COMP2402
Abstract Data Types and Algorithms

Asymptotic Analysis and Other Math

Selection Sort

***Find** the **Smallest** and **Swap** it with the **First Position**,*

***Find** the **Next Smallest** and **Swap** it with the **Second Position**,*

...

***Find** the **Second-to-Last Smallest** and **Swap** it with the **Second Last Position**,*

the Largest** is **Already** in the **Last Position

```
for i ∊ [0, length-1):
    nextsmallestindex ← i
    for j ∊ [i+1, length):
        if (item[j] < item[nextsmallest]):
            nextsmallest ← j
    swap item[i] with item[nextsmallest]
```

Selection Sort

***Find** the **Smallest** and **Swap** it with the **First Position**,*

***Find** the **Next Smallest** and **Swap** it with the **Second Position**,*

…

***Find** the **Second-to-Last Smallest** and **Swap** it with the **Second Last Position**,*

*the **Largest** is **Already** in the **Last Position***

Example

❼ ❸ ❾ ❽ ❺ ❷ ❸ ❶

Selection Sort

***Find** the **Smallest** and **Swap** it with the **First Position**,*

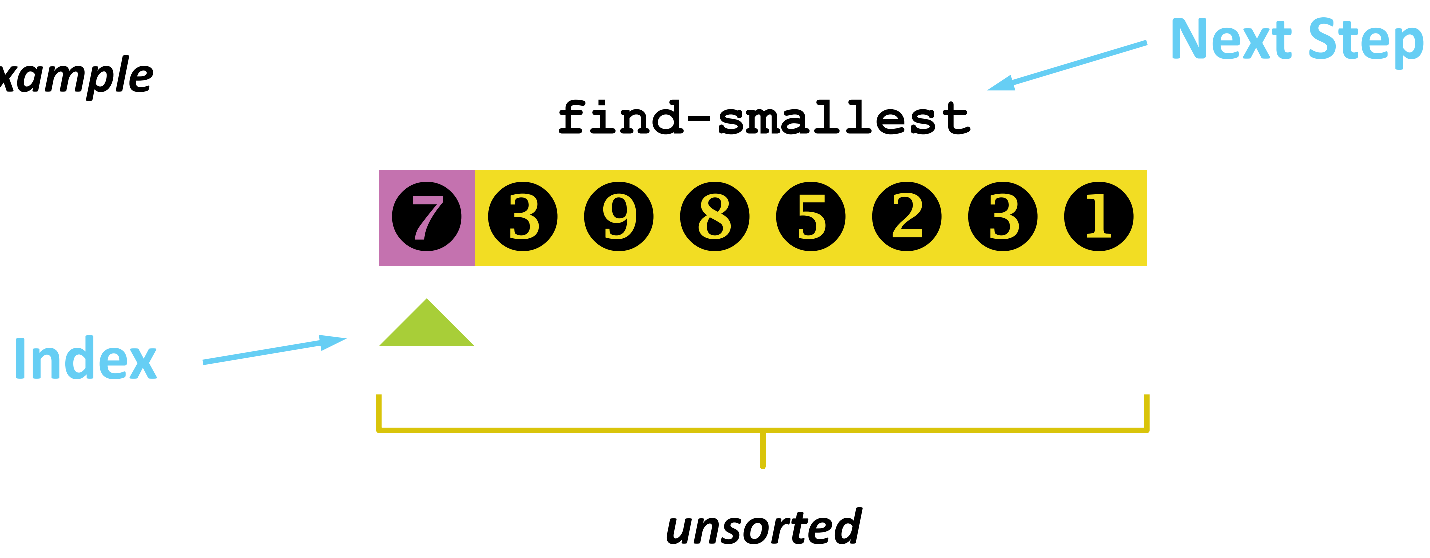
***Find** the **Next Smallest** and **Swap** it with the **Second Position**,*

...

***Find** the **Second-to-Last Smallest** and **Swap** it with the **Second Last Position**,*

*the **Largest** is **Already** in the **Last Position***

Example



Selection Sort

***Find** the **Smallest** and **Swap** it with the **First Position**,*

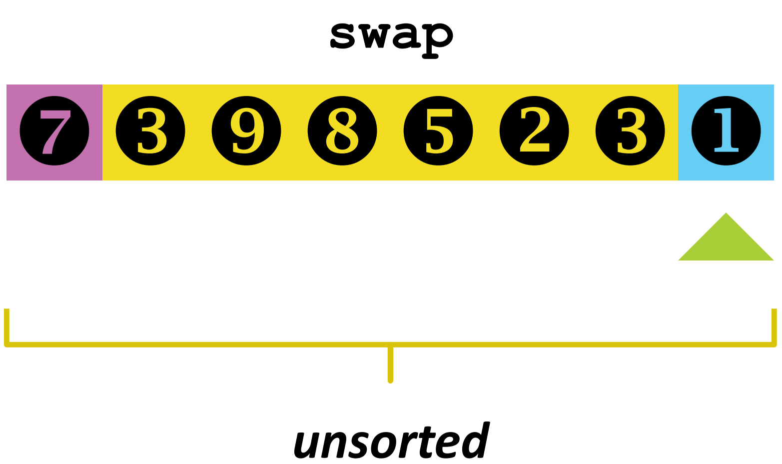
***Find** the **Next Smallest** and **Swap** it with the **Second Position**,*

...

***Find** the **Second-to-Last Smallest** and **Swap** it with the **Second Last Position**,*

the Largest** is **Already** in the **Last Position

Example



Selection Sort

***Find** the **Smallest** and **Swap** it with the **First Position**,*

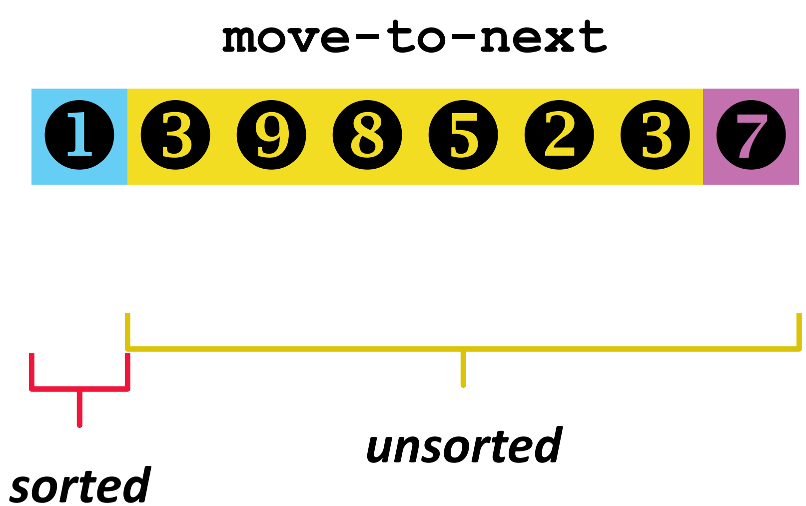
***Find** the **Next Smallest** and **Swap** it with the **Second Position**,*

…

***Find** the **Second-to-Last Smallest** and **Swap** it with the **Second Last Position**,*

*the **Largest** is **Already** in the **Last Position***

Example



Selection Sort

Find the Smallest and Swap it with the First Position,

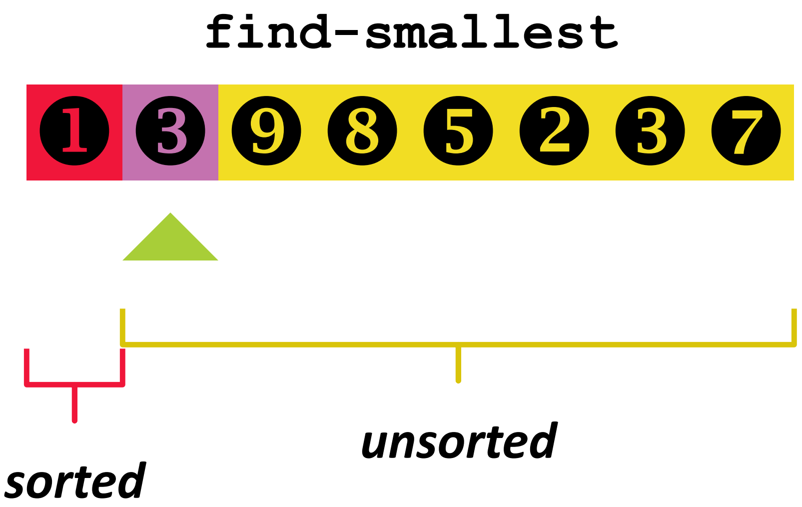
Find the Next Smallest and Swap it with the Second Position,

...

Find the Second-to-Last Smallest and Swap it with the Second Last Position,

the Largest is Already in the Last Position

Example



Selection Sort

***Find** the **Smallest** and **Swap** it with the **First Position**,*

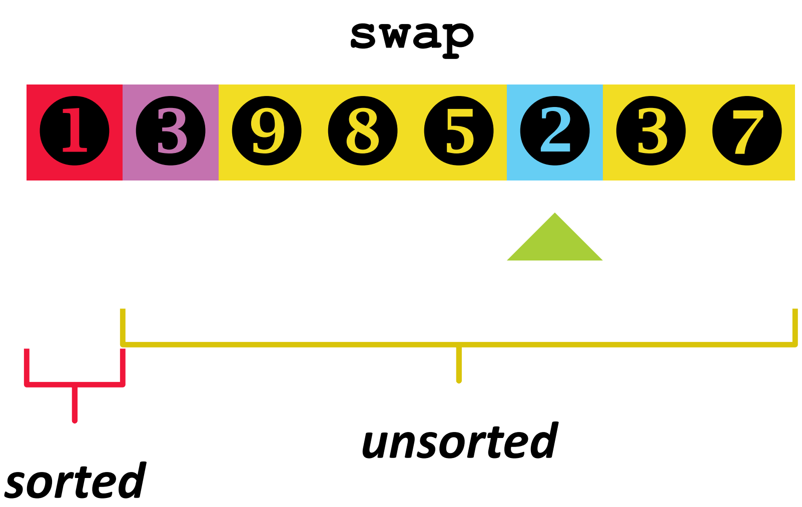
***Find** the **Next Smallest** and **Swap** it with the **Second Position**,*

...

