Introduction to Algorithms
Insertion Sort

CSE 680
Prof. Roger Crawfis

Insertion Sort Overview

Figure 2.1 Sorting a hand of cards using insertion sort.

- How do we know where to place the ***next*** card?
- What assumptions do we make at each step?
- What is the algorithm?

Insertion Sort Algorithm

- Given: a *set* **U** of unsorted elements
- Algorithm:
  1. Foreach element *e* in **U**
  2. remove *e* from **U**
  3. place *e* in the **sorted** *sequence* **S** at the correct location.
- Step 3 needs some refinement.
  - What is it's Problem Statement?
  - What are the ***Data Structures***?

Inserting an Element

- Any ideas on how to insert element *e* into our sorted sequence **S**?

Insertion Sort Algorithm

- Your book's pseudo-code:

Why go backwards?

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

Loop invariants and the correctness of insertion sort

- What are the differences?

Insertion Sort Example

- The following slides are from David Luebke when he taught the course at the University of Virginia:
  http://www.cs.virginia.edu/~luebke/cs332/
- Thanks David!
- Unfortunately, he flipped the indices *i* and *j* from the book.

An Example: 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
  }
}
```

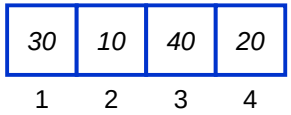

An Example: Insertion Sort

30	10	40	20
1	2	3	4

i = ∅ j = ∅ key = ∅
A[j] = ∅ A[j+1] = ∅

```
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
  }
}
```

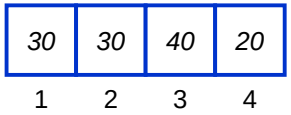
An Example: Insertion Sort



i = 2 j = 1 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
  }
}
```

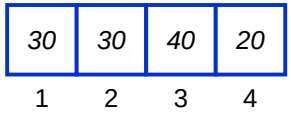

An Example: Insertion Sort



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
  }
}
```

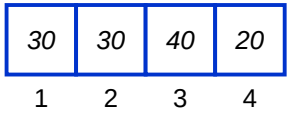

An Example: Insertion Sort



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
  }
}
```

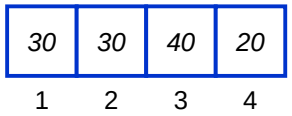

An Example: Insertion Sort



i = 2 j = 0 key = 10
A[j] = ∅ 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
  }
}
```

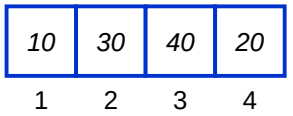
An Example: Insertion Sort



i = 2 j = 0 key = 10
A[j] = ∅ 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
  }
}
```

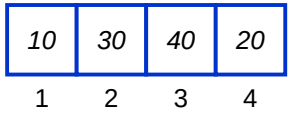

An Example: Insertion Sort



i = 2 j = 0 key = 10
A[j] = ∅ 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
  }
}
```

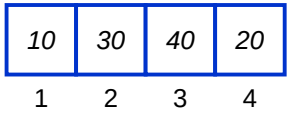

An Example: Insertion Sort



i = 3 j = 0 key = 10
A[j] = ∅ 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
  }
}
```

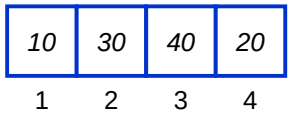

An Example: Insertion Sort



i = 3 j = 0 key = 40
A[j] = ∅ 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
  }
}
```

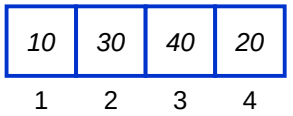
An Example: Insertion Sort



i = 3 j = 0 key = 40
A[j] = ∅ 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
  }
}
```

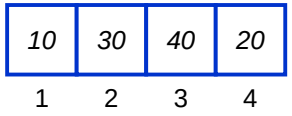

An Example: Insertion Sort



i = 3 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
  }
}
```

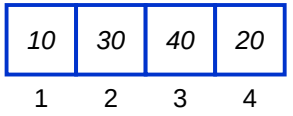

An Example: Insertion Sort



i = 3 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
  }
}
```

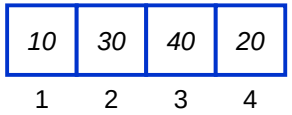

An Example: Insertion Sort



i = 3 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
  }
}
```

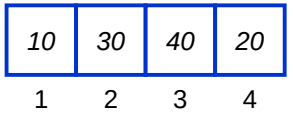
An Example: Insertion Sort



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
  }
}
```

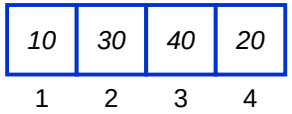

An Example: Insertion Sort



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
  }
}
```

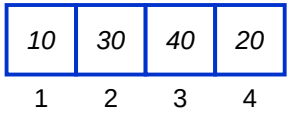

An Example: Insertion Sort



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
  }
}
```

