

Algorithms

Dr. Khalil Yousef

Lecture 1: Introduction

Reading Assignment:

Read Chapters 1 and 2 of *the book*

Course Learning Outcomes

- Use algorithm design methods,.
- Show *how time and space complexities can be traded-off*
- Analyze numerical computations algorithms

Designing Algorithms: Techniques/Strategies

- Brute force
- Divide and conquer
- **Decrease and conquer**
- Transform and conquer
- Space and time tradeoffs
- Greedy approach
- Dynamic programming
- Iterative improvement
- Backtracking
- Branch and bound

Decrease-and-Conquer

1. Reduce problem instance to smaller instance of the same problem
 2. Solve smaller instance
 3. Extend solution of smaller instance to obtain solution to original instance
- Can be implemented either top-down or bottom-up
 - Also referred to as *inductive* or incremental approach

3 Types of Decrease and Conquer

- Decrease by a constant (usually by 1):
 - insertion sort
 - topological sorting
 - algorithms for generating permutations, subsets
- Decrease by a constant factor (usually by half)
 - binary search and bisection method
 - exponentiation by squaring
 - multiplication à la russe
- Variable-size decrease
 - Euclid's algorithm
 - selection by partition
 - Nim-like games

Recall: Sorting Problem

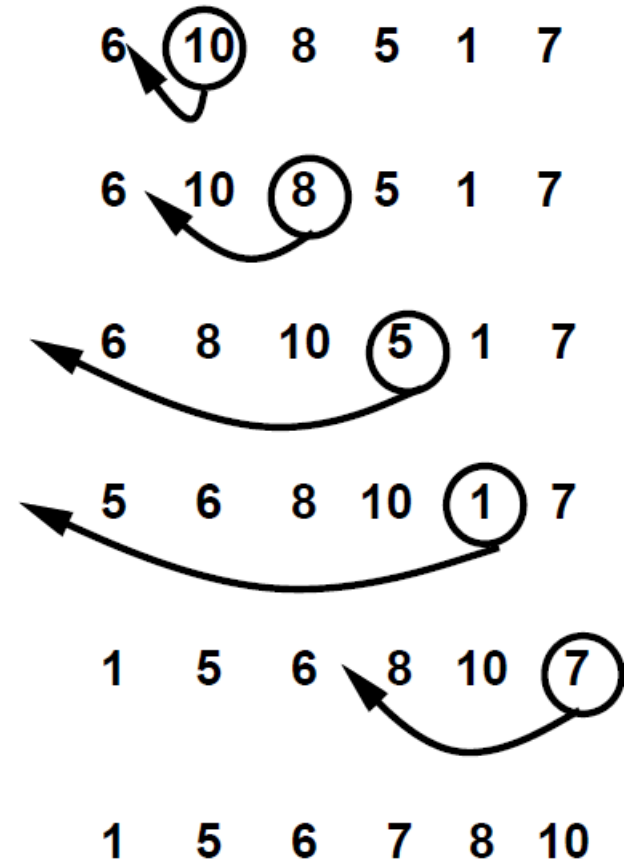
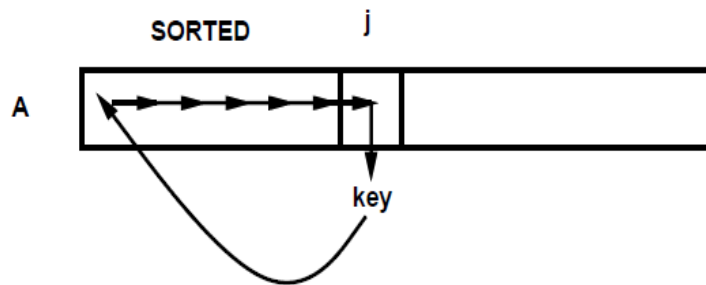
- **The Sorting Problem (non-decreasing):**
 - **Input:** a sequence of n numbers $\langle a_1, a_2, \dots, a_n \rangle$
 - **Output:** a permutation of the input sequence $\langle a_1', a_2', \dots, a_n' \rangle$ such that $a_1' \leq a_2' \dots \leq a_n'$
 - **Basic Operation:** Comparison of two elements
- **Recall: How to Characterize an Algorithm**
 - Use the resources in terms of:
 - Time Complexity
 - Space Complexity
 - Goal: Order of Growth (Asymptotic Analysis)
 - Find the running time as a function of the input size in terms of the numbers of basic operations

INSERTION-SORT Example:

<6, 10, 8, 5, 1, 7>

INSERTION-SORT(A)

