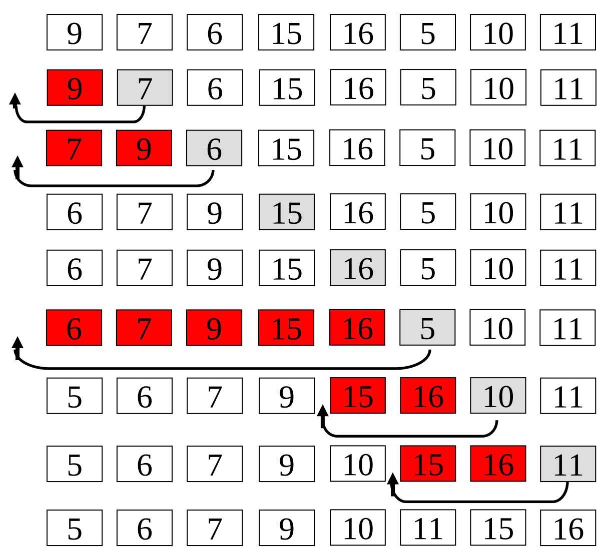
Introduction to Algorithms
Chapter 2
Getting Started

Insertion Sort Execution Example



An Example: Insertion Sort

```
InsertionSort(A, n) {
    for i = 2 to n {
        key = A[i]                          next key
        j = i - 1;                          go left
        while (j > 0) and (A[j] > key) {    find place for key
            A[j+1] = A[j]                   shift sorted right
            j = j – 1                       go left
        }
        A[j+1] = key                        put key in place

    }
}
```

3

An Example: Insertion Sort

30	10	40	20
1	2	3	4

$i = \varnothing \quad j = \varnothing \quad key = \varnothing$

$A[j] = \varnothing \qquad A[j+1] = \varnothing$

```
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
    }
}
```

4

An Example: Insertion Sort

30	10	40	20
1	2	3	4

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
   }
}
```

5

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
   }
}
```

6

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   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

30	30	40	20
1	2	3	4

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
   }
}
```

9

An Example: Insertion Sort

10	30	40	20
1	2	3	4

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
   }
}
```

10

An Example: Insertion Sort

10	30	40	20
1	2	3	4

$i = 3 \quad j = 0 \quad key = 10$
$A[j] = \varnothing \quad A[j+1] = 10$

```
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
  }
}
```

11

An Example: Insertion Sort

10	30	40	20
1	2	3	4

$i = 3 \quad j = 0 \quad key = 40$
$A[j] = \varnothing \quad A[j+1] = 10$

```
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
  }
}
```

12

An Example: Insertion Sort

10	30	40	20
1	2	3	4

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
  }
}
```

13

An Example: Insertion Sort

10	30	40	20
1	2	3	4

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
  }
}
```

14

An Example: Insertion Sort

1	2	3	4
10	*30*	*40*	*20*

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
   }
}
```

15

An Example: Insertion Sort

1	2	3	4
10	*30*	*40*	*20*

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
   }
}
```

16

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 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
  }
}
```

21

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
  }
}
```

22

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
  }
}
```

23

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
  }
}
```

24

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
  }
}
```

25

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
  }
}
```

26

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
   }
}
```

27

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
   }
}
```

28

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
  }
}
```

29

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
  }
}
```

30

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
   }
}
```

31

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!

32

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
  }
}
```

How many times will this while loop execute?

33

Insertion Sort

Statement	Effort
`InsertionSort(A, n) {`	
`for i = 2 to n {`	$c_1 n$
`key = A[i]`	$c_2(n-1)$
`j = i - 1`	$c_3(n-1)$
`while (j > 0) and (A[j] > key) {`	$c_4\boldsymbol{C}$
`A[j+1] = A[j]`	$c_5(\boldsymbol{C}-(n-1))$
`j = j – 1}`	$c_6(\boldsymbol{C}-(n-1))$
`A[j+1] = key`	$c_7(n-1)$
`}}`	

$\boldsymbol{C} = t_2 + t_3 + \ldots + t_n$ where t_i is number of **while** expression evaluations for the i^{th} **for** loop iteration

Body of the **while** statement is executed$= (t_2 - 1) + (t_3 - 1) + \ldots + (t_n - 1)$
$= t_2 + t_3 + \ldots + t_n - (n-1) = \boldsymbol{C} - (n-1)$

