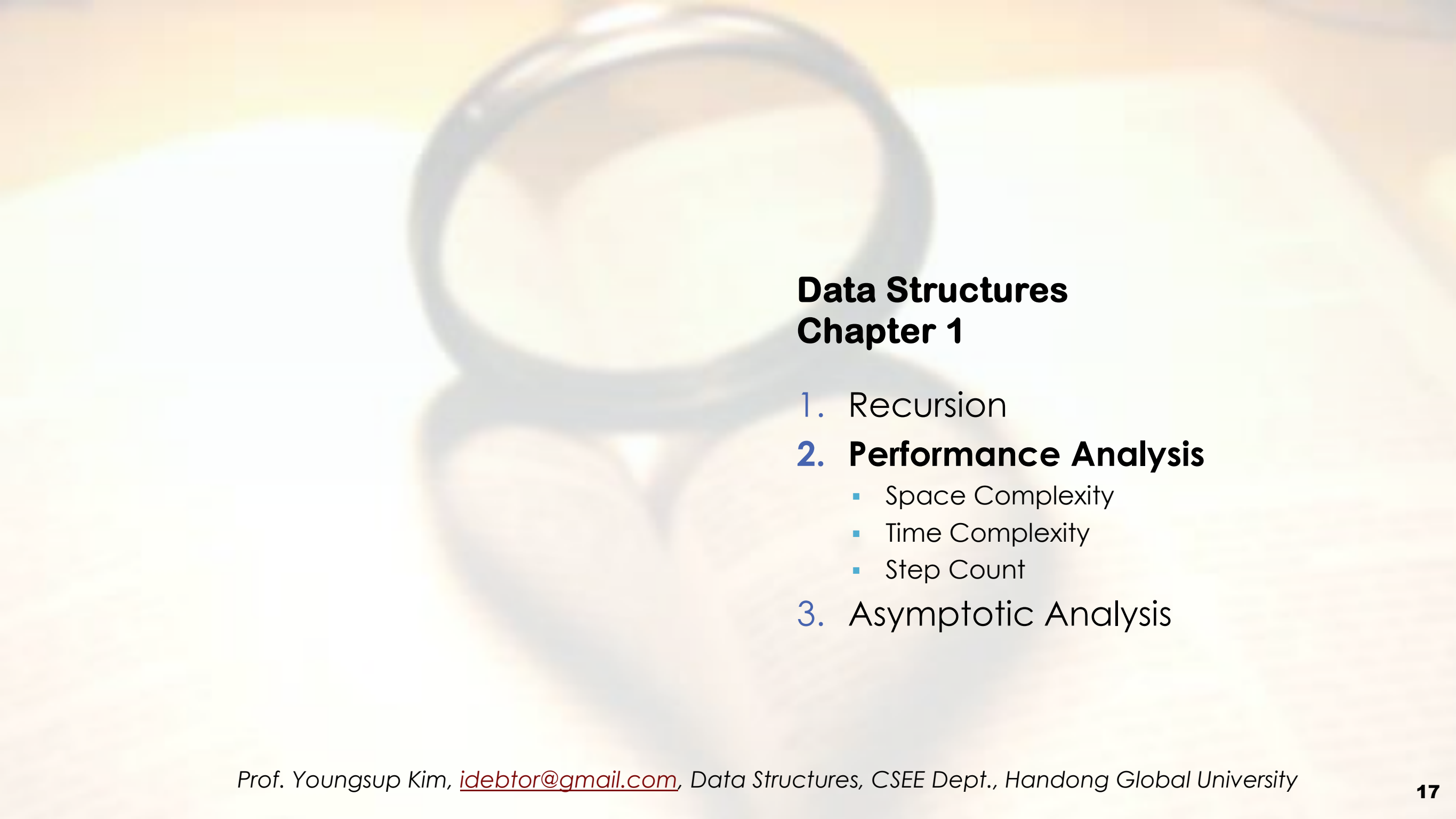
b)  $1 + 2 + 3 + \dots + (N - 1) + N =$

c)  $1 + 2 + 4 + \dots + 16 =$

### Useful formulas:

$$1 + 2 + 3 + \dots + N = N(N+1)/2$$

$$1 + 2 + 4 + 8 + \dots + 2^n = 2^{n+1} - 1$$

A pair of glasses with a dark frame and light-colored lenses is resting on a piece of white paper. The background is a soft, out-of-focus yellow and orange gradient.

# Data Structures

## Chapter 1

1. Recursion

### 2. Performance Analysis

- Space Complexity
- Time Complexity
- Step Count

3. Asymptotic Analysis