***Find** the **Second-to-Last Smallest** and **Swap** it with the **Second Last Position**,*

*the **Largest** is **Already** in the **Last Position***

Example



Selection Sort

***Find** the **Smallest** and **Swap** it with the **First Position**,*

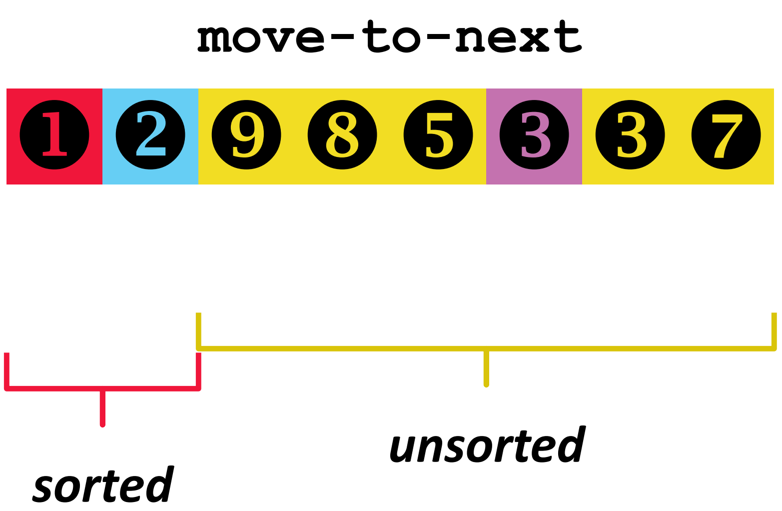
***Find** the **Next Smallest** and **Swap** it with the **Second Position**,*

...

***Find** the **Second-to-Last Smallest** and **Swap** it with the **Second Last Position**,*

the Largest** is **Already** in the **Last Position

Example



Selection Sort

***Find** the **Smallest** and **Swap** it with the **First Position**,*

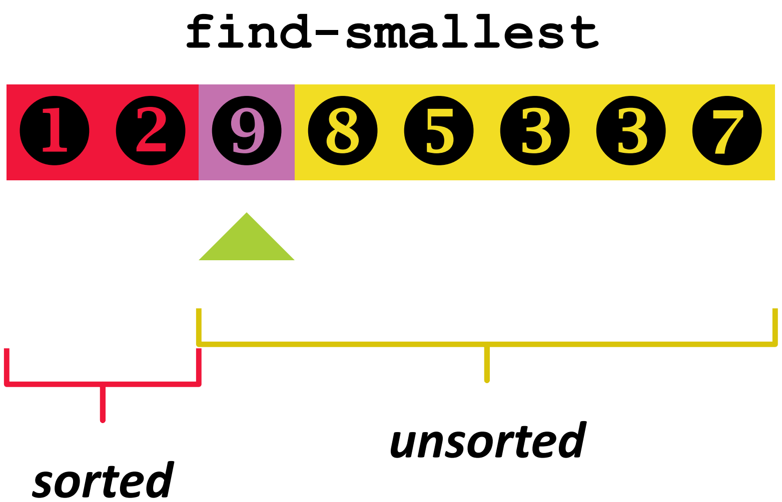
***Find** the **Next Smallest** and **Swap** it with the **Second Position**,*

...

***Find** the **Second-to-Last Smallest** and **Swap** it with the **Second Last Position**,*

*the **Largest** is **Already** in the **Last Position***

Example



Selection Sort

***Find** the **Smallest** and **Swap** it with the **First Position**,*

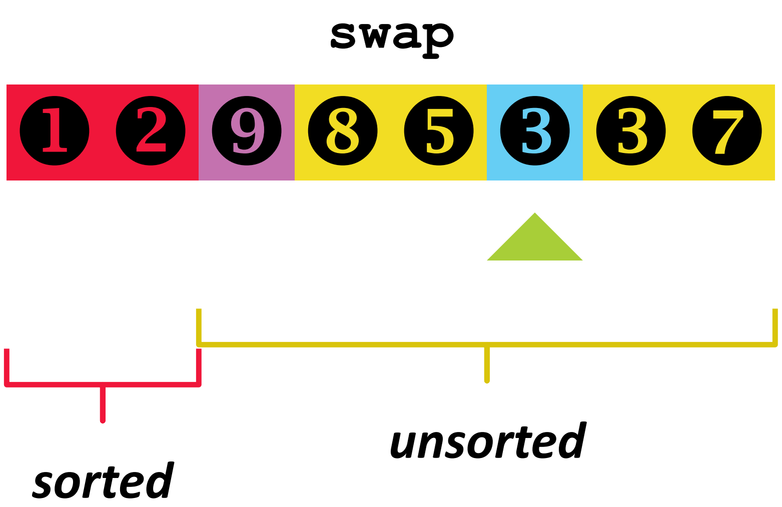
***Find** the **Next Smallest** and **Swap** it with the **Second Position**,*

...

***Find** the **Second-to-Last Smallest** and **Swap** it with the **Second Last Position**,*

*the **Largest** is **Already** in the **Last Position***

Example



Selection Sort

***Find** the **Smallest** and **Swap** it with the **First Position**,*

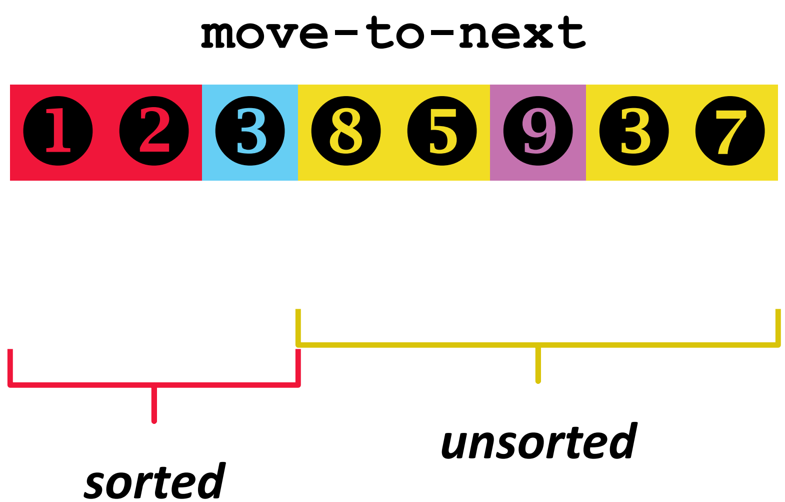
***Find** the **Next Smallest** and **Swap** it with the **Second Position**,*

...

***Find** the **Second-to-Last Smallest** and **Swap** it with the **Second Last Position**,*

*the **Largest** is **Already** in the **Last Position***

Example



Selection Sort

***Find** the **Smallest** and **Swap** it with the **First Position**,*

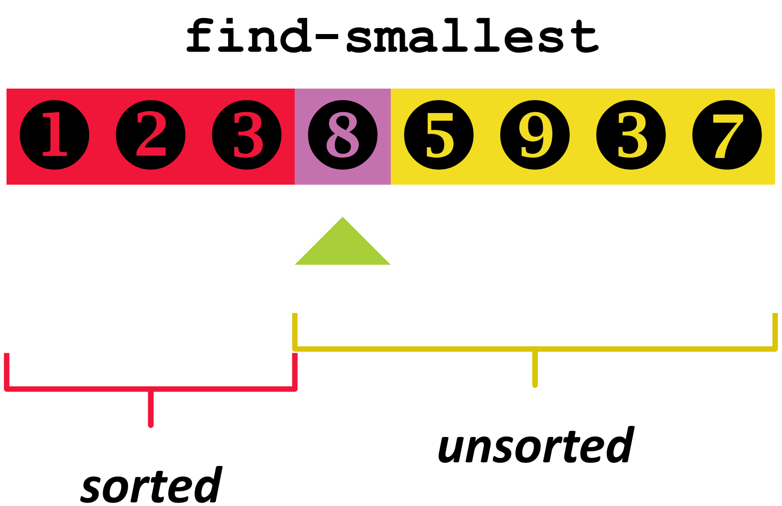
***Find** the **Next Smallest** and **Swap** it with the **Second Position**,*

...

***Find** the **Second-to-Last Smallest** and **Swap** it with the **Second Last Position**,*

*the **Largest** is **Already** in the **Last Position***

Example



Selection Sort

***Find** the **Smallest** and **Swap** it with the **First Position**,*

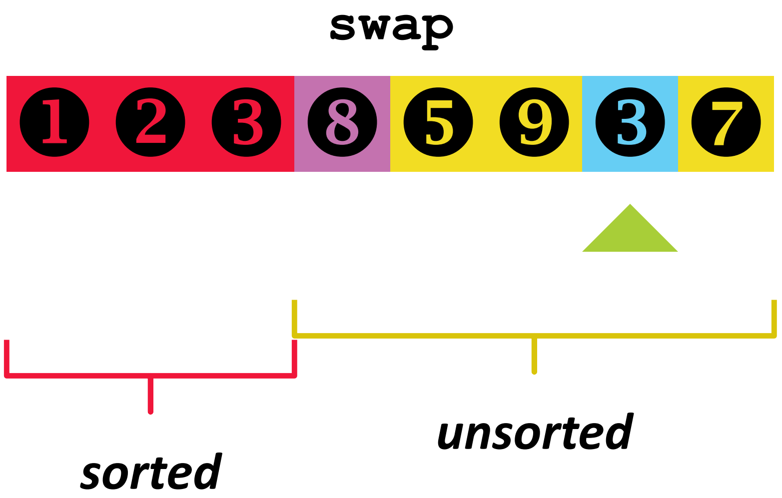
***Find** the **Next Smallest** and **Swap** it with the **Second Position**,*

...