1. **for** $j \leftarrow 2$ to $\text{length}[A]$
2. **do** $\text{key} \leftarrow A[j]$
3. \triangleright Insert $A[j]$ into the sorted sequence $A[1..j-1]$
4. $i \leftarrow j - 1$
5. **while** $i > 0$ and $A[i] > \text{key}$
6. **do** $A[i+1] \leftarrow A[i]$
7. $i \leftarrow i - 1$
8. $A[i+1] \leftarrow \text{key}$



Pseudocode Notation

```
INSERTION-SORT(A)
1. for  $j \leftarrow 2$  to  $length[A]$ 
2.   do  $key \leftarrow A[j]$ 
3.      $\triangleright$  Insert  $A[j]$  into the sorted sequence  $A[1..j-1]$ 
4.      $i \leftarrow j - 1$ 
5.     while  $i > 0$  and  $A[i] > key$ 
6.       do  $A[i+1] \leftarrow A[i]$ 
7.          $i \leftarrow i - 1$ 
8.      $A[i+1] \leftarrow key$ 
```

- Indentation reflects block structure.
- Looping and conditional constructs have Pascal semantics.
- \triangleright is used for comments.
- \leftarrow is used for assignment.
- Variables are local unless otherwise indicated.
- Array elements are accessed as in Pascal, and we can specify subranges using: $A[i..j]$. Arrays are compound data types with a length attribute accessed using $length[array_name]$.
- To access the value of a field in a compound data type, use $field_name[compound]$.
- Parameters are passed by value.
- Omit error handling used in real programs.

An Example: Insertion Sort

(Note the change in the indexing of the loops compared to the previous code)

```
InsertionSort(A, n) {  
    for i = 2 to n {  
        key = A[i]  
        j = i - 1;  
        while (j > 0) and (A[j] > key) {  
            A[j+1] = A[j]  
            j = j - 1  
        }  
        A[j+1] = key  
    }  
}
```

Insertion Sort

```
InsertionSort(A, n) {
```

```
  for i = 2 to n {
```

```
    key = A[i]
```

```
    j = i - 1;
```

```
    while (j > 0) and (A[j] > key) {
```

```
      A[j+1] = A[j]
```

```
      j = j - 1
```

```
    }
```

```
    A[j+1] = key
```

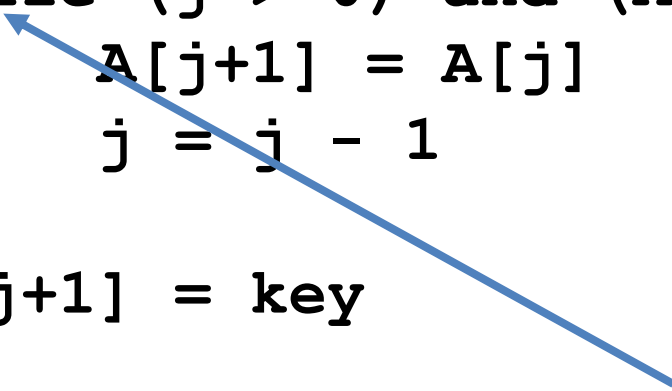
```
  }
```

```
}
```

What is the precondition for this loop?

Insertion Sort: Basic Operation

```
InsertionSort(A, n) {  
    for i = 2 to n {  
        key = A[i]  
        j = i - 1;  
        while (j > 0) and (A[j] > key) {  
            A[j+1] = A[j]  
            j = j - 1  
        }  
        A[j+1] = key  
    }  
}
```



How many times will this loop execute?

An Example: Insertion Sort

| | | | |
|----|----|----|----|
| 30 | 10 | 40 | 20 |
| 1 | 2 | 3 | 4 |

| | | |
|--------------------|----------------------|-------------------|
| $i = \emptyset$ | $j = \emptyset$ | $key = \emptyset$ |
| $A[j] = \emptyset$ | $A[j+1] = \emptyset$ | |




```
InsertionSort(A, n) {  
    for i = 2 to n {  
        key = A[i]  
        j = i - 1;  
        while (j > 0) and (A[j] > key) {  
            A[j+1] = A[j]  
            j = j - 1  
        }  
        A[j+1] = key  
    }  
}
```

An Example: Insertion Sort

| | | | |
|----|----|----|----|
| 30 | 10 | 40 | 20 |
| 1 | 2 | 3 | 4 |

| | | |
|-------------|---------------|-------------------|
| $i = 2$ | $j = 1$ | $\text{key} = 10$ |
| $A[j] = 30$ | $A[j+1] = 10$ | |

```
InsertionSort(A, n) {  
    for i = 2 to n {  
        key = A[i]  
        j = i - 1;  
        while (j > 0) and (A[j] > key) {  
            A[j+1] = A[j]  
            j = j - 1  
        }  
        A[j+1] = key  
    }  
}
```



