Exercise:
Analysis of Iterative (Nonrecursive) Algorithm

Example

```
A( )
{
    for (i = 1 to n){
        printf ("Salam");
}}
```

```
A( )
{
    int i, j;
    for (i = 1 to n){
        for (j = 1 to n){
            printf ("Salam");
}}}
```

Analysis of Nonrecursive Algos

Example: Find the value of the **largest element** in a list of n real numbers.

Algorithm MaxElement($A[0..n-1]$)

// Input: An array $A[0..n-1]$

//Output: The value the max element in A

$max \leftarrow A[0]$

for $i \leftarrow 1$ **to** $n-1$ **do**

  if $A[i] > max$

    $max \leftarrow A[i]$

return max

Two operations in the loop

Basic operation: the comparison (it is executed in each repetition of the loop!)

Analysis of Nonrecursive Algos

Analysis:

- n is the number of times the comparison is executed.
- The **number of comparisons** will be the **same**.
- **No need to distinguish** among the worst, average and best cases.

$$C(n) = \sum_{i=1}^{n-1} 1 = n - 1 \in \Theta(n).$$

General Plan for Analysis

- Decide on parameter(s) indicating **input size(s)**
- Identify algorithm's **basic operation(s)**
- Determine **worst**, **average**, and **best** case efficiencies
- Set up a **sum** for the number of times the basic operation(s) is (are) executed
- **Simplify** the sum using standard formulas and rules

Analysis of Nonrecursive Algos

Example: The element uniqueness problem: check if all elements in a given array of n elements are distinct.

Algorithm UniqueElement($A[0..n-1]$)

// Input: An array $A[0..n-1]$

//Output: Return "True" if all elements in A are distinct & "False" otherwise

for $i \leftarrow 0$ **to** $n-2$ **do**

  for $j \leftarrow i+1$ **to** $n-1$ **do**

    if $A[i] = A[j]$ **return** False

return True

Analysis of Nonrecursive Algos

- **The input size:** n, the number of the elements in an array
- **The basic operation:** the comparison operation in the innermost loop
- The number of comparisons depends not only on n but also on if there are **equal elements in the array**, if there are, which **array positions** they occupy.
- **The worst-case inputs:**
  - arrays with no equal elements
  - arrays with the last two equal elements only

Analysis of Nonrecursive Algos

- **Worst-case analysis:**

$$C_{worst}(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1$$

$$= \sum_{i=0}^{n-2} [(n-1) - ((i+1)-1)]$$

$$= \sum_{i=0}^{n-2} (n-1-i) = (n-1) + (n-2) + \cdots + 1$$

$$= \frac{n(n-1)}{2} \approx \frac{1}{2} n^2 \in \Theta(n^2)$$

Exercise

a) 1+2+3+4+….+100 = ? (Generalize: if 1+2+3+4+….+n ?)

b) 10 + 20 + 30 + 40 + … + 1000 = ? (Generalize)

c) $(n-1)+(n-2)+(n-3)+\cdots+1=?$

d) 1 + 3 + 5+ 7 +9 + … + 999 = ? (Generalize)

e) $1+4+9+16+25=?$

f) $1+4+9+16+\cdots+100=?$

Exercise

a) $\sum_{i=3}^{n+1} 1$

b) $\sum_{i=3}^{n+1} i$

c) $\sum_{i=1}^{n} \sum_{j=1}^{n} ij$

Analysis of Nonrecursive Algos

Example: The element uniqueness problem: check if all elements in a given array of n elements are distinct.

Algorithm UniqueElement($A[0..n-1]$)

// Input: An array $A[0..n-1]$

//Output: Return "True" if all elements in A are distinct & "False" otherwise

for $i \leftarrow 0$ **to** $n-2$ **do**

  for $j \leftarrow i+1$ **to** $n-1$ **do**

    if $A[i] = A[j]$ **return** False

return True

Analysis of Nonrecursive Algos

- **The input size:** n, the number of the elements in an array
- **The basic operation:** the comparison operation in the innermost loop
- The number of comparisons depends not only on n but also on if there are **equal elements in the array**, if there are, which **array positions** they occupy.
- **The worst-case inputs:**
  - arrays with no equal elements
  - arrays with the last two equal elements only

Analysis of Nonrecursive Algos

- **Worst-case analysis:**

$$C_{worst}(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1$$

$$= \sum_{i=0}^{n-2} [(n-1) - ((i+1)-1)]$$

$$= \sum_{i=0}^{n-2} (n-1-i) = (n-1) + (n-2) + \cdots + 1$$

$$= \frac{n(n-1)}{2} \approx \frac{1}{2} n^2 \in \Theta(n^2)$$

```
A( )
{
    int i = 1, s = 1;
    while (s ≤ n){
        i + +;
        s = s + i;
        printf ("Salam");
}}
```

Exercise 1

Exercise 2 & 3

```
A( )
{
    for (i = 1; i^2 ≤ n; i + +){
        printf ("Salam");
}
```

```
A( )
{
    int i, j, k, n;
     for (i = 1; i ≤ n; i + +){
         for (j = 1; j ≤ i;  j + +){
             for (k = 1; k ≤ 100; k + +){
                 printf ("Salam");
    }}}
}
```

Exercise: Answ

```
A( )
{
    for (i = 1; i^2 ≤ n; i++){
        printf ("Salam");
}
```

$O(\sqrt{n})$

```
A( )
{
    int i, j, k, n;
    for (i = 1; i ≤ n; i++){
        for (j = 1; j ≤ i; j++){
            for (k = 1; k ≤ 100; k++){
                printf ("Salam");
    }}}
}
```

$O(n^2)$

Exercise 4

```
A( )
{
    int i, j, k, n;
     for (i = 1; i ≤ n; i + +){
         for (j = 1; j ≤ i^2; j + +){
             for (k = 1; k ≤ n/2; k + +){
                 printf ("Salam");
    }}}
}
```

Exercise: Ans

- $O(n^4)$

```
A( )
{
    int i, j, k, n;
     for (i = 1; i ≤ n; i + +){
         for (j = 1; j ≤ i^2; j + +){
             for (k = 1; k ≤ n/2; k + +){
                 printf ("Salam");
   }}}
}
```