Problem 3.2 (page 61)

Indicate, for each pair of expressions (A, B) in the table below, whether A is $O, o, \Omega, \omega, \Theta$ of B. Assume that $k \geq 1$, $\epsilon > 0$, and $c > 1$ are constants. Your answer should be in the form of the table with "yes" or "no" written in each box.

	A	B	O	o	Ω	ω	Θ
a.	$lg^k n$	n^ϵ					
b.	n^k	c^n					
c.	$\sqrt{n}$	$n^{sin\ n}$					
d.	2^n	$2^{n/2}$					
e.	n^{lgc}	c^{lgn}					
f.	$lg(n!)$	$lg(n^n)$					

Problem 3.2 (page 61)

Indicate, for each pair of expressions (A, B) in the table below, whether A is $O, o, \Omega, \omega, \Theta$ of B. Assume that $k \geq 1$, $\epsilon > 0$, and $c > 1$ are constants. Your answer should be in the form of the table with "yes" or "no" written in each box.

	A	B	O	o	Ω	ω	Θ
a.	$lg^k n$	n^ϵ	yes	yes	no	no	no
b.	n^k	c^n	yes	yes	no	no	no
c.	$\sqrt{n}$	$n^{sin\ n}$	no	no	no	no	no
d.	2^n	$2^{n/2}$	no	no	yes	yes	no
e.	n^{lgc}	c^{lgn}	yes	no	yes	no	yes
f.	$lg(n!)$	$lg(n^n)$	yes	no	yes	no	yes

(a) Apply L'Hospital's rule repeatedly to see that $\lim_{n\to\infty} \frac{(lgn)^k}{n^\epsilon} = 0$ to conclude that $(lgn)^k = o(n^\epsilon)$.

$$\begin{aligned}\lim_{n\to\infty} \frac{(lgn)^k}{n^\epsilon} &= \lim_{n\to\infty} \frac{k(lgn)^{k-1}\frac{1}{n}}{\epsilon n^{\epsilon-1}} \\ &= \lim_{n\to\infty} \frac{k(lgn)^{k-1}}{\epsilon n^\epsilon} \\ &= \lim_{n\to\infty} \frac{k\frac{d(lgn)^{k-1}}{dn}}{\epsilon\frac{dn^\epsilon}{dn}} \\ &= \lim_{n\to\infty} \frac{k(k-1)(lgn)^{k-2}\frac{1}{n}}{\epsilon^2 n^{\epsilon-1}}\end{aligned}$$

After k applications of the rule, we get

$$\lim_{n\to\infty} \frac{k(k-1)(k-2)....1}{\epsilon^k n^\epsilon} = 0$$

(b) Apply L'Hospital's rule repeatedly to see that $\lim_{n\to\infty} \frac{n^k}{c^n} = 0$ to conclude that $n^k = o(c^n)$.

(c) You can visually inspect the plots to see that $n^{sin\ n}$ is an oscillating function. $sin\ n$ oscillates between 1 and -1. When at its maximum value, $n^{sinn} > c\sqrt{n}$ and thus $n^{sin\ n} \neq O(\sqrt{n})$. When $sin\ n$ is at its minimum, $n^{sin\ n} < c\sqrt{n}$ and thus $n^{sin\ n} \neq \Omega(\sqrt{n})$.

(d) $\lim_{n\to\infty} \frac{2^n}{2^{n/2}} = \infty$ and therefore $2^n = \omega(2^{n/2})$.

(e) Recall that $n^{lgc} = c^{lgc}$.

(f) Note $lg(n^n) = nlg(n)$, and using Stirling's formula it is shown in the text that $lg(n!) = \Theta(nlg(n))$.

Common Functions Review

Monotonicity

- $f(n)$ is
  - **monotonically increasing** if $m \le n \Rightarrow f(m) \le f(n)$.
  - **monotonically decreasing** if $m \ge n \Rightarrow f(m) \ge f(n)$.
  - **strictly increasing** if $m < n \Rightarrow f(m) < f(n)$.
  - **strictly decreasing** if $m > n \Rightarrow f(m) > f(n)$.