***Find** the **Second-to-Last Smallest** and **Swap** it with the **Second Last Position**,*

*the **Largest** is **Already** in the **Last Position***

Example



Selection Sort

Find **the** ***Smallest*** **and** ***Swap*** **it with the** ***First Position*,**

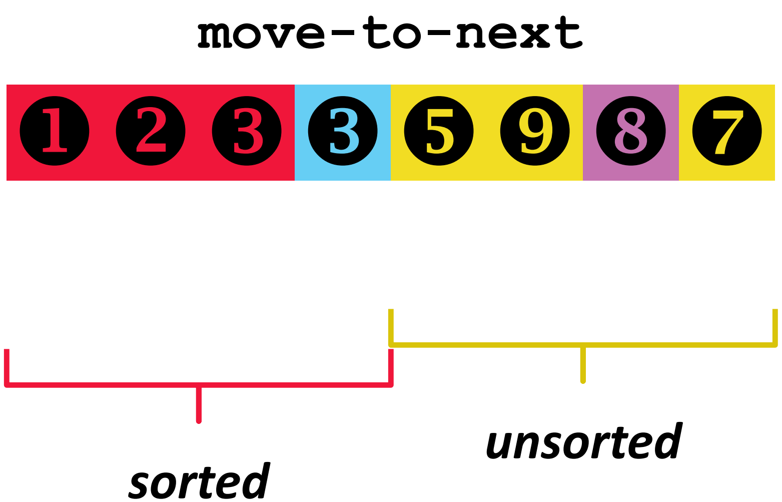
Find **the** ***Next Smallest*** **and** ***Swap*** **it with the** ***Second Position*,**

...

Find **the** ***Second-to-Last Smallest*** **and** ***Swap*** **it with the** ***Second Last Position*,**

the ***Largest*** **is** ***Already*** **in the** ***Last Position***

Example



Selection Sort

Find the Smallest and Swap it with the First Position,

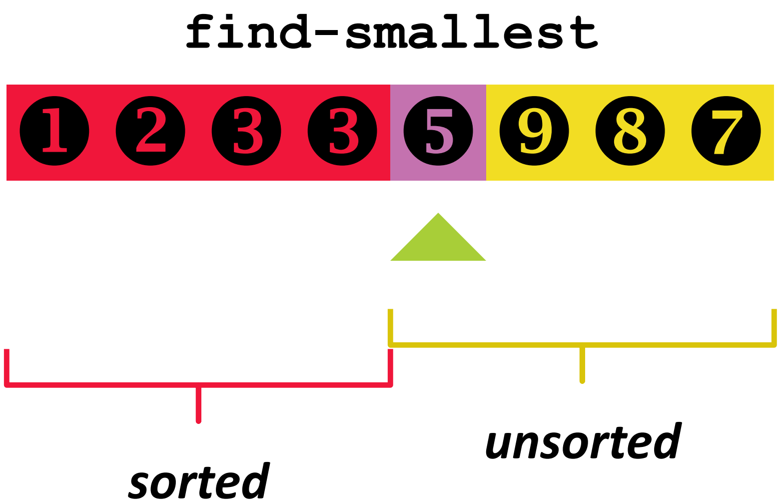
Find the Next Smallest and Swap it with the Second Position,

...

Find the Second-to-Last Smallest and Swap it with the Second Last Position,

the Largest is Already in the Last Position

Example



Selection Sort

Find the Smallest and Swap it with the First Position,

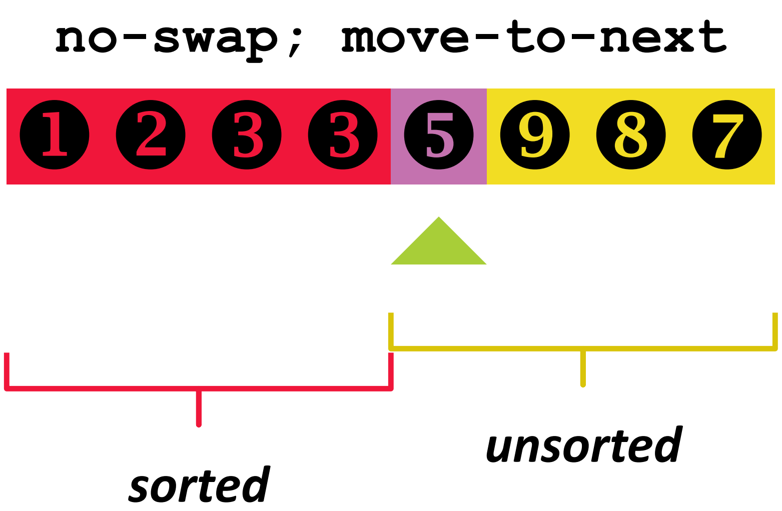
Find the Next Smallest and Swap it with the Second Position,

...

Find the Second-to-Last Smallest and Swap it with the Second Last Position,

the Largest is Already in the Last Position

Example



Selection Sort

Find the Smallest and Swap it with the First Position,

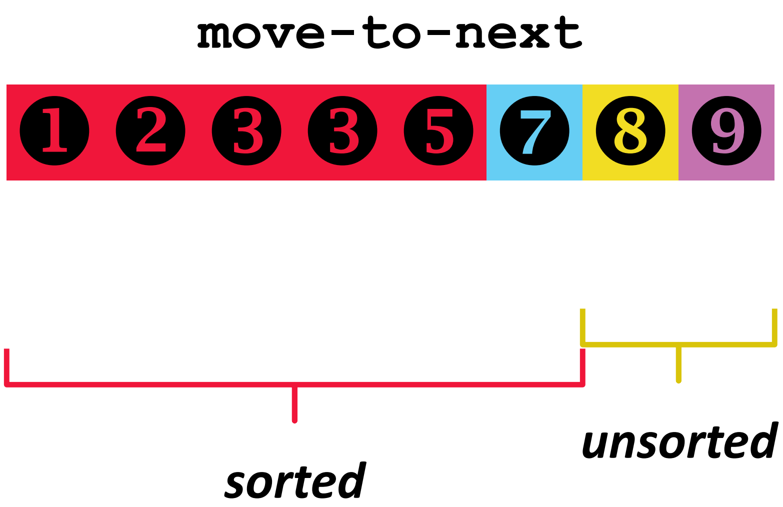
Find the Next Smallest and Swap it with the Second Position,

...

Find the Second-to-Last Smallest and Swap it with the Second Last Position,

the Largest is Already in the Last Position

Example



Selection Sort

***Find** the **Smallest** and **Swap** it with the **First Position**,*

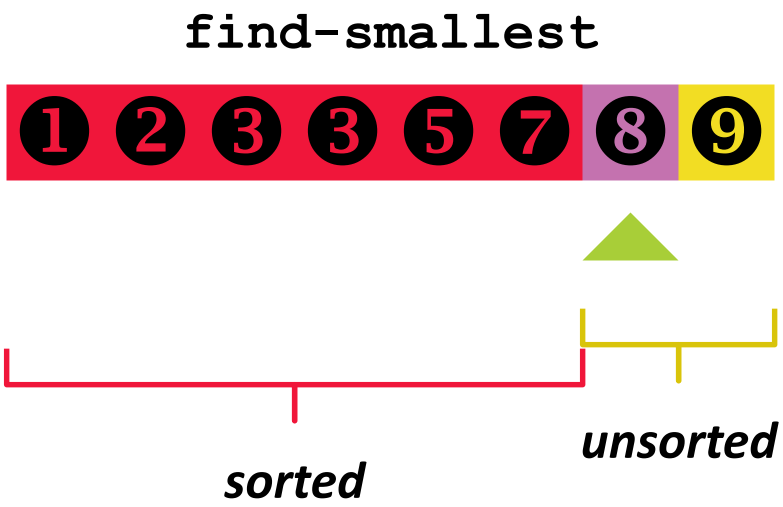
***Find** the **Next Smallest** and **Swap** it with the **Second Position**,*

...

***Find** the **Second-to-Last Smallest** and **Swap** it with the **Second Last Position**,*

the Largest** is **Already** in the **Last Position

Example



Selection Sort

Find the Smallest and Swap it with the First Position,

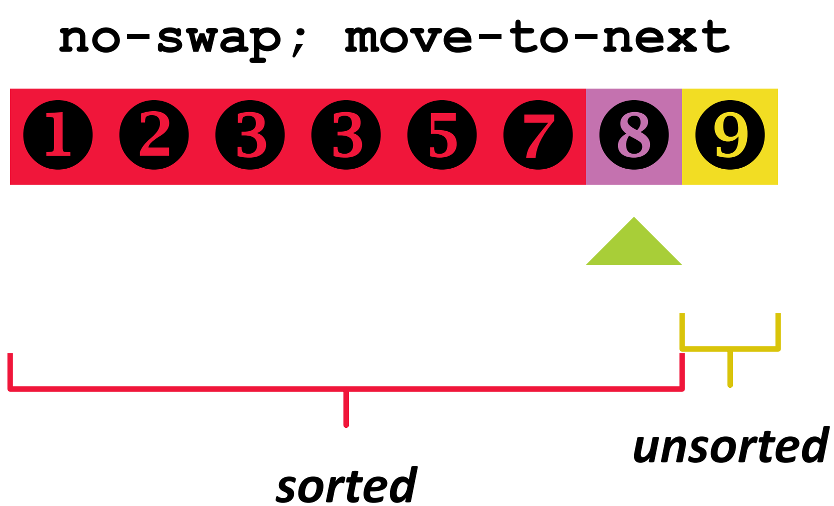
Find the Next Smallest and Swap it with the Second Position,

...

Find the Second-to-Last Smallest and Swap it with the Second Last Position,

the Largest is Already in the Last Position

Example



Selection Sort

***Find** the **Smallest** and **Swap** it with the **First Position**,*

***Find** the **Next Smallest** and **Swap** it with the **Second Position**,*

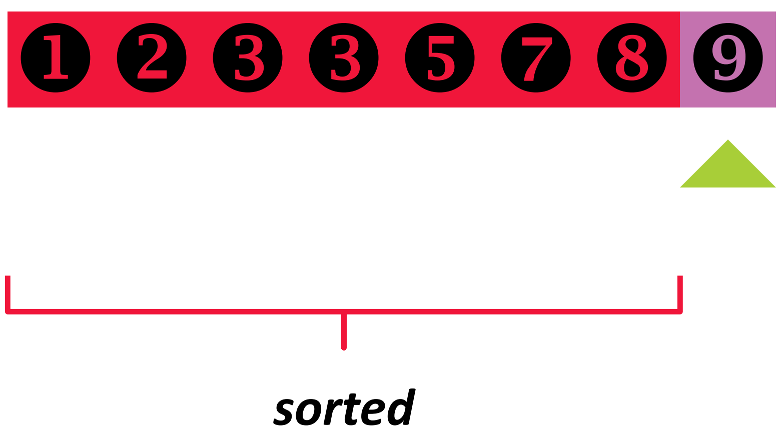
...