An Example: Insertion Sort

| | | | |
|----|----|----|----|
| 30 | 30 | 40 | 20 |
| 1 | 2 | 3 | 4 |

| | | |
|-------------|---------------|-------------------|
| $i = 2$ | $j = 1$ | $\text{key} = 10$ |
| $A[j] = 30$ | $A[j+1] = 30$ | |

```
InsertionSort(A, n) {  
    for i = 2 to n {  
        key = A[i]  
        j = i - 1;  
        while (j > 0) and (A[j] > key) {  
            A[j+1] = A[j]  
            j = j - 1  
        }  
        A[j+1] = key  
    }  
}
```




An Example: Insertion Sort

| | | | |
|----|----|----|----|
| 30 | 30 | 40 | 20 |
| 1 | 2 | 3 | 4 |

| | | |
|-------------|---------------|------------|
| $i = 2$ | $j = 1$ | $key = 10$ |
| $A[j] = 30$ | $A[j+1] = 30$ | |

```
InsertionSort(A, n) {  
    for i = 2 to n {  
        key = A[i]  
        j = i - 1;  
        while (j > 0) and (A[j] > key) {  
            A[j+1] = A[j]  
            j = j - 1  
        }  
        A[j+1] = key  
    }  
}
```



An Example: Insertion Sort

| | | | |
|----|----|----|----|
| 30 | 30 | 40 | 20 |
| 1 | 2 | 3 | 4 |

| | | |
|--------------------|---------------|-------------------|
| $i = 2$ | $j = 0$ | $\text{key} = 10$ |
| $A[j] = \emptyset$ | $A[j+1] = 30$ | |

```
InsertionSort(A, n) {  
    for i = 2 to n {  
        key = A[i]  
        j = i - 1;  
        while (j > 0) and (A[j] > key) {  
            A[j+1] = A[j]  
            j = j - 1  
        }  
        A[j+1] = key  
    }  
}
```




An Example: Insertion Sort

| | | | |
|----|----|----|----|
| 30 | 30 | 40 | 20 |
| 1 | 2 | 3 | 4 |

| | | |
|--------------------|---------------|-------------------|
| $i = 2$ | $j = 0$ | $\text{key} = 10$ |
| $A[j] = \emptyset$ | $A[j+1] = 30$ | |

```
InsertionSort(A, n) {  
    for i = 2 to n {  
        key = A[i]  
        j = i - 1;  
        while (j > 0) and (A[j] > key) {  
            A[j+1] = A[j]  
            j = j - 1  
        }  
        A[j+1] = key  
    }  
}
```




An Example: Insertion Sort

| | | | |
|----|----|----|----|
| 10 | 30 | 40 | 20 |
| 1 | 2 | 3 | 4 |

| | | |
|--------------------|---------------|-------------------|
| $i = 2$ | $j = 0$ | $\text{key} = 10$ |
| $A[j] = \emptyset$ | $A[j+1] = 10$ | |

```
InsertionSort(A, n) {  
    for i = 2 to n {  
        key = A[i]  
        j = i - 1;  
        while (j > 0) and (A[j] > key) {  
            A[j+1] = A[j]  
            j = j - 1  
        }  
        A[j+1] = key  
    }  
}
```



An Example: Insertion Sort

| | | | |
|----|----|----|----|
| 10 | 30 | 40 | 20 |
| 1 | 2 | 3 | 4 |

| | | |
|--------------------|---------------|-------------------|
| $i = 3$ | $j = 0$ | $\text{key} = 10$ |
| $A[j] = \emptyset$ | $A[j+1] = 10$ | |



```
InsertionSort(A, n) {  
    for i = 2 to n {  
        key = A[i]  
        j = i - 1;  
        while (j > 0) and (A[j] > key) {  
            A[j+1] = A[j]  
            j = j - 1  
        }  
        A[j+1] = key  
    }  
}
```

An Example: Insertion Sort

| | | | |
|----|----|----|----|
| 10 | 30 | 40 | 20 |
| 1 | 2 | 3 | 4 |

| | | |
|--------------------|---------------|-------------------|
| $i = 3$ | $j = 0$ | $\text{key} = 40$ |
| $A[j] = \emptyset$ | $A[j+1] = 10$ | |



```
InsertionSort(A, n) {  
    for i = 2 to n {  
        key = A[i]  
        j = i - 1;  
        while (j > 0) and (A[j] > key) {  
            A[j+1] = A[j]  
            j = j - 1  
        }  
        A[j+1] = key  
    }  
}
```