Exponentials

- **Useful Identities:**

$$a^{-1} = \frac{1}{a}$$
$$(a^m)^n = a^{mn}$$
$$a^m a^n = a^{m+n}$$

- **Exponentials and polynomials**

$$\lim_{n\to\infty} \frac{n^b}{a^n} = 0$$
$$\Rightarrow n^b = o(a^n)$$

Logarithms

$x = \log_b a$ is the exponent for $a = b^x$.

Natural log: $\ln a = \log_e a$

Binary log: $\lg a = \log_2 a$

$\lg^2 a = (\lg a)^2$

$\lg \lg a = \lg (\lg a)$

$$a = b^{\log_b a}$$
$$\log_c(ab) = \log_c a + \log_c b$$
$$\log_b a^n = n \log_b a$$
$$\log_b a = \frac{\log_c a}{\log_c b}$$
$$\log_b (1/a) = -\log_b a$$
$$\log_b a = \frac{1}{\log_a b}$$
$$a^{\log_b c} = c^{\log_b a}$$

Logarithms and exponentials – Bases

- If the base of a logarithm is changed from one constant to another, the value is altered by a constant factor.
  - **Ex:** $\log_{10} n * \mathbf{\log_2 10} = \log_2 n$.
  - Base of logarithm is not an issue in asymptotic notation.
- Exponentials with different bases differ by a exponential factor (not a constant factor).
  - **Ex:** $2^n = \mathbf{(2/3)^n} * 3^n$.

Exercise

Express functions in A in asymptotic notation using functions in B.

A	B	
$5n^2 + 100n$	$3n^2 + 2$	$A \in \Theta(B)$

$A \in \Theta(n^2),\ n^2 \in \Theta(B) \Rightarrow A \in \Theta(B)$

$\log_3(n^2)$	$\log_2(n^3)$	$A \in \Theta(B)$

$\log_b a = \log_c a / \log_c b$; $A = 2\lg n / \lg 3$, $B = 3\lg n$, $A/B = 2/(3\lg 3)$

$n^{\lg 4}$	$3^{\lg n}$	$A \in \omega(B)$

$a^{\log b} = b^{\log a}$; $B = 3^{\lg n} = n^{\lg 3}$; $A/B = n^{\lg(4/3)} \to \infty$ as $n \to \infty$

$\lg^2 n$	$n^{1/2}$	$A \in o(B)$

$\lim_{n\to\infty} (\lg^a n / n^b) = 0$ (here $a = 2$ and $b = 1/2$) $\Rightarrow A \in o(B)$

Summations – Review

Review on Summations

- Why do we need summation formulas?

  For computing the running times of iterative constructs (loops). (CLRS – Appendix A)

  Example: Maximum Subvector

  Given an array $A[1\ldots n]$ of numeric values (can be positive, zero, and negative) determine the subvector $A[i\ldots j]$ $(1 \le i \le j \le n)$ whose sum of elements is maximum over all subvectors.

1	-2	2	2

Review on Summations

```
MaxSubvector(A, n)
    maxsum ← 0;
    for i ← 1 to n
      do for j = i to n
          sum ← 0
          for k ← i to j
            do sum += A[k]
          maxsum ← max(sum, maxsum)
    return maxsum
```

- $T(n) = \sum_{i=1}^{n}\sum_{j=i}^{n}\sum_{k=i}^{j} 1$
- NOTE: This is not a simplified solution. What is the final answer?

Review on Summations

- **Cubic Series:** For $n \geq 0$,

$$\sum_{i=1}^{n} i^3 = 1^3 + 2^3 + \ldots + n^3 = \frac{n^2(n+1)^2}{4}$$

- **Geometric Series:** For real $x \neq 1$,

$$\sum_{k=0}^{n} x^k = 1 + x + x^2 + \ldots + x^n = \frac{x^{n+1} - 1}{x - 1}$$

For $|x| < 1$, $\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$

Review on Summations

- **Linear-Geometric Series:** For $n \geq 0$, real $c \neq 1$,

$$\sum_{i=1}^{n} ic^i = c + 2c^2 + \ldots + nc^n = \frac{-(n+1)c^{n+1} + nc^{n+2} + c}{(c-1)^2}$$

- **Harmonic Series:** nth harmonic number, $n \in \mathrm{I}^+$,

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}$$

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

Review on Summations

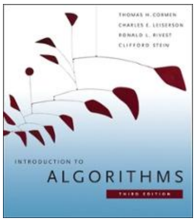
- **Telescoping Series:**

$$\sum_{k=1}^{n} a_k - a_{k-1} = a_n - a_0$$

- **Differentiating Series:** For $|x| < 1$,

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

CS146 Data Structures and Algorithms



Chapter 2: Getting Started

(Analysis and Design Algorithms Insertion Sort and Merge Sort)

L2.17

Why study algorithms and performance?