***Find** the **Second-to-Last Smallest** and **Swap** it with the **Second Last Position**,*

*the **Largest** is **Already** in the **Last Position***

Example

`last-element; no-swap`



Selection Sort

***Find** the **Smallest** and **Swap** it with the **First Position**,*

***Find** the **Next Smallest** and **Swap** it with the **Second Position**,*

...

***Find** the **Second-to-Last Smallest** and **Swap** it with the **Second Last Position**,*

*the **Largest** is **Already** in the **Last Position***

Example

`finished.`

1	2	3	3	5	7	8	9

sorted

Bubble Sort

Compare** the **First Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

Compare** the **Next Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

…

Compare** the **Last Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

If Any Swaps** were performed, **Reset the Algorithm** and **Start Again

```
n ← length
do:
    flag ← false
    for i ∊ [1, n):
        if (item[i-1] > item[i]):
            swap item[i-1] with item[i]
            flag ← true
    n--
while (flag)
```

Bubble Sort

Compare **the** ***First Pair*** ***of*** ***Adjacent Elements*** ***and*** ***Swap If*** ***they are*** ***Out of Order***

Compare **the** ***Next Pair*** ***of*** ***Adjacent Elements*** ***and*** ***Swap If*** ***they are*** ***Out of Order***

…

Compare **the** ***Last Pair*** ***of*** ***Adjacent Elements*** ***and*** ***Swap If*** ***they are*** ***Out of Order***

If Any Swaps ***were performed,*** ***Reset the Algorithm*** ***and*** ***Start Again***

Example

7	3	9	8	5	2	3	1

Bubble Sort

Compare** the **First Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

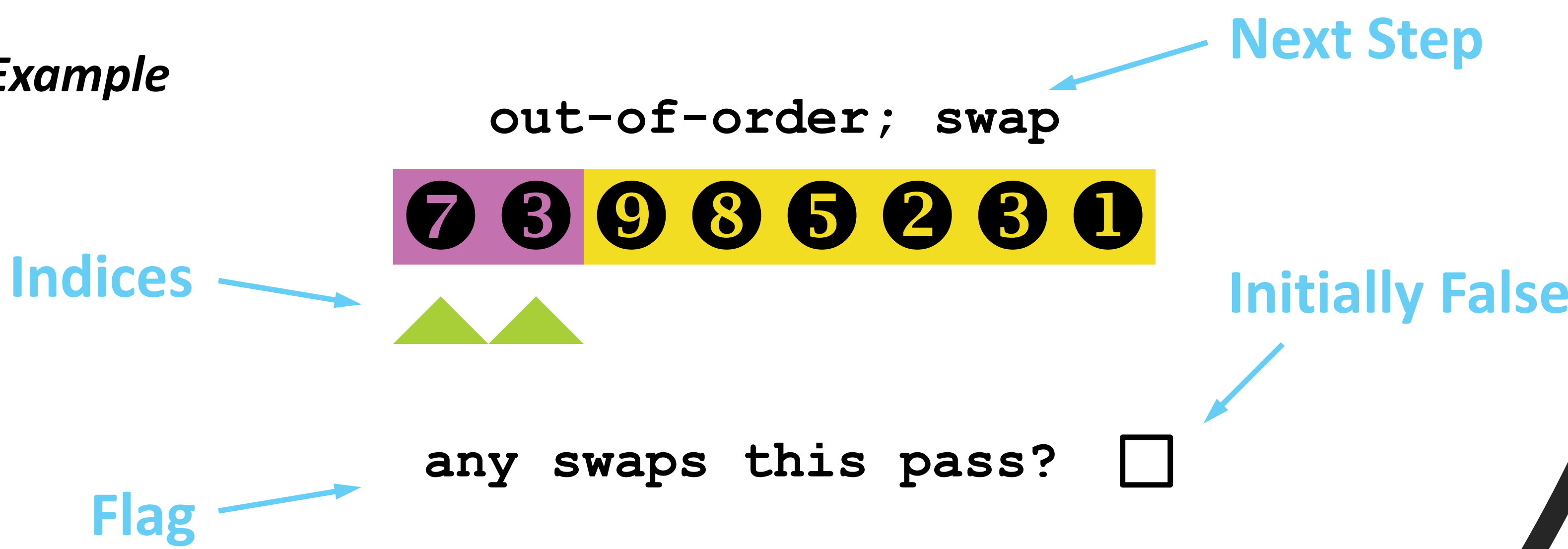
Compare** the **Next Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

...

Compare** the **Last Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

If Any Swaps** were performed, **Reset the Algorithm** and **Start Again

Example



Bubble Sort

Compare** the **First Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

Compare** the **Next Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

…

Compare** the **Last Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

If Any Swaps** were performed, **Reset the Algorithm** and **Start Again

Example

`move-to-next`

3	7	9	8	5	2	3	1

`any swaps this pass?` ☑

Bubble Sort

Compare** the **First Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

Compare** the **Next Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

...

Compare** the **Last Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

If Any Swaps** were performed, **Reset the Algorithm** and **Start Again

Example

`no-swap; move-to-next`

`any swaps this pass?`

Bubble Sort

Compare** the **First Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

Compare** the **Next Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

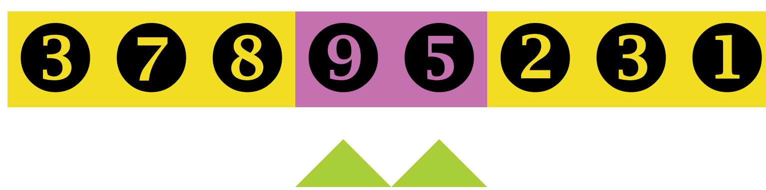
…

Compare** the **Last Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

If Any Swaps** were performed, **Reset the Algorithm** and **Start Again

Example

out-of-order; swap



any swaps this pass?

Bubble Sort

Compare** the **First Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

Compare** the **Next Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

…

Compare** the **Last Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

If Any Swaps** were performed, **Reset the Algorithm** and **Start Again

Example

`move-to-next`

3 7 8 5 9 2 3 1

`any swaps this pass?` ☑

Bubble Sort

Compare the First Pair of Adjacent Elements and Swap If they are Out of Order

Compare the Next Pair of Adjacent Elements and Swap If they are Out of Order

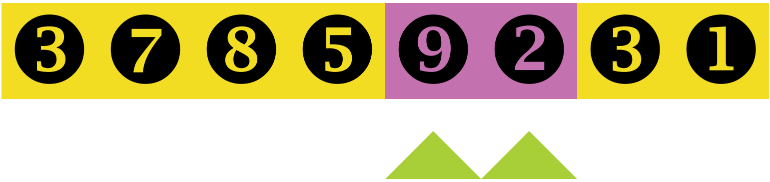
...

Compare the Last Pair of Adjacent Elements and Swap If they are Out of Order

If Any Swaps were performed, Reset the Algorithm and Start Again

Example

`out-of-order; swap`



`any swaps this pass?` ☑

Bubble Sort

Compare ***the*** ***First Pair*** ***of*** ***Adjacent Elements*** ***and*** ***Swap If*** ***they are*** ***Out of Order***

Compare ***the*** ***Next Pair*** ***of*** ***Adjacent Elements*** ***and*** ***Swap If*** ***they are*** ***Out of Order***

...

Compare ***the*** ***Last Pair*** ***of*** ***Adjacent Elements*** ***and*** ***Swap If*** ***they are*** ***Out of Order***

If Any Swaps ***were performed,*** ***Reset the Algorithm*** ***and*** ***Start Again***

Example

move-to-next

3 7 8 5 2 9 3 1

any swaps this pass? ☑

Bubble Sort

Compare* the *First Pair* of *Adjacent Elements* and *Swap If* they are *Out of Order

Compare* the *Next Pair* of *Adjacent Elements* and *Swap If* they are *Out of Order

...

Compare* the *Last Pair* of *Adjacent Elements* and *Swap If* they are *Out of Order

If Any Swaps* were performed, *Reset the Algorithm* and *Start Again

Example

out-of-order; swap

any swaps this pass? ☑

Bubble Sort

Compare the First Pair of Adjacent Elements and Swap If they are Out of Order

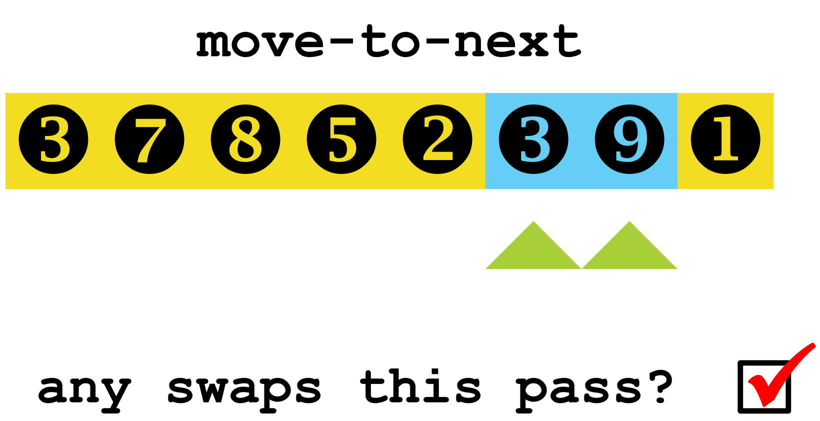
Compare the Next Pair of Adjacent Elements and Swap If they are Out of Order

...

Compare the Last Pair of Adjacent Elements and Swap If they are Out of Order

If Any Swaps were performed, Reset the Algorithm and Start Again

Example



Bubble Sort

Compare **the** ***First Pair*** ***of*** ***Adjacent Elements*** ***and*** ***Swap If*** ***they are*** ***Out of Order***

Compare **the** ***Next Pair*** ***of*** ***Adjacent Elements*** ***and*** ***Swap If*** ***they are*** ***Out of Order***

…

Compare **the** ***Last Pair*** ***of*** ***Adjacent Elements*** ***and*** ***Swap If*** ***they are*** ***Out of Order***

If Any Swaps ***were performed,*** ***Reset the Algorithm*** ***and*** ***Start Again***

Example

out-of-order; swap

any swaps this pass? ☑

Bubble Sort

Compare the First Pair of Adjacent Elements and Swap If they are Out of Order

Compare the Next Pair of Adjacent Elements and Swap If they are Out of Order

...