An Example: Insertion Sort

| | | | |
|----|----|----|----|
| 10 | 30 | 40 | 20 |
| 1 | 2 | 3 | 4 |

| | | |
|--------------------|---------------|-------------------|
| $i = 3$ | $j = 0$ | $\text{key} = 40$ |
| $A[j] = \emptyset$ | $A[j+1] = 10$ | |



```
InsertionSort(A, n) {  
    for i = 2 to n {  
        key = A[i]  
        j = i - 1;  
        while (j > 0) and (A[j] > key) {  
            A[j+1] = A[j]  
            j = j - 1  
        }  
        A[j+1] = key  
    }  
}
```

An Example: Insertion Sort

| | | | |
|----|----|----|----|
| 10 | 30 | 40 | 20 |
| 1 | 2 | 3 | 4 |

| | | |
|-------------|---------------|-------------------|
| $i = 3$ | $j = 2$ | $\text{key} = 40$ |
| $A[j] = 30$ | $A[j+1] = 40$ | |




```
InsertionSort(A, n) {  
    for i = 2 to n {  
        key = A[i]  
        j = i - 1;  
        while (j > 0) and (A[j] > key) {  
            A[j+1] = A[j]  
            j = j - 1  
        }  
        A[j+1] = key  
    }  
}
```

An Example: Insertion Sort

| | | | |
|----|----|----|----|
| 10 | 30 | 40 | 20 |
| 1 | 2 | 3 | 4 |

| | | |
|-------------|---------------|-------------------|
| $i = 3$ | $j = 2$ | $\text{key} = 40$ |
| $A[j] = 30$ | $A[j+1] = 40$ | |

```
InsertionSort(A, n) {  
    for i = 2 to n {  
        key = A[i]  
        j = i - 1;  
        while (j > 0) and (A[j] > key) {  
            A[j+1] = A[j]  
            j = j - 1  
        }  
        A[j+1] = key  
    }  
}
```




An Example: Insertion Sort

| | | | |
|----|----|----|----|
| 10 | 30 | 40 | 20 |
| 1 | 2 | 3 | 4 |

| | | |
|-------------|---------------|-------------------|
| $i = 3$ | $j = 2$ | $\text{key} = 40$ |
| $A[j] = 30$ | $A[j+1] = 40$ | |

```
InsertionSort(A, n) {  
    for i = 2 to n {  
        key = A[i]  
        j = i - 1;  
        while (j > 0) and (A[j] > key) {  
            A[j+1] = A[j]  
            j = j - 1  
        }  
        A[j+1] = key  
    }  
}
```



An Example: Insertion Sort

| | | | |
|----|----|----|----|
| 10 | 30 | 40 | 20 |
| 1 | 2 | 3 | 4 |

| | | |
|-------------|---------------|------------|
| $i = 4$ | $j = 2$ | $key = 40$ |
| $A[j] = 30$ | $A[j+1] = 40$ | |



```
InsertionSort(A, n) {  
    for i = 2 to n {  
        key = A[i]  
        j = i - 1;  
        while (j > 0) and (A[j] > key) {  
            A[j+1] = A[j]  
            j = j - 1  
        }  
        A[j+1] = key  
    }  
}
```

An Example: Insertion Sort

| | | | |
|----|----|----|----|
| 10 | 30 | 40 | 20 |
| 1 | 2 | 3 | 4 |

| | | |
|-------------|---------------|------------|
| $i = 4$ | $j = 2$ | $key = 20$ |
| $A[j] = 30$ | $A[j+1] = 40$ | |



```
InsertionSort(A, n) {  
    for i = 2 to n {  
        key = A[i]  
        j = i - 1;  
        while (j > 0) and (A[j] > key) {  
            A[j+1] = A[j]  
            j = j - 1  
        }  
        A[j+1] = key  
    }  
}
```

An Example: Insertion Sort

| | | | |
|----|----|----|----|
| 10 | 30 | 40 | 20 |
| 1 | 2 | 3 | 4 |

| | | |
|-------------|---------------|-------------------|
| $i = 4$ | $j = 2$ | $\text{key} = 20$ |
| $A[j] = 30$ | $A[j+1] = 40$ | |



```
InsertionSort(A, n) {  
    for i = 2 to n {  
        key = A[i]  
        j = i - 1;  
        while (j > 0) and (A[j] > key) {  
            A[j+1] = A[j]  
            j = j - 1  
        }  
        A[j+1] = key  
    }  
}
```

An Example: Insertion Sort

| | | | |
|----|----|----|----|
| 10 | 30 | 40 | 20 |
| 1 | 2 | 3 | 4 |

| | | |
|-------------|---------------|------------|
| $i = 4$ | $j = 3$ | $key = 20$ |
| $A[j] = 40$ | $A[j+1] = 20$ | |



```
InsertionSort(A, n) {  
    for i = 2 to n {  
        key = A[i]  
        j = i - 1;  
        while (j > 0) and (A[j] > key) {  
            A[j+1] = A[j]  
            j = j - 1  
        }  
        A[j+1] = key  
    }  
}
```