- Algorithms help us to understand scalability.
- Performance often draws the line between what is feasible and what is impossible.
- Algorithmic mathematics provides a language for talking about program behavior.
- Performance is the currency of computing.
- The lessons of program performance generalize to other computing resources.
- Speed is fun!

L2.18

The problem of sorting

Input: sequence $\langle a_1, a_2, \ldots, a_n \rangle$ of numbers.

Output: permutation $\langle a'_1, a'_2, \ldots, a'_n \rangle$ Such that $a'_1 \le a'_2 \le \ldots \le a'_n$.

Example:

Input: 8 2 4 9 3 6

Output: 2 3 4 6 8 9

L2.19

INSERTION-SORT

"pseudocode"

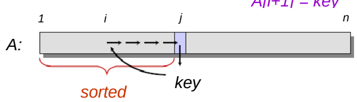
```
INSERTION-SORT (A, n)      ⊳ A[1 . . n]
  for j ←2 to n
    do key ← A[ j]
       i ← j –1
       while i > 0 and A[i] > key
            do A[i+1] ← A[i]
               i ← i –1
       A[i+1] = key
```

L2.20

INSERTION-SORT

"pseudocode"

```
INSERTION-SORT (A, n)        ⊳ A[1 . . n]
   for j ← 2 to n
     do key ← A[ j]
        i ← j – 1
        while i > 0 and A[i] > key
           do A[i+1] ← A[i]
              i ← i – 1
        A[i+1] = key
```



L2.21

Example of insertion sort

8 2 4 9 3 6

L2.22

Example of insertion sort

8 2 4 9 3 6

L2.23

Example of insertion sort

8 2 4 9 3 6

2 8 4 9 3 6

L2.24

Example of insertion sort

8 2 4 9 3 6

2 8 4 9 3 6

Example of insertion sort

8 2 4 9 3 6

2 8 4 9 3 6

2 4 8 9 3 6

Example of insertion sort

8 2 4 9 3 6

2 8 4 9 3 6

2 4 8 9 3 6

Example of insertion sort

8 2 4 9 3 6

2 8 4 9 3 6

2 4 8 9 3 6

2 4 8 9 3 6

Example of insertion sort

8 2 4 9 3 6

2 8 4 9 3 6

2 4 8 9 3 6

2 4 8 9 3 6

L2.29

Example of insertion sort

8 2 4 9 3 6

2 8 4 9 3 6

2 4 8 9 3 6

2 4 8 9 3 6

2 3 4 8 9 6

L2.30

Example of insertion sort

8 2 4 9 3 6

2 8 4 9 3 6

2 4 8 9 3 6

2 4 8 9 3 6

2 3 4 8 9 6

L2.31

Example of insertion sort

8 2 4 9 3 6

2 8 4 9 3 6

2 4 8 9 3 6

2 4 8 9 3 6

2 3 4 8 9 6

2 3 4 6 8 9 *done*

L2.32

Source Code: 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

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
  }
}
```


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 = 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
  }
}
```

L2.37

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
  }
}
```

L2.38

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
  }
}
```

L2.39

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
  }
}
```

L2.40

An Example: Insertion Sort

10	30	40	20
1	2	3	4

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
  }
}
```

L2.41

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
  }
}
```

L2.42

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
  }
}
```

L2.43

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
  }
}
```

L2.44

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
  }
}
```

L2.45

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
  }
}
```

L2.46

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
  }
}
```

L2.47

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
  }
}
```

L2.48

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
  }
}
```


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
  }
}
```

L2.53

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
  }
}
```

L2.54

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
  }
}
```

L2.55

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
  }
}
```

L2.56

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
  }
}
```

L2.57

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
  }
}
```

L2.58

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
  }
}
```

L2.59

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
  }
}
```

L2.60

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
   }
}
```

L2.61

An Example: Insertion Sort

10	20	30	40
1	2	3	4

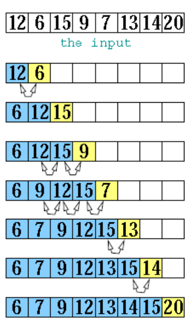
i = 4 j = 1 key = 20
A[j] = 10 A[j+1] = 20

```
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!