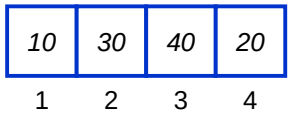

An Example: Insertion Sort



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
  }
}
```

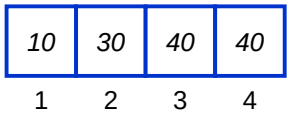
An Example: Insertion Sort



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
  }
}
```

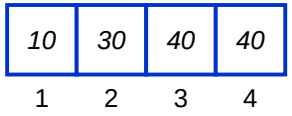

An Example: Insertion Sort



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
  }
}
```

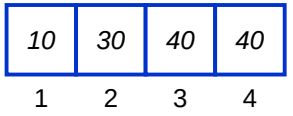

An Example: Insertion Sort



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
  }
}
```

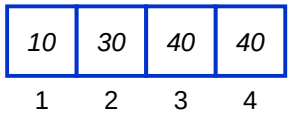

An Example: Insertion Sort



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
  }
}
```

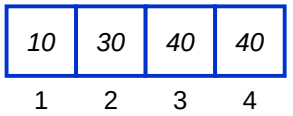
An Example: Insertion Sort



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
  }
}
```

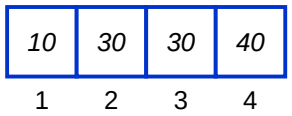

An Example: Insertion Sort



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
  }
}
```

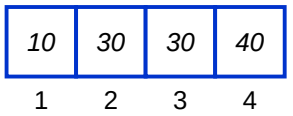

An Example: Insertion Sort



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
  }
}
```

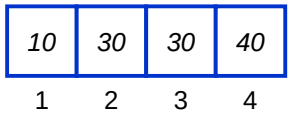

An Example: Insertion Sort



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
  }
}
```

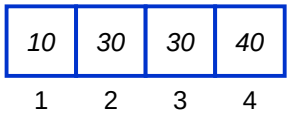
An Example: Insertion Sort



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
  }
}
```

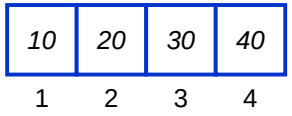

An Example: Insertion Sort



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
  }
}
```

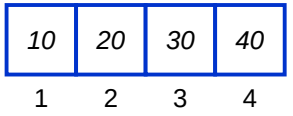

An Example: Insertion Sort



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



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!

Memory Analysis

- The algorithm is said to be ***in-place*** if it uses only a constant amount of memory in accomplishing its solution.
- Clearly, the book's ***implementation*** is better than mine.
- ~~Clearly~~, the book's ***algorithm*** is better than mine.

That is, do we consider my implementation to be a different algorithm or the same?

Correctness

- We can use a property known as **loop-invariance** to reason about the correctness of the algorithm.
- For Insertion-Sort, we can say that:
  - The subarray A[1..*j*-1] elements are in sorted order
  - A[1..*j*-1] is the set of numbers from the original A[1..*j*-1]

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

Insertion Sort Example

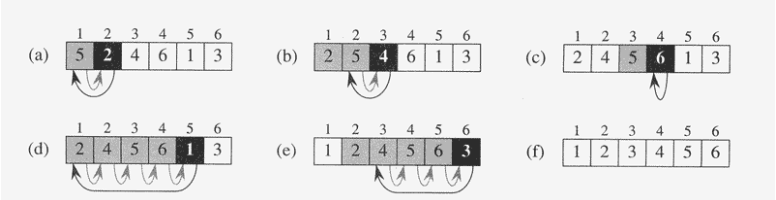


Figure 2.2 The operation of INSERTION-SORT on the array $A = \langle 5, 2, 4, 6, 1, 3 \rangle$. Array indices appear above the rectangles, and values stored in the array positions appear within the rectangles. **(a)–(e)** The iterations of the **for** loop of lines 1–8. In each iteration, the black rectangle holds the key taken from $A[j]$, which is compared with the values in shaded rectangles to its left in the test of line 5. Shaded arrows show array values moved one position to the right in line 6, and black arrows indicate where the key is moved to in line 8. **(f)** The final sorted array.

Loop Invariant

- Initialization
  - True at the beginning of the loop.
- Termination
  - The loop terminates.
  - True, when the loop exists.
- Maintenance
  - True at the end of each iteration of the loop.
  - This allows us to prove the invariant similarly to proof-by-induction.

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?

Bubble-Sort

- Can you come up with loop invariants?
  - Aka, can you prove it is correct?
  - How expensive is it?
    - When is it most expensive?
    - When is it least expensive?

```
BUBBLESORT(A)
1  for i ← 1 to length[A]
2      do for j ← length[A] downto i + 1
3          do if A[j] < A[j − 1]
4              then exchange A[j] ↔ A[j − 1]
```

Selection Sort

- What about selection sort?

```
void selectionSort(int[] a) {
    for (int i = 0; i < a.length - 1; i++) {
        int min = i;
        for (int j = i + 1; j < a.length; j++) {
            if (a[j] < a[min]) {
                min = j;
            }
        }
        if (i != min) {
            int swap = a[i];
            a[i] = a[min];
            a[min] = swap;
        }
    }
}
```