An Example: Insertion Sort

| | | | |
|----|----|----|----|
| 10 | 30 | 40 | 20 |
| 1 | 2 | 3 | 4 |

| | | |
|-------------|---------------|------------|
| $i = 4$ | $j = 3$ | $key = 20$ |
| $A[j] = 40$ | $A[j+1] = 20$ | |

```
InsertionSort(A, n) {  
    for i = 2 to n {  
        key = A[i]  
        j = i - 1;  
        while (j > 0) and (A[j] > key) {  
            A[j+1] = A[j]  
            j = j - 1  
        }  
        A[j+1] = key  
    }  
}
```



An Example: Insertion Sort

| | | | |
|----|----|----|----|
| 10 | 30 | 40 | 40 |
| 1 | 2 | 3 | 4 |

| | | |
|-------------|---------------|------------|
| $i = 4$ | $j = 3$ | $key = 20$ |
| $A[j] = 40$ | $A[j+1] = 40$ | |

```
InsertionSort(A, n) {  
    for i = 2 to n {  
        key = A[i]  
        j = i - 1;  
        while (j > 0) and (A[j] > key) {  
            A[j+1] = A[j]  
            j = j - 1  
        }  
        A[j+1] = key  
    }  
}
```



An Example: Insertion Sort

| | | | |
|----|----|----|----|
| 10 | 30 | 40 | 40 |
| 1 | 2 | 3 | 4 |

| | | |
|-------------|---------------|------------|
| $i = 4$ | $j = 3$ | $key = 20$ |
| $A[j] = 40$ | $A[j+1] = 40$ | |

```
InsertionSort(A, n) {  
  for i = 2 to n {  
    key = A[i]  
    j = i - 1;  
    while (j > 0) and (A[j] > key) {  
      A[j+1] = A[j]  
      j = j - 1  
    }  
    A[j+1] = key  
  }  
}
```




An Example: Insertion Sort

| | | | |
|----|----|----|----|
| 10 | 30 | 40 | 40 |
| 1 | 2 | 3 | 4 |

| | | |
|-------------|---------------|------------|
| $i = 4$ | $j = 3$ | $key = 20$ |
| $A[j] = 40$ | $A[j+1] = 40$ | |

```
InsertionSort(A, n) {  
  for i = 2 to n {  
    key = A[i]  
    j = i - 1;  
    while (j > 0) and (A[j] > key) {  
      A[j+1] = A[j]  
      j = j - 1  
    }  
    A[j+1] = key  
  }  
}
```



An Example: Insertion Sort

| | | | |
|----|----|----|----|
| 10 | 30 | 40 | 40 |
| 1 | 2 | 3 | 4 |

| | | |
|-------------|---------------|------------|
| $i = 4$ | $j = 2$ | $key = 20$ |
| $A[j] = 30$ | $A[j+1] = 40$ | |

```
InsertionSort(A, n) {  
  for i = 2 to n {  
    key = A[i]  
    j = i - 1;  
    while (j > 0) and (A[j] > key) {  
      A[j+1] = A[j]  
      j = j - 1  
    }  
    A[j+1] = key  
  }  
}
```




An Example: Insertion Sort

| | | | |
|----|----|----|----|
| 10 | 30 | 40 | 40 |
| 1 | 2 | 3 | 4 |

| | | |
|-------------|---------------|------------|
| $i = 4$ | $j = 2$ | $key = 20$ |
| $A[j] = 30$ | $A[j+1] = 40$ | |

```
InsertionSort(A, n) {  
    for i = 2 to n {  
        key = A[i]  
        j = i - 1;  
        while (j > 0) and (A[j] > key) {  
            A[j+1] = A[j]  
            j = j - 1  
        }  
        A[j+1] = key  
    }  
}
```



An Example: Insertion Sort

| | | | |
|----|----|----|----|
| 10 | 30 | 30 | 40 |
| 1 | 2 | 3 | 4 |

| | | |
|-------------|---------------|-------------------|
| $i = 4$ | $j = 2$ | $\text{key} = 20$ |
| $A[j] = 30$ | $A[j+1] = 30$ | |

```
InsertionSort(A, n) {  
    for i = 2 to n {  
        key = A[i]  
        j = i - 1;  
        while (j > 0) and (A[j] > key) {  
            A[j+1] = A[j]  
            j = j - 1  
        }  
        A[j+1] = key  
    }  
}
```