34

Analyzing Insertion Sort

$T(n) = c_1 n + c_2(n\text{-}1) + c_3(n\text{-}1) + c_4 C + c_5(C - (n\text{-}1)) + c_6(C - (n\text{-}1)) + c_7(n\text{-}1)$

$= \boldsymbol{c_8 C + c_9 n + c_{10}}$

- What can C be?
  - Best case -- inner loop body never executed (array is sorted)
    - $t_i = 1 \Rightarrow T(n)$ is a linear function $= a.n + b$
  - Worst case -- inner loop body executed for all previous elements (array sorted in reverse order)
    - $t_i = i \Rightarrow T(n)$ is a quadratic function $= a.n^2 + b.n + c$
  - *If T is a quadratic function, which terms in the above equation matter?*

$c_1 n$
$c_2(n\text{-}1)$
$c_3(n\text{-}1)$
$c_4 C$
$c_5(C\text{-}(n\text{-}1))$
$c_6(C\text{-}(n\text{-}1))$
$c_7(n\text{-}1)$

35

Analyzing Insertion Sort (Cont.)

- Best Case
  - If A is sorted: $O(n)$ comparisons
- Worst Case
  - If A is reversed sorted: $O(n^2)$ comparisons
- Average Case
  - If A is randomly sorted: $O(n^2)$ comparisons

36

Divide-and-Conquer

- Recursive in structure
  - *Divide* the problem into several smaller sub-problems that are similar to the original but smaller in size
  - *Conquer* the sub-problems by solving them recursively. If they are small enough, just solve them in a straightforward manner.
  - *Combine* the solutions to create a solution to the original problem

An Example: Merge Sort

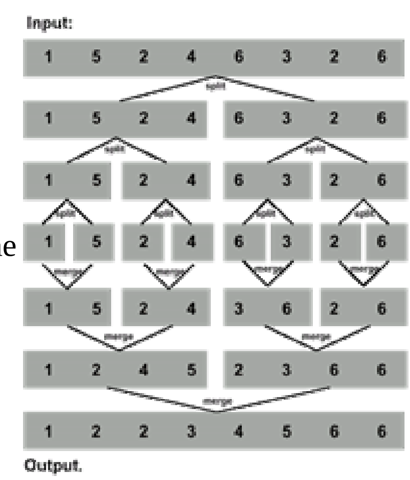
- *Divide*: Divide the n-element sequence to be sorted into two subsequences of $n/2$ elements each
- *Conquer:* Sort the two subsequences recursively using merge sort.
- *Combine*: Merge the two sorted subsequences to produce the sorted answer.

Merge Sort

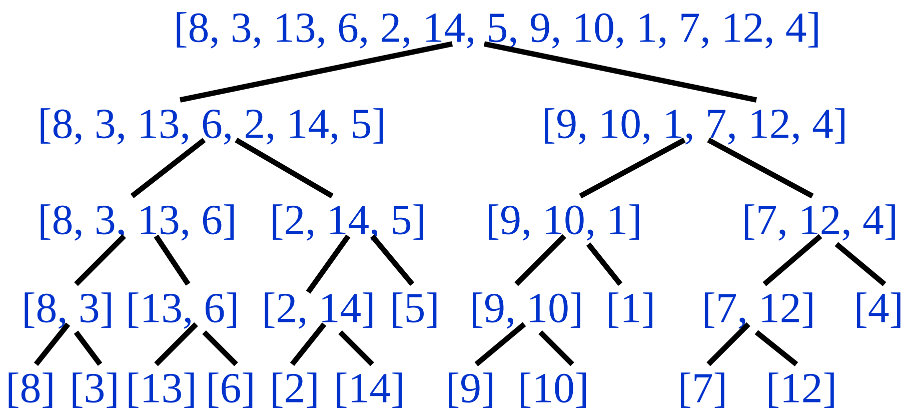
```
MergeSort(A, left, right) {
  if (left < right) {
      mid = floor((left + right) / 2);
      MergeSort(A, left, mid);
      MergeSort(A, mid+1, right);
      Merge(A, left, mid, right);
  }
}

// Merge() takes two sorted subarrays of A and
// merges them into a single sorted subarray of A.
// It requires O(n) time
```