Compare the Last Pair of Adjacent Elements and Swap If they are Out of Order

If Any Swaps were performed, Reset the Algorithm and Start Again

Example

```
any-swaps? yes
```

3 7 8 5 2 3 1 9

```
any swaps this pass?
```

Bubble Sort

Compare the First Pair of Adjacent Elements and Swap If they are Out of Order

Compare the Next Pair of Adjacent Elements and Swap If they are Out of Order

...

Compare the Last Pair of Adjacent Elements and Swap If they are Out of Order

If Any Swaps were performed, Reset the Algorithm and Start Again

Example

`reset-the-algorithm`

`any swaps this pass?` ☐

Bubble Sort

Compare** the **First Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

Compare** the **Next Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

...

Compare** the **Last Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

If Any Swaps** were performed, **Reset the Algorithm** and **Start Again

Example

`no-swap; move-to-next`

3 7 8 5 2 3 1 9

`any swaps this pass?` ☐

Bubble Sort

Compare** the **First Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

Compare** the **Next Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

...

Compare** the **Last Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

If Any Swaps** were performed, **Reset the Algorithm** and **Start Again

Example

`no-swap; move-to-next`

`any swaps this pass?` ☐

Bubble Sort

Compare the First Pair of Adjacent Elements and Swap If they are Out of Order

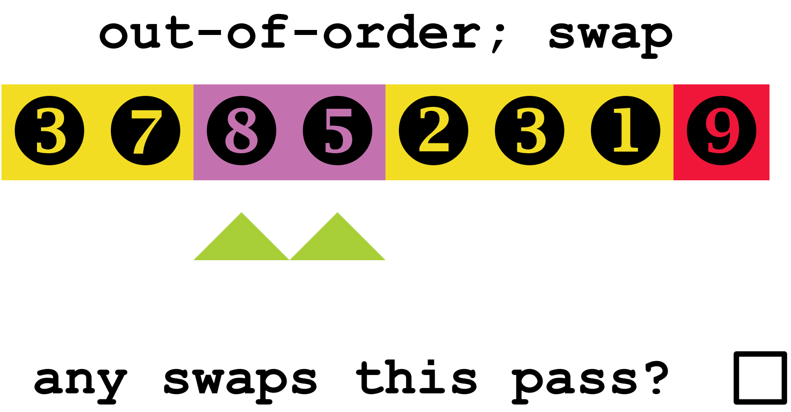
Compare the Next Pair of Adjacent Elements and Swap If they are Out of Order

...

Compare the Last Pair of Adjacent Elements and Swap If they are Out of Order

If Any Swaps were performed, Reset the Algorithm and Start Again

Example



Bubble Sort

Compare **the** ***First Pair*** ***of*** ***Adjacent Elements*** ***and*** ***Swap If*** ***they are*** ***Out of Order***

Compare **the** ***Next Pair*** ***of*** ***Adjacent Elements*** ***and*** ***Swap If*** ***they are*** ***Out of Order***

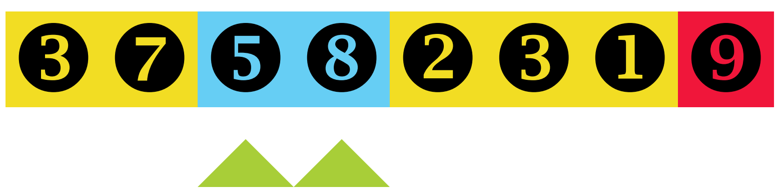
...

Compare **the** ***Last Pair*** ***of*** ***Adjacent Elements*** ***and*** ***Swap If*** ***they are*** ***Out of Order***

If Any Swaps ***were performed,*** ***Reset the Algorithm*** ***and*** ***Start Again***

Example

`move-to-next`



`any swaps this pass?` ☑

Bubble Sort

Compare the First Pair of Adjacent Elements and Swap If they are Out of Order

Compare the Next Pair of Adjacent Elements and Swap If they are Out of Order

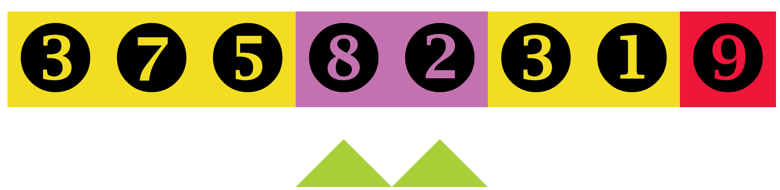
…

Compare the Last Pair of Adjacent Elements and Swap If they are Out of Order

If Any Swaps were performed, Reset the Algorithm and Start Again

Example

`out-of-order; swap`



`any swaps this pass?` ☑

Bubble Sort

Compare** the **First Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

Compare** the **Next Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

...

Compare** the **Last Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

If Any Swaps** were performed, **Reset the Algorithm** and **Start Again

Example

`move-to-next`

3	7	5	2	8	3	1	9

`any swaps this pass?`

Bubble Sort

Compare** the **First Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

Compare** the **Next Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

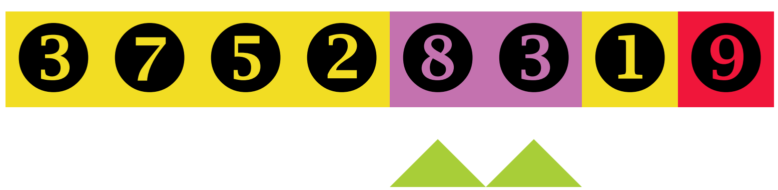
...

Compare** the **Last Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

If Any Swaps** were performed, **Reset the Algorithm** and **Start Again

Example

`out-of-order; swap`



`any swaps this pass?`

Bubble Sort

Compare** the **First Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

Compare** the **Next Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

...

Compare** the **Last Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

If Any Swaps** were performed, **Reset the Algorithm** and **Start Again

Example

`move-to-next`

3	7	5	2	3	8	1	9

`any swaps this pass?`

Bubble Sort

Compare ***the*** ***First Pair*** ***of*** ***Adjacent Elements*** ***and*** ***Swap If*** ***they are*** ***Out of Order***

Compare ***the*** ***Next Pair*** ***of*** ***Adjacent Elements*** ***and*** ***Swap If*** ***they are*** ***Out of Order***

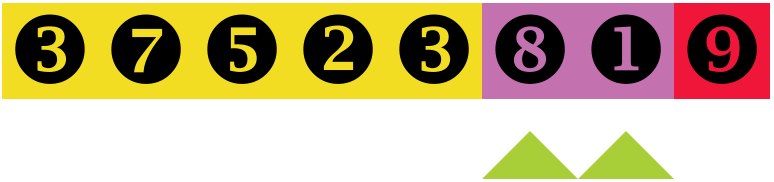
...

Compare ***the*** ***Last Pair*** ***of*** ***Adjacent Elements*** ***and*** ***Swap If*** ***they are*** ***Out of Order***

If Any Swaps ***were performed,*** ***Reset the Algorithm*** ***and*** ***Start Again***

Example

out-of-order; swap



any swaps this pass?

Bubble Sort

Compare** the **First Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

Compare** the **Next Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

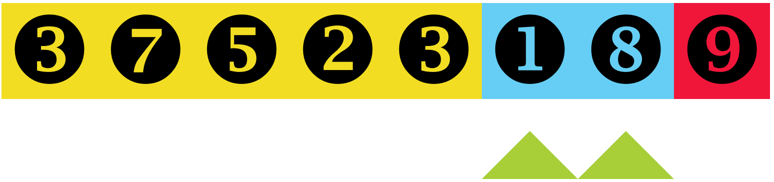
…

Compare** the **Last Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

If Any Swaps** were performed, **Reset the Algorithm** and **Start Again

Example

`move-to-next`



`any swaps this pass?` ☑

Bubble Sort

Compare** the **First Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

Compare** the **Next Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

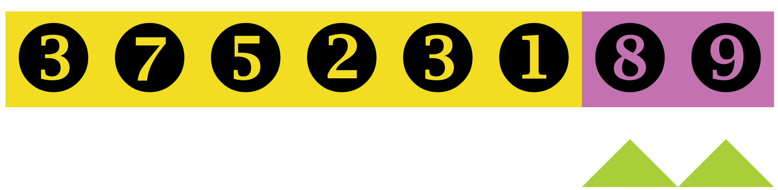
...

Compare** the **Last Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

If Any Swaps** were performed, **Reset the Algorithm** and **Start Again

Example

`no-swap; move-to-next`



`any swaps this pass?`

Bubble Sort

Compare **the *First Pair* of *Adjacent Elements* and *Swap If* they are *Out of Order***

Compare **the *Next Pair* of *Adjacent Elements* and *Swap If* they are *Out of Order***

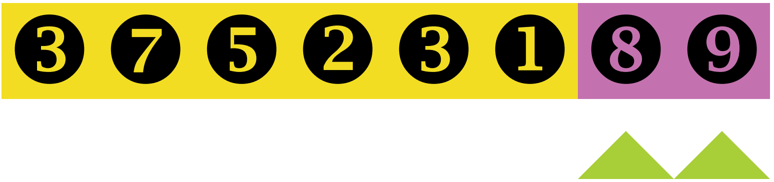
...

Compare **the *Last Pair* of *Adjacent Elements* and *Swap If* they are *Out of Order***

If Any Swaps **were performed, *Reset the Algorithm* and *Start Again***

Example

`any-swaps? yes`



`any swaps this pass?` ☑

Bubble Sort

Compare *the* ***First Pair*** *of* ***Adjacent Elements*** *and* ***Swap If*** *they are* ***Out of Order***

Compare *the* ***Next Pair*** *of* ***Adjacent Elements*** *and* ***Swap If*** *they are* ***Out of Order***

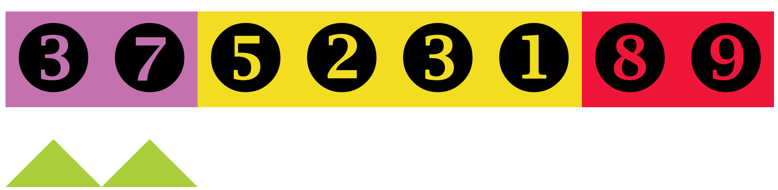
...