An Example: Insertion Sort

| | | | |
|----|----|----|----|
| 10 | 30 | 30 | 40 |
| 1 | 2 | 3 | 4 |

| | | |
|-------------|---------------|------------|
| $i = 4$ | $j = 2$ | $key = 20$ |
| $A[j] = 30$ | $A[j+1] = 30$ | |

```
InsertionSort(A, n) {  
    for i = 2 to n {  
        key = A[i]  
        j = i - 1;  
        while (j > 0) and (A[j] > key) {  
            A[j+1] = A[j]  
            j = j - 1  
        }  
        A[j+1] = key  
    }  
}
```



An Example: Insertion Sort

| | | | |
|----|----|----|----|
| 10 | 30 | 30 | 40 |
| 1 | 2 | 3 | 4 |

| | | |
|-------------|---------------|------------|
| $i = 4$ | $j = 1$ | $key = 20$ |
| $A[j] = 10$ | $A[j+1] = 30$ | |

```
InsertionSort(A, n) {  
  for i = 2 to n {  
    key = A[i]  
    j = i - 1;  
    while (j > 0) and (A[j] > key) {  
      A[j+1] = A[j]  
      j = j - 1  
    }  
    A[j+1] = key  
  }  
}
```




An Example: Insertion Sort

| | | | |
|----|----|----|----|
| 10 | 30 | 30 | 40 |
| 1 | 2 | 3 | 4 |

| | | |
|-------------|---------------|------------|
| $i = 4$ | $j = 1$ | $key = 20$ |
| $A[j] = 10$ | $A[j+1] = 30$ | |

```
InsertionSort(A, n) {  
  for i = 2 to n {  
    key = A[i]  
    j = i - 1;  
    while (j > 0) and (A[j] > key) {  
      A[j+1] = A[j]  
      j = j - 1  
    }  
    A[j+1] = key  
  }  
}
```




An Example: Insertion Sort

| | | | |
|----|----|----|----|
| 10 | 20 | 30 | 40 |
| 1 | 2 | 3 | 4 |

| | | |
|-------------|---------------|------------|
| $i = 4$ | $j = 1$ | $key = 20$ |
| $A[j] = 10$ | $A[j+1] = 20$ | |

```
InsertionSort(A, n) {  
    for i = 2 to n {  
        key = A[i]  
        j = i - 1;  
        while (j > 0) and (A[j] > key) {  
            A[j+1] = A[j]  
            j = j - 1  
        }  
        A[j+1] = key  
    }  
}
```



An Example: Insertion Sort

| | | | |
|----|----|----|----|
| 10 | 20 | 30 | 40 |
| 1 | 2 | 3 | 4 |

| | | |
|-------------|---------------|------------|
| $i = 4$ | $j = 1$ | $key = 20$ |
| $A[j] = 10$ | $A[j+1] = 20$ | |

```
InsertionSort(A, n) {  
    for i = 2 to n {  
        key = A[i]  
        j = i - 1;  
        while (j > 0) and (A[j] > key) {  
            A[j+1] = A[j]  
            j = j - 1  
        }  
        A[j+1] = key  
    }  
}
```

Done!

Analyzing the INSERTION-SORT Algorithm

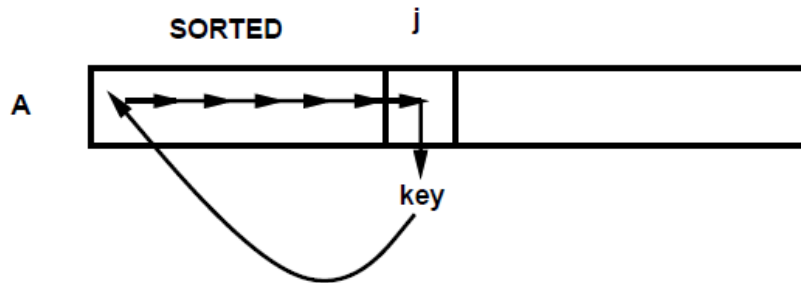
(Similar to the Book)

INSERTION-SORT(A)

```
1. for  $j \leftarrow 2$  to  $length[A]$ 
2.   do  $key \leftarrow A[j]$ 
3.     ▷ Insert  $A[j]$  into the sorted sequence  $A[1..j-1]$ 
4.      $i \leftarrow j-1$ 
5.     while  $i > 0$  and  $A[i] > key$ 
6.       do  $A[i+1] \leftarrow A[i]$ 
7.        $i \leftarrow i-1$ 
8.      $A[i+1] \leftarrow key$ 
```

What is the precondition for this loop?

How many times will this loop execute?