L2.62

Another Example: Insertion Sort using Swap(x,y)



L2.63

Animating Sorting Algorithms

- Check out the Sorting Algorithms Animator, at:
  https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html

https://www.toptal.com/developers/sorting-algorithms

- Try it out with random, ascending, and descending inputs

L2.64

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

```
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?

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) {`	$c_4 T$
`A[j+1] = A[j]`	$c_5(T-(n-1))$
`j = j - 1`	$c_6(T-(n-1))$
`}`	0
`A[j+1] = key`	$c_7(n-1)$
`}`	0
`}`	

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

Analyzing Insertion Sort

- $T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 T + c_5(T - (n-1)) + c_6(T - (n-1)) + c_7(n-1)$
  $= c_8 T + c_9 n + c_{10}$
- What can T be?
  - Best case -- inner loop body never executed
    - $t_i = 1 \Rightarrow T(n)$ is a linear function
  - Worst case -- inner loop body executed for all previous elements
    - $t_i = i \Rightarrow T(n)$ is a quadratic function
  - Average case
    - ???

Running time

- The running time depends on the input: an already sorted sequence is easier to sort.
- Parameterize the running time by the size of the input, since short sequences are easier to sort than long ones.
- Generally, we seek upper bounds on the running time, because everybody likes a guarantee.

Kinds of analyses

Worst-case: (usually)
- $T(n)$ =maximum time of algorithm on any input of size n.

Average-case: (sometimes)
- $T(n)$ =expected time of algorithm over all inputs of size n.
- Need assumption of statistical distribution of inputs.

Best-case: (bogus)
- Cheat with a slow algorithm that works fast on *some* input.

Machine-independent time

What is insertion sort's worst-case time?
- It depends on the speed of our computer:
  - relative speed (on the same machine),
  - absolute speed (on different machines).

BIG IDEA:
- Ignore machine-dependent constants.
- Look at growth of $T(n)$ as $n \to \infty$.

"Asymptotic Analysis"

Θ-notation

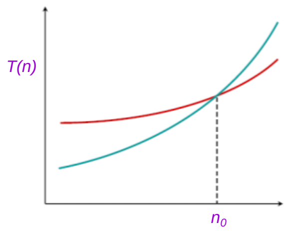
Math:

$\Theta(g(n)) = \{ f(n)$: there exist positive constants c_1, c_2, and n_0 such that $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ for all $n \ge n_0 \}$

Engineering:
- Drop low-order terms; ignore leading constants.
- Example: $3n^3 + 90n^2 - 5n + 6046 = \Theta(n^3)$

Asymptotic performance

When n gets large enough, a $\Theta(n^2)$algorithm **always** beats a $\Theta(n^3)$algorithm.



- *We shouldn't ignore asymptotically slower algorithms, however.*
- *Real-world design situations often call for a careful balancing of engineering objectives.*
- *Asymptotic analysis is a useful tool to help to structure our thinking.*

Insertion sort analysis

Worst case: Input reverse sorted.

$$T(n)=\sum_{j=2}^{n}\Theta(j)=\Theta(n^2)$$ [arithmetic series]

Average case: All permutations equally likely.

$$T(n)=\sum_{j=2}^{n}\Theta(j/2)=\Theta(n^2)$$

Is insertion sort a fast sorting algorithm?

•Moderately so, for small n.

•Not at all, for large n.

Merge sort

- Merge sort is a well-known example of an algorithm design called Divide-and-Conquer consisting of the following 3 steps:
  - Divide: divide the given instance into smaller instances.
  - Conquer: solve all of the smaller instances.
  - Combine: combine the outcomes of the smaller instances.

Merge sort

MERGE-SORT *A*[1 . . n]

1. If n= 1, done.
2. Recursively sort A[1 . . .n/2.]and A[[n/2]+1 . . n] .
3. "Merge" the 2 sorted lists.

Key subroutine: MERGE

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
//    (how long should this take?)
```