Compare *the* ***Last Pair*** *of* ***Adjacent Elements*** *and* ***Swap If*** *they are* ***Out of Order***

If Any Swaps *were performed,* ***Reset the Algorithm*** *and* ***Start Again***

Example

reset-the-algorithm



any swaps this pass? ☐

Bubble Sort

Compare the First Pair of Adjacent Elements and Swap If they are Out of Order

Compare the Next Pair of Adjacent Elements and Swap If they are Out of Order

...

Compare the Last Pair of Adjacent Elements and Swap If they are Out of Order

If Any Swaps were performed, Reset the Algorithm and Start Again

Example

```
no-swap; move-to-next
```

3	7	5	2	3	1	8	9

```
any swaps this pass? ☐
```

Bubble Sort

Compare ***the*** ***First Pair*** ***of*** ***Adjacent Elements*** ***and*** ***Swap If*** ***they are*** ***Out of Order***

Compare ***the*** ***Next Pair*** ***of*** ***Adjacent Elements*** ***and*** ***Swap If*** ***they are*** ***Out of Order***

…

Compare ***the*** ***Last Pair*** ***of*** ***Adjacent Elements*** ***and*** ***Swap If*** ***they are*** ***Out of Order***

If Any Swaps ***were performed,*** ***Reset the Algorithm*** ***and*** ***Start Again***

Example

out-of-order; swap

3 7 5 2 3 1 8 9

any swaps this pass? ☐

Bubble Sort

Compare ***the*** ***First Pair*** ***of*** ***Adjacent Elements*** ***and*** ***Swap If*** ***they are*** ***Out of Order***

Compare ***the*** ***Next Pair*** ***of*** ***Adjacent Elements*** ***and*** ***Swap If*** ***they are*** ***Out of Order***

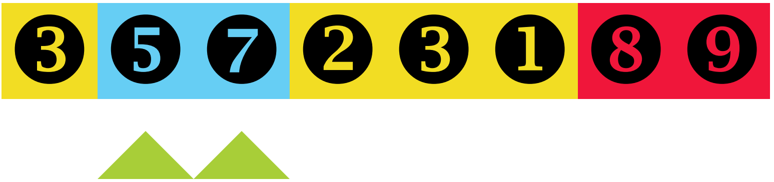
...

Compare ***the*** ***Last Pair*** ***of*** ***Adjacent Elements*** ***and*** ***Swap If*** ***they are*** ***Out of Order***

If Any Swaps ***were performed,*** ***Reset the Algorithm*** ***and*** ***Start Again***

Example

move-to-next



any swaps this pass? ☑

Bubble Sort

Compare** the **First Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

Compare** the **Next Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

...

Compare** the **Last Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

If Any Swaps** were performed, **Reset the Algorithm** and **Start Again

Example

`out-of-order; swap`

3	5	7	2	3	1	8	9

`any swaps this pass?`

Bubble Sort

Compare** the **First Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

Compare** the **Next Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

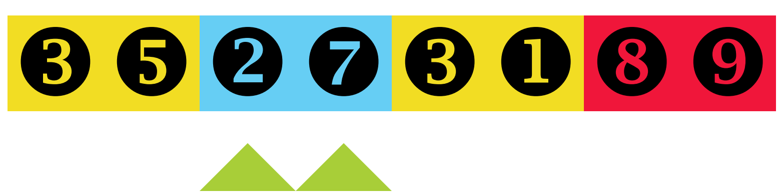
...

Compare** the **Last Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

If Any Swaps** were performed, **Reset the Algorithm** and **Start Again

Example

`move-to-next`



`any swaps this pass?`

Bubble Sort

Compare** the **First Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

Compare** the **Next Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

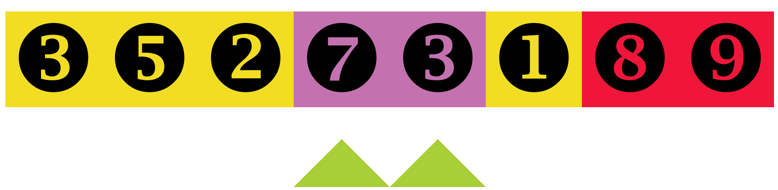
...

Compare** the **Last Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

If Any Swaps** were performed, **Reset the Algorithm** and **Start Again

Example

out-of-order; swap



any swaps this pass? ☑

Bubble Sort

Compare** the **First Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

Compare** the **Next Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

...

Compare** the **Last Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

If Any Swaps** were performed, **Reset the Algorithm** and **Start Again

Example

move-to-next

3	5	2	3	7	1	8	9

any swaps this pass? ☑

Bubble Sort

Compare** the **First Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

Compare** the **Next Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

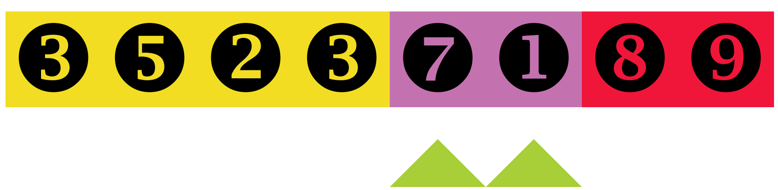
...

Compare** the **Last Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

If Any Swaps** were performed, **Reset the Algorithm** and **Start Again

Example

`out-of-order; swap`



`any swaps this pass?` ☑

Bubble Sort

Compare** the **First Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

Compare** the **Next Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

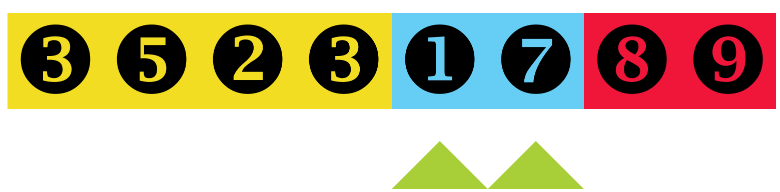
…

Compare** the **Last Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

If Any Swaps** were performed, **Reset the Algorithm** and **Start Again

Example

`move-to-next`



`any swaps this pass?` ☑

Bubble Sort

Compare* the *First Pair* of *Adjacent Elements* and *Swap If* they are *Out of Order

Compare* the *Next Pair* of *Adjacent Elements* and *Swap If* they are *Out of Order

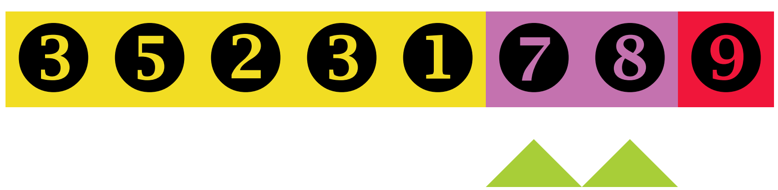
...

Compare* the *Last Pair* of *Adjacent Elements* and *Swap If* they are *Out of Order

If Any Swaps* were performed, *Reset the Algorithm* and *Start Again

Example

`no-swap; move-to-next`



`any swaps this pass?` ☑

Bubble Sort

Compare the First Pair of Adjacent Elements and Swap If they are Out of Order

Compare the Next Pair of Adjacent Elements and Swap If they are Out of Order

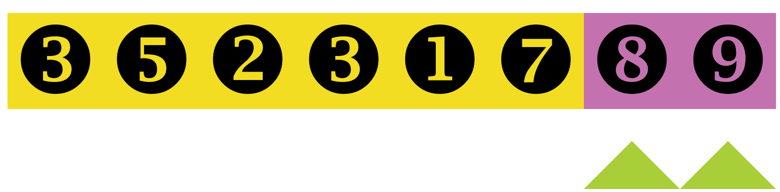
…

Compare the Last Pair of Adjacent Elements and Swap If they are Out of Order

If Any Swaps were performed, Reset the Algorithm and Start Again

Example

`no-swap; move-to-next`



`any swaps this pass?`

Bubble Sort

Compare** the **First Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

Compare** the **Next Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

...

Compare** the **Last Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

If Any Swaps** were performed, **Reset the Algorithm** and **Start Again

Example

any-swaps? yes

3	5	2	3	1	7	8	9

any swaps this pass? ☑

Bubble Sort

Compare** the **First Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

Compare** the **Next Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

…

Compare** the **Last Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

If Any Swaps** were performed, **Reset the Algorithm** and **Start Again

Example

Fast Forward

reset-the-algorithm

3	5	2	3	1	7	8	9

any swaps this pass? ☐

Bubble Sort

Compare the First Pair of Adjacent Elements and Swap If they are Out of Order

Compare the Next Pair of Adjacent Elements and Swap If they are Out of Order

…

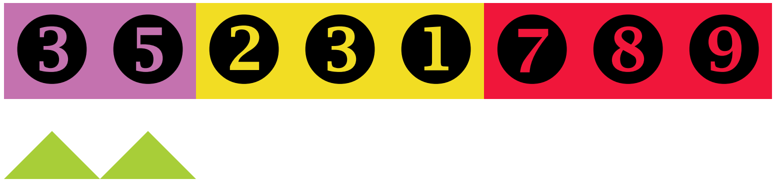
Compare the Last Pair of Adjacent Elements and Swap If they are Out of Order

If Any Swaps were performed, Reset the Algorithm and Start Again

Example

Fast Forward

`no-swap; move-to-next`



`any swaps this pass?` ☐

Bubble Sort

Compare** the **First Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

Compare** the **Next Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

...

Compare** the **Last Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

If Any Swaps** were performed, **Reset the Algorithm** and **Start Again

Example

Fast Forward

`swap; move-to-next`

3	2	5	3	1	7	8	9

`any swaps this pass?` ☑

Bubble Sort

Compare** the **First Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

Compare** the **Next Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

...

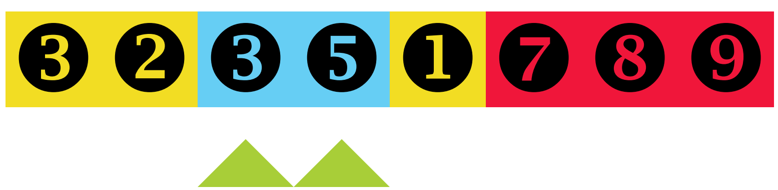
Compare** the **Last Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

If Any Swaps** were performed, **Reset the Algorithm** and **Start Again

Example

Fast Forward

`swap; move-to-next`



`any swaps this pass?` ☑

Bubble Sort

Compare** the **First Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

Compare** the **Next Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

...