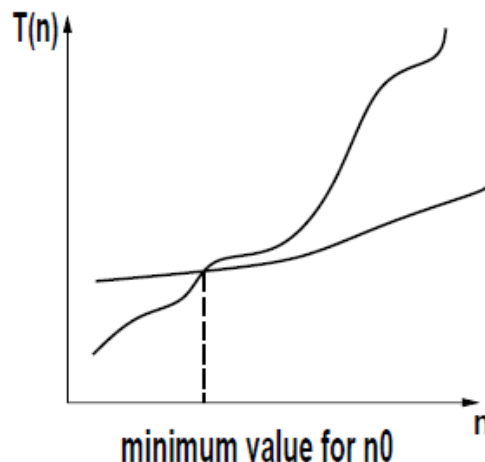


An Incremental Algorithm



Recall: How to Characterize an Algorithm

- Find the running time as a function of the input size.
 - Asymptotic Performance
 - The “order of growth” of an algorithm.
 - How does the running time of the algorithm grow if the input gets very large?



Analysis of Algorithms

Assumptions

- Analysis is performed with respect to a computational model
- We will usually use a generic uniprocessor random-access machine (RAM)
 - All memory equally expensive to access
 - No concurrent operations
 - All reasonable instructions take unit time
 - Except, of course, function calls
 - Constant word size
 - Unless we are explicitly manipulating bits

Analysis of Algorithms

Assumptions (Cont ...)

- Simplifications
 - Ignore actual and abstract statement costs
 - *Order of growth* is the interesting measure:
 - Highest-order term is what counts
 - Remember, we are doing asymptotic analysis
 - As the input size grows larger it is the high order term that dominates

Recall: Asymptotic Analysis

- Kinds of Time Analysis:
 - **Worst-case:** $T(n)$ is the maximum time on any input of size n (usually we use this).
 - Provides an upper bound on running time
 - An absolute guarantee
 - **Average-case:** $T(n)$ is the average time (given some distribution) over all inputs of size n (we sometimes use this).
 - Provides the expected running time
 - Very useful, but treat with care: what is “average”?
 - Random (equally likely) inputs
 - Real-life inputs
 - **Best-case:** $T(n)$ is the minimum time on any input of size n (never use this).

Analyzing the INSERTION-SORT Algorithm

- **Input size:** number of items in the input (or number of bits, number of edges, etc.)
- **Running time:** number of computational steps (Basic operations)

INSERTION-SORT(A)

```

1. for  $j \leftarrow 2$  to  $length[A]$ 
2.   do  $key \leftarrow A[j]$ 
3.     ▷ Insert  $A[j]$  into the sorted
       ▷ sequence  $A[1..j-1]$ 
4.      $i \leftarrow j - 1$ 
5.     while  $i > 0$  and  $A[i] > key$ 
6.       do  $A[i + 1] \leftarrow A[i]$ 
7.          $i \leftarrow i - 1$ 
8.      $A[i + 1] \leftarrow key$ 
    
```

| <i>cost</i> | <i>times</i> |
|-------------|--------------------------|
| c_1 | n |
| c_2 | $n - 1$ |
| c_4 | $n - 1$ |
| c_5 | $\sum_{j=2}^n t_j$ |
| c_6 | $\sum_{j=2}^n (t_j - 1)$ |
| c_7 | $\sum_{j=2}^n (t_j - 1)$ |
| c_8 | $n - 1$ |

Note that: t_j is number of while expression evaluations for the j^{th} for loop iteration

$$\begin{aligned}
 T(n) = & c_1 n + c_2(n - 1) + c_4(n - 1) + c_5 \sum_{j=2}^n t_j + c_6 \sum_{j=2}^n (t_j - 1) \\
 & + c_7 \sum_{j=2}^n (t_j - 1) + c_8(n - 1)
 \end{aligned}$$

Analyzing the INSERTION-SORT Algorithm

- The running time of INSERTION-SORT is calculated as follows:

$$T(n) = c_1n + c_2(n - 1) + c_4(n - 1) + c_5 \sum_{j=2}^n t_j + c_6 \sum_{j=2}^n (t_j - 1) \\ + c_7 \sum_{j=2}^n (t_j - 1) + c_8(n - 1)$$

Some Simple Summations

$$\sum_{j=1}^n j = 1 + 2 + 3 + \dots + (n-1) + n$$

Gauss's trick for summing n numbers:

Add :

$$1 + 2 + 3 + 4 + \dots + (n-1) + n$$

To:

$$n + (n-1) + (n-2) + (n-3) + \dots + 2 + 1$$

Giving:

$$(n+1) + (n+1) + (n+1) + (n+1) + \dots + (n+1) + (n+1)$$

Hence:

$$\sum_{j=1}^n j = \frac{n(n+1)}{2}$$

To obtain a solution for $\sum_{j=1}^{n-1} j$, substitute in $(n-1)$ for n to obtain:

$$\sum_{j=1}^{n-1} j = 1+2+3+\dots+((n-1)-1)+(n-1) = \frac{(n-1)((n-1)+1)}{2} = \frac{n(n-1)}{2}$$

Some Simple Summations (Cont...)