L2.77

Merging two sorted arrays

20	12
13	11
7	9
2	1

L2.78

Merging two sorted arrays

20	*12*
13	*11*
7	*9*
2	*1*

1

L2.79

Merging two sorted arrays

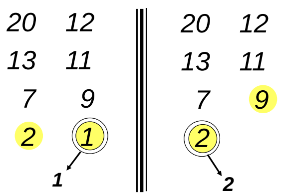
20	12
13	11
7	9
2	1

1

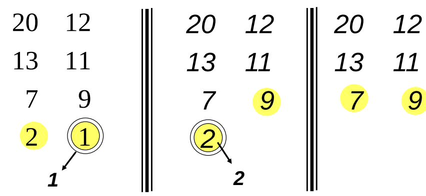
20	*12*
13	*11*
7	*9*
2	

L2.80

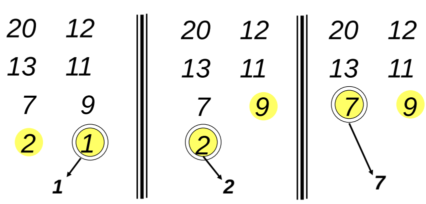
Merging two sorted arrays



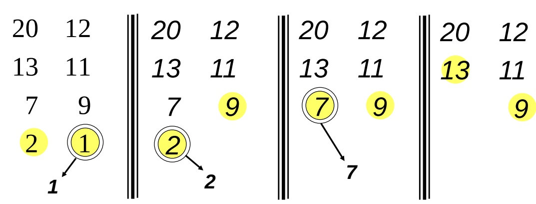
Merging two sorted arrays



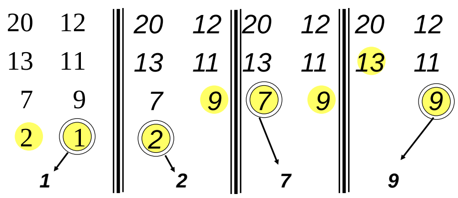
Merging two sorted arrays



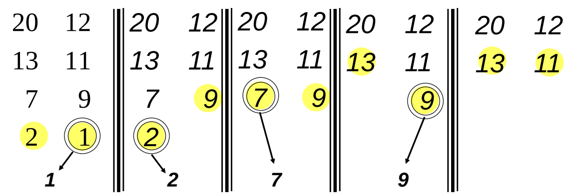
Merging two sorted arrays



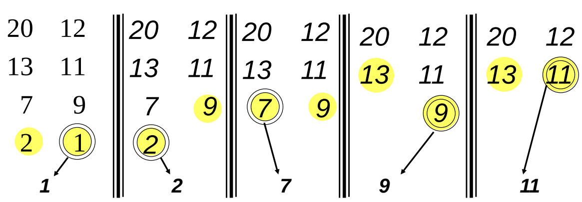
Merging two sorted arrays



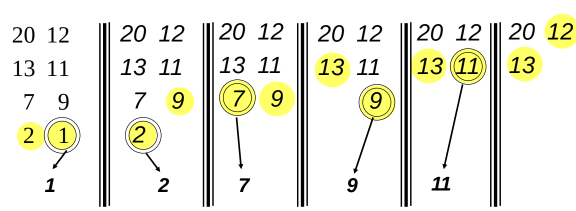
Merging two sorted arrays



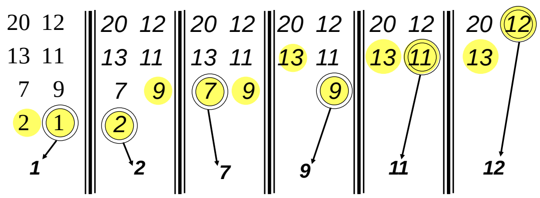
Merging two sorted arrays



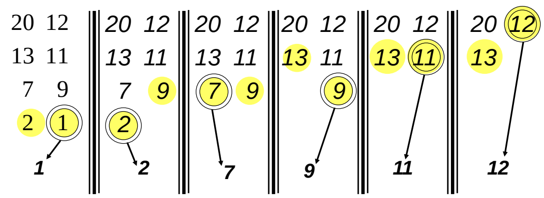
Merging two sorted arrays



Merging two sorted arrays



Merging two sorted arrays



Time = Θ(n) to merge a total of n elements (linear time).