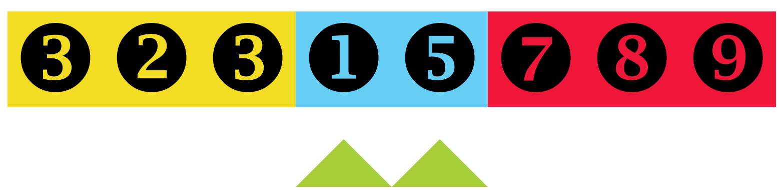
Compare** the **Last Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

If Any Swaps** were performed, **Reset the Algorithm** and **Start Again

Fast Forward

Example

```
swap; move-to-next
```




```
any swaps this pass?
```

Bubble Sort

Compare the First Pair of Adjacent Elements and Swap If they are Out of Order

Compare the Next Pair of Adjacent Elements and Swap If they are Out of Order

...

Compare the Last Pair of Adjacent Elements and Swap If they are Out of Order

If Any Swaps were performed, Reset the Algorithm and Start Again

Example

Fast Forward

`no-swap; move-to-next`

3	2	3	1	5	7	8	9

`any swaps this pass?` ☑

Bubble Sort

Compare the First Pair of Adjacent Elements and Swap If they are Out of Order

Compare the Next Pair of Adjacent Elements and Swap If they are Out of Order

...

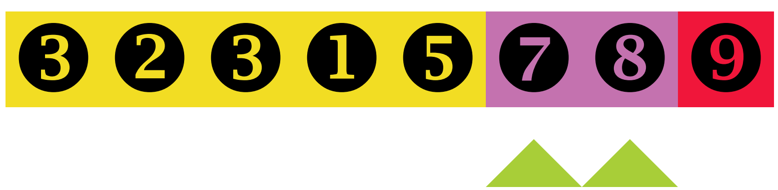
Compare the Last Pair of Adjacent Elements and Swap If they are Out of Order

If Any Swaps were performed, Reset the Algorithm and Start Again

Example

Fast Forward

`no-swap; move-to-next`



`any swaps this pass?`

Bubble Sort

Compare** the **First Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

Compare** the **Next Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

...

Compare** the **Last Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

If Any Swaps** were performed, **Reset the Algorithm** and **Start Again

Example

Fast Forward

`no-swap; move-to-next`

3 2 3 1 5 7 8 9

`any swaps this pass?` ☑

Bubble Sort

Compare** the **First Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

Compare** the **Next Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

...

Compare** the **Last Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

If Any Swaps** were performed, **Reset the Algorithm** and **Start Again

Example

Fast Forward

`any-swaps? yes`

`any swaps this pass?` ☑

Bubble Sort

Compare the First Pair of Adjacent Elements and Swap If they are Out of Order

Compare the Next Pair of Adjacent Elements and Swap If they are Out of Order

...

Compare the Last Pair of Adjacent Elements and Swap If they are Out of Order

If Any Swaps were performed, Reset the Algorithm and Start Again

Example

Fast Forward

`reset-the-algorithm`

3	2	3	1	5	7	8	9

`any swaps this pass?` ☐

Bubble Sort

Compare the First Pair of Adjacent Elements and Swap If they are Out of Order

Compare the Next Pair of Adjacent Elements and Swap If they are Out of Order

...

Compare the Last Pair of Adjacent Elements and Swap If they are Out of Order

If Any Swaps were performed, Reset the Algorithm and Start Again

Example

Fast Forward

`swap; move-to-next`

2	3	3	1	5	7	8	9

`any swaps this pass?` ☑

Bubble Sort

Compare* the *First Pair* of *Adjacent Elements* and *Swap If* they are *Out of Order

Compare* the *Next Pair* of *Adjacent Elements* and *Swap If* they are *Out of Order

...

Compare* the *Last Pair* of *Adjacent Elements* and *Swap If* they are *Out of Order

If Any Swaps* were performed, *Reset the Algorithm* and *Start Again

Example

Fast Forward

`no-swap; move-to-next`

`any swaps this pass?`

Bubble Sort

Compare ***the*** ***First Pair*** ***of*** ***Adjacent Elements*** ***and*** ***Swap If*** ***they are*** ***Out of Order***

Compare ***the*** ***Next Pair*** ***of*** ***Adjacent Elements*** ***and*** ***Swap If*** ***they are*** ***Out of Order***

...

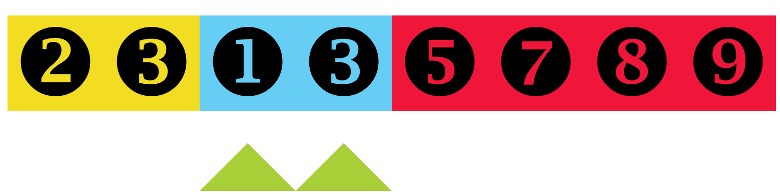
Compare ***the*** ***Last Pair*** ***of*** ***Adjacent Elements*** ***and*** ***Swap If*** ***they are*** ***Out of Order***

If Any Swaps ***were performed,*** ***Reset the Algorithm*** ***and*** ***Start Again***

Example

Fast Forward

`swap; move-to-next`



`any swaps this pass?` ☑

Bubble Sort

Compare** the **First Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

Compare** the **Next Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

...

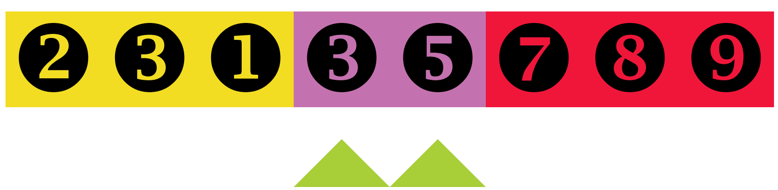
Compare** the **Last Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

If Any Swaps** were performed, **Reset the Algorithm** and **Start Again

Example

Fast Forward

`no-swap; move-to-next`



`any swaps this pass?` ☑

Bubble Sort

Compare** the **First Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

Compare** the **Next Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

…

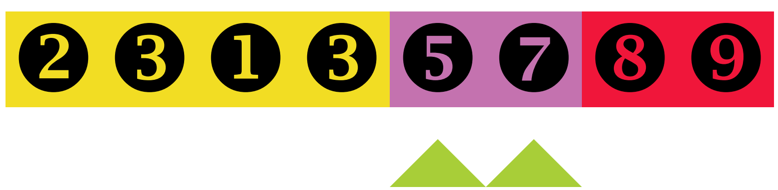
Compare** the **Last Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

If Any Swaps** were performed, **Reset the Algorithm** and **Start Again

Example

Fast Forward

`no-swap; move-to-next`



`any swaps this pass?` ☑

Bubble Sort

Compare ***the*** ***First Pair*** ***of*** ***Adjacent Elements*** ***and*** ***Swap If*** ***they are*** ***Out of Order***

Compare ***the*** ***Next Pair*** ***of*** ***Adjacent Elements*** ***and*** ***Swap If*** ***they are*** ***Out of Order***

...

Compare ***the*** ***Last Pair*** ***of*** ***Adjacent Elements*** ***and*** ***Swap If*** ***they are*** ***Out of Order***

If Any Swaps ***were performed,*** ***Reset the Algorithm*** ***and*** ***Start Again***

Example

Fast Forward

`no-swap; move-to-next`

`any swaps this pass?` ☑

Bubble Sort

Compare** the **First Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

Compare** the **Next Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

...

Compare** the **Last Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

If Any Swaps** were performed, **Reset the Algorithm** and **Start Again

Example

Fast Forward

```
no-swap; move-to-next
```

2 3 1 3 5 7 8 9

```
any swaps this pass?
```
☑

Bubble Sort

Compare** the **First Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

Compare** the **Next Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

...

Compare** the **Last Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

If Any Swaps** were performed, **Reset the Algorithm** and **Start Again

Example

Fast Forward

`any-swaps? yes`

2 3 1 3 5 7 8 9

`any swaps this pass?` ☑

Bubble Sort

Compare ***the*** ***First Pair*** ***of*** ***Adjacent Elements*** ***and*** ***Swap If*** ***they are*** ***Out of Order***

Compare ***the*** ***Next Pair*** ***of*** ***Adjacent Elements*** ***and*** ***Swap If*** ***they are*** ***Out of Order***

...

Compare ***the*** ***Last Pair*** ***of*** ***Adjacent Elements*** ***and*** ***Swap If*** ***they are*** ***Out of Order***

If Any Swaps ***were performed,*** ***Reset the Algorithm*** ***and*** ***Start Again***

Example

Fast Forward

`reset-the-algorithm`

`any swaps this pass?` ☐

Bubble Sort

Compare the First Pair of Adjacent Elements and Swap If they are Out of Order

Compare the Next Pair of Adjacent Elements and Swap If they are Out of Order

...

Compare the Last Pair of Adjacent Elements and Swap If they are Out of Order

If Any Swaps were performed, Reset the Algorithm and Start Again

Example

Fast Forward

`any swaps this pass?` ☑

Bubble Sort

Compare the First Pair of Adjacent Elements and Swap If they are Out of Order

Compare the Next Pair of Adjacent Elements and Swap If they are Out of Order

...

Compare the Last Pair of Adjacent Elements and Swap If they are Out of Order

If Any Swaps were performed, Reset the Algorithm and Start Again

Example

Fast Forward

1	2	3	3	5	7	8	9

`any swaps this pass?` ☑

Bubble Sort

Compare** the **First Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

Compare** the **Next Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

...

Compare** the **Last Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

If Any Swaps** were performed, **Reset the Algorithm** and **Start Again

Example

Fast Forward

1	2	3	3	5	7	8	9

`any swaps this pass?` ☐

Bubble Sort

Compare** the **First Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

Compare** the **Next Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

...