How about the following?

$$\sum_{j=1}^n c$$

$$\sum_{j=1}^n c = nc$$

$$\sum_{j=2}^n j$$

$$\sum_{j=2}^n j = \frac{n(n+1)}{2} - 1 = \frac{n^2 + n - 2}{2}$$

$$\sum_{j=2}^n (j - 1)$$

$$\sum_{j=2}^n (j - 1) = \sum_{j=2}^n j - \sum_{j=2}^n 1 = \frac{n^2 + n - 2}{2} - (n - 1) = \frac{n^2 - n}{2}$$

Useful Summation Formulas

- **Arithmetic Series:** Constant differences $a_k - a_{k-1}$.

$$\sum_{k=1}^n k = \frac{n(n+1)}{2} = \Theta(n^2) \qquad \sum_{k=1}^n k^2 = \frac{n(2n+1)(n+1)}{6}.$$

- **Geometric Series:** Constant ratio $\frac{a_k}{a_{k-1}}$. For real $x \neq 1$,

$$\sum_{k=0}^n x^k = \frac{x^{n+1} - 1}{x - 1}$$

Infinite decreasing geometric series if $|x| < 1$:

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$$

- **Harmonic Series:** For positive integers n ,

$$H_n = \sum_{k=1}^n \frac{1}{k} = \ln n + O(1)$$

- **Integration and Differentiation of Series:** By differentiating both sides of the infinite geometric series formula and multiplying by x , we get:

$$\sum_{k=0}^{\infty} k * x^{k-1} * x = \sum_{k=0}^{\infty} k * x^k = \frac{x}{(1-x)^2}$$

Analyzing Code: Example 1

Analyze the running time of the following code segment, assuming that the time to perform the assignment on line 3 is 2 units of time. Provide a summation and solve it in closed form.

```
1. for  $i \leftarrow 1$  to  $n$   
2.   do for  $j \leftarrow i$  to  $n$   
3.     do  $k \leftarrow k + j$ 
```

Analyzing Code: Example 2

Analyze the running time of the following code segment, assuming that the time to perform the assignment on line 4 is 2 units of time. Provide a summation and solve it in closed form.

```
1. for  $i \leftarrow 1$  to  $n$   
2.   do for  $j \leftarrow 1$  to  $i$   
3.     do for  $k \leftarrow 1$  to  $j$   
4.       do  $x \leftarrow x + 1$ 
```

Discussion of the INSERTION-SORT Analysis

- What is the best-case running time for insertion sort? When does it occur?
 - Best case -- inner loop body never executed
 - The input array is already sorted
 - $T(n)$ is a linear function

$$T(n) = c_1n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^n c_j + c_6 \sum_{j=2}^n (c_j - 1) + c_7 \sum_{j=2}^n (c_j - 1) + c_8(n-1)$$

$$T(n) = c_9n + c_{10}$$

Discussion of the INSERTION-SORT Analysis continued

- What is the worst-case running time for insertion sort? When does it occur?
 - Worst case -- inner loop body executed for all previous elements
 - $T(n)$ is a quadratic function

$$\begin{aligned}T(n) &= c_1n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) \\&\quad + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1) \\&= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)n \\&\quad - (c_2 + c_4 + c_5 + c_8)\end{aligned}$$

$$T(n) = c_9n^2 + c_{10}n + c_{11}$$

$$\sum_{j=2}^n j = \frac{n(n+1)}{2} - 1$$

Discussion of the INSERTION-SORT Analysis continued

- What is the **average-case running time** for insertion sort? (Refer to the book)
 - The “average case” is often roughly as bad as the worst case.
 - Suppose that we randomly choose n numbers and apply insertion sort. How long does it take to determine where in subarray $A[1 \dots j-1]$ to insert element $A[j]$
 - On average, half the elements in $A[1 \dots j-1]$ are less than $A[j]$, and half the elements are greater.
 - On average, therefore, we check half of the subarray $A[1 \dots j-1]$, and so t_j is about $j/2$. The resulting average-case running time turns out to be a quadratic function of the input size, just like the worst-case running time.
 - It should be mentioned that the average-case running time of an algorithm is usually being computed by applying the technique of **probabilistic analysis** (See Chapter 5)
- How much **space** is needed in (best-case, average-case, worst-case)?
- Is Insertion sort considered as a stable algorithm?