Merge Sort Revisited

- To sort n numbers
  - if $n = 1$ done!
  - recursively sort 2 lists of numbers $\lfloor n/2 \rfloor$ and $\lceil n/2 \rceil$ elements
  - merge 2 sorted lists in $O(n)$ time
- Strategy
  - break problem into similar (smaller) subproblems
  - recursively solve subproblems
  - combine solutions to answer

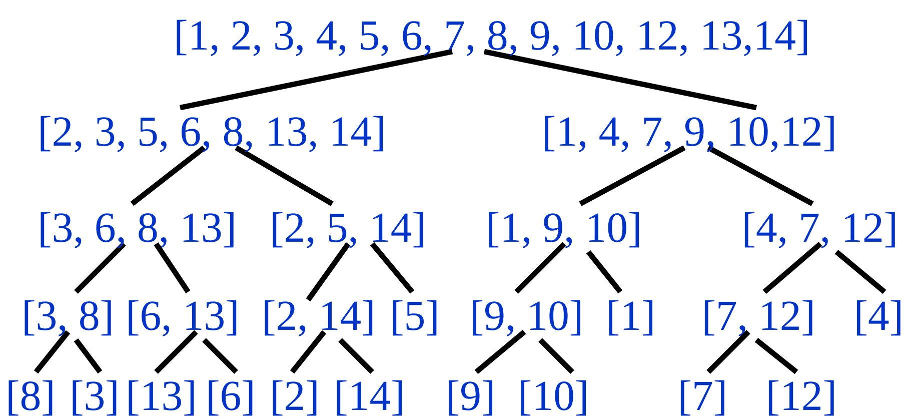


Merge Sort



41

Merge Sort



42

Analysis of Merge Sort

- Divide: computing the middle takes $O(1)$
- Conquer: solving 2 sub-problem takes $2T(n/2)$
- Combine: merging n-element takes $O(n)$
- Total:

$$T(n) = O(1) \quad \text{if } n = 1$$
$$T(n) = 2T(n/2) + O(n) + O(1) \quad \text{if } n > 1$$

$$\Rightarrow T(n) = O(n \lg n)$$

- Solving this recurrence (*how?*) gives $T(n) = O(n \lg n)$
- This expression is a *recurrence*
- **T(n)** denote the number of operations required by an algorithm to solve a given class of problems.

43

Review: Analysis of Merge Sort

Statement	Effort
`MergeSort(A, left, right) {`	T(n)
`   if (left < right) {`	$\Theta(1)$
`      mid = floor((left + right) / 2);`	$\Theta(1)$
`      MergeSort(A, left, mid);`	T(n/2)
`      MergeSort(A, mid+1, right);`	T(n/2)
`      Merge(A, left, mid, right);`	$\Theta(n)$
`   }`	
`}`	

- So $T(n) = \Theta(1)$ when n = 1, and

  $2T(n/2) + \Theta(n)$ when n > 1
- Solving this recurrence (*how?*) gives **T(n) = n log n**
- This expression is a *recurrence*

44

Analysis of Merge Sort

Assume $n=2^k$ for $k>=1$

$T(n) = 2\,\underline{T(n/2)} + bn + c$

$\quad T(n/2) = 2T((n/2)/2) + b(n/2) + c$

$= 2[2T(n/4) + b(n/2) + c] + bn + c$

$= 4\,T(n/4) + bn + 2c + bn + c$

$= 4\,T(n/4) + 2bn + (1 + 2)\,c = 2^2\,T(n/2^2) + 2bn + (2^0 + 2^1)$

$= 4\,[2T((n/4)/2) + b(n/4) + c] + 2bn + (1+2)c$

$= 8\,T(n/8) + 3bn + (1+2+4)c$

$= 2^3\,T(n/2^3) + 3bn + (2^0+2^1+2^2)c$

$\vdots$

$= 2^k\,\underline{T(n/2^k)} + kbn + (2^0+2^1+\ldots+2^{k-1})c$

$\quad T(1) = a$, since $n=2^k \rightarrow \log n = \log 2^k = k$

$T(n) = 2^k.\,a + \underline{k}\,bn + (2^0+2^1+\ldots+2^{k-1})\,c$

$= b.\,n\ \underline{\log n} + (a + c)\,n - c$

$= O(n \log n)$ Worst case