Compare** the **Last Pair** of **Adjacent Elements** and **Swap If** they are **Out of Order

If Any Swaps** were performed, **Reset the Algorithm** and **Start Again

Example

`finished.`

1	2	3	3	5	7	8	9

sorted

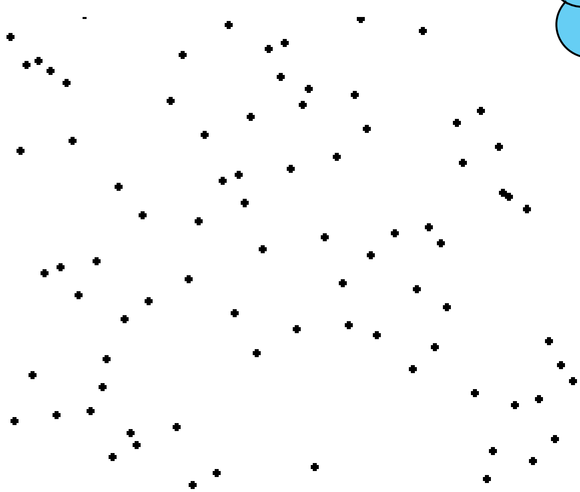
Selection Sort vs. Bubble Sort

Selection Sort and Bubble Sort are Very Different Algorithms

(Finding Minimum Values vs. Exchanging Adjacent Elements)

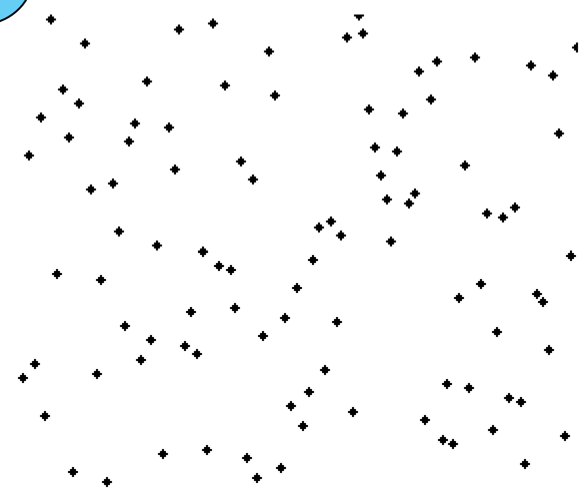
Sorting Random Pixels by Height

Selection Sort



en.wikipedia.org/wiki/User:Marco_Polo (Public Domain)

Bubble Sort



en.wikipedia.org/wiki/User:Nmnogueira (CC BY-SA 2.5)

How do they Compare in terms of Efficiency?

Analyzing Performance

a Fundamental Operation for Sorting Collections of Elements

is Accessing (i.e., Visiting) the Individual Elements

Analyzing Performance

a Fundamental Operation for Sorting Collections of Elements

is Accessing (i.e., Visiting) the Individual Elements

How Often does Selection Sort Visit an element?

How Often does Bubble Sort Visit an element?

Analyzing Performance

a Fundamental Operation for Sorting Collections of Elements

is Accessing (i.e., Visiting) the Individual Elements

How Often does Selection Sort Visit an element?

How Often does Bubble Sort Visit an element?

in Order to Produce a Generalizable Result

Express the Solution as a Function of the Size of the Collection

Analyzing Performance

we can Examine the Implementation of `swap` in the Source for java.util.Collections to see How Many `get`/`set` Calls are Made

```
/**
 * Swaps the elements at the specified positions within the list. Equal
 * positions have no effect.
 *
 * @param l the list to work on
 * @param i the first index to swap
 * @param j the second index
 * @throws UnsupportedOperationException if list.set is not supported
 * @throws IndexOutOfBoundsException if either i or j is &lt; 0 or &gt;=
 *         list.size()
 * @since 1.4
 */
public static void swap(List<?> l, int i, int j)
{
  List<Object> list = (List<Object>) l;
  list.set(i, list.set(j, list.get(i)));
}
```

Analyzing Selection Sort Performance

Every Outer Loop Requires

Swap and Finding Next Smallest

(i.e., (3 + 2(n-1)) + (3 + 2(n-2)) + ...)

```
for i ∈ [0, length-1):
    nextsmallestindex ← i
    for j ∈ [i+1, length):
        if (item[j] < item[nextsmallest]):
            nextsmallest ← j
    swap item[i] with item[nextsmallest]
```

$$= \sum_{i=0}^{n-2} (3 + 2(n-1-i))$$

$$= 3(n-1) \sum_{i=0}^{n-2} 2(n-1-i)$$

$$= 3(n-1) + \sum_{i=0}^{n-2} 2(n-1) - 2i$$

$$= 3(n-1) + (2(n-1)(n-1)) + \sum_{i=0}^{n-2} - 2i$$

$$= 3(n-1) + (2(n-1)(n-1)) - 2 \sum_{i=0}^{n-2} i$$

$$= 3(n-1) + (2(n-1)(n-1)) - 2 ((n-2)(n-1)/2)$$

$$= n^2 + 2n - 3$$

Analyzing Bubble Sort Performance

for an array of Length n:

```
n ← length
do:
    flag ← false
    for i ∊ [1, n):
        if (item[i-1] > item[i]):
            swap item[i-1] with item[i]
            flag ← true
    n--
while (flag)
```

What is the Maximum number of Do-While Loops? n

What is the Maximum number of Swaps on First Pass? n-1

What is the Maximum number of Swaps on Next Pass? n-2

(n.b., a Swap is still 3 Array Accesses)

Analyzing Bubble Sort Performance

Every Outer Loop Requires

that all Remaining Pairs are

Compared and Swapped

(i.e., (n-1) + (n-2) + (n-3) + ... + 1)

```
n ← length
do:
    flag ← false
    for i ∈ [1, n):
        if (item[i-1] > item[i]):
            swap item[i-1] with item[i]
            flag ← true
    n--
while (flag)
```

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

$$= 5 \left(\sum_{i=1}^{n-1} n - \sum_{i=1}^{n-1} i\right)$$

$$= 5 \left((n-1)(n) - \sum_{i=1}^{n-1} i\right)$$

$$= 5 ((n-1)(n) - ((n-1)(n)/2))$$

$$= 5 (((n-1)(n)/2))$$

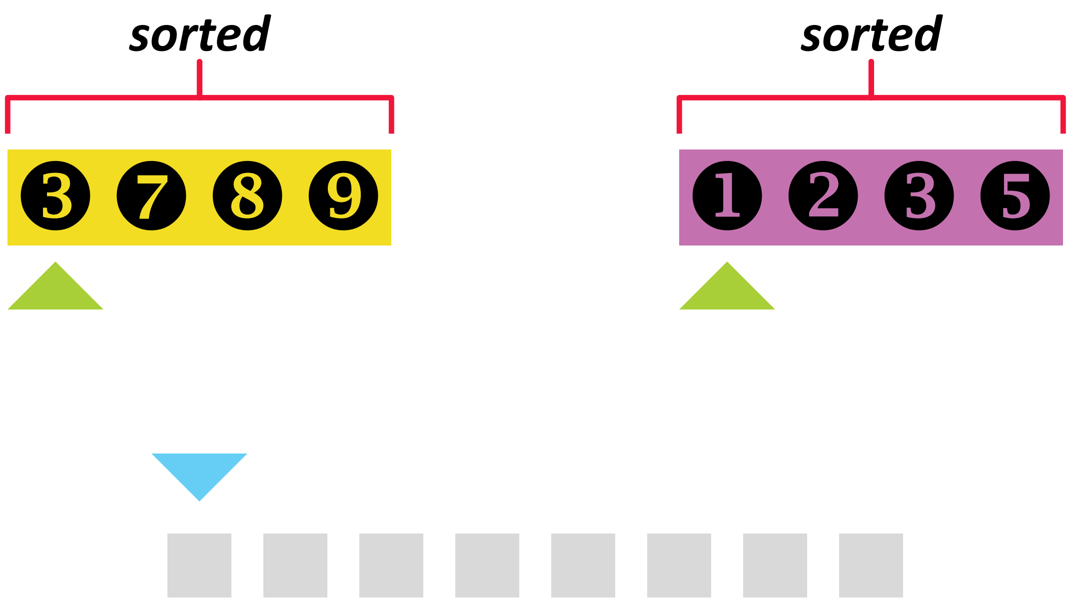
$$= {}^{5}/_{2} \, (n^2 - n)$$

Leading Question

If you were Provided Two Sorted Lists (of lengths x and y)

Could you Merge them into a Single Sorted List (of length x+y)?

Example



Separate Indices into the New List and the Sorted Sublists

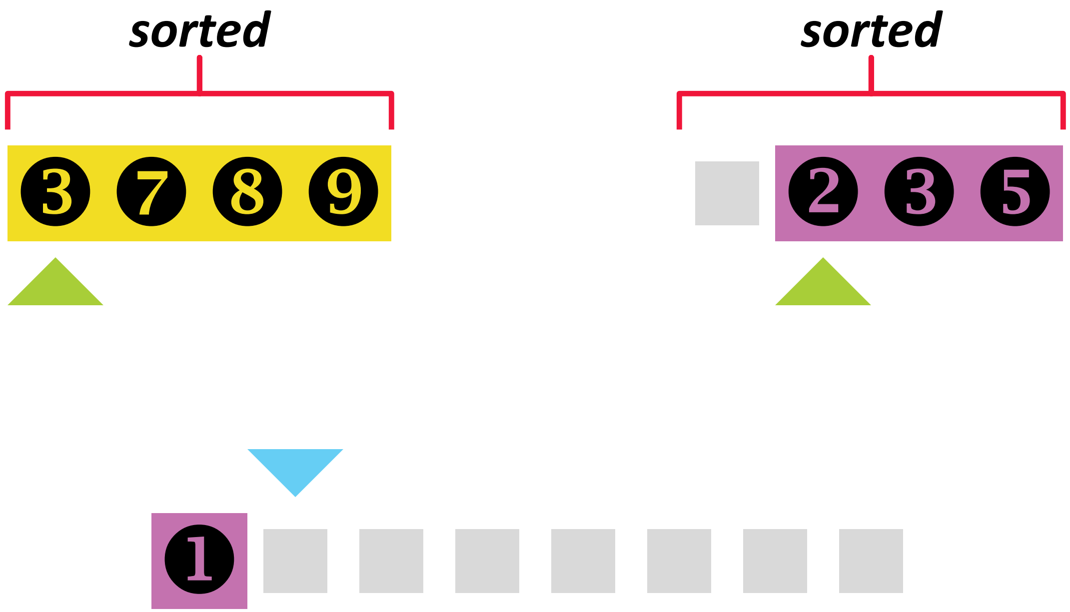
Copy the Lesser Sublist Value and Move that Index

Leading Question

If you were **Provided Two Sorted Lists** (of lengths x and y)

Could you **Merge** them into a **Single Sorted List** (of length x+y)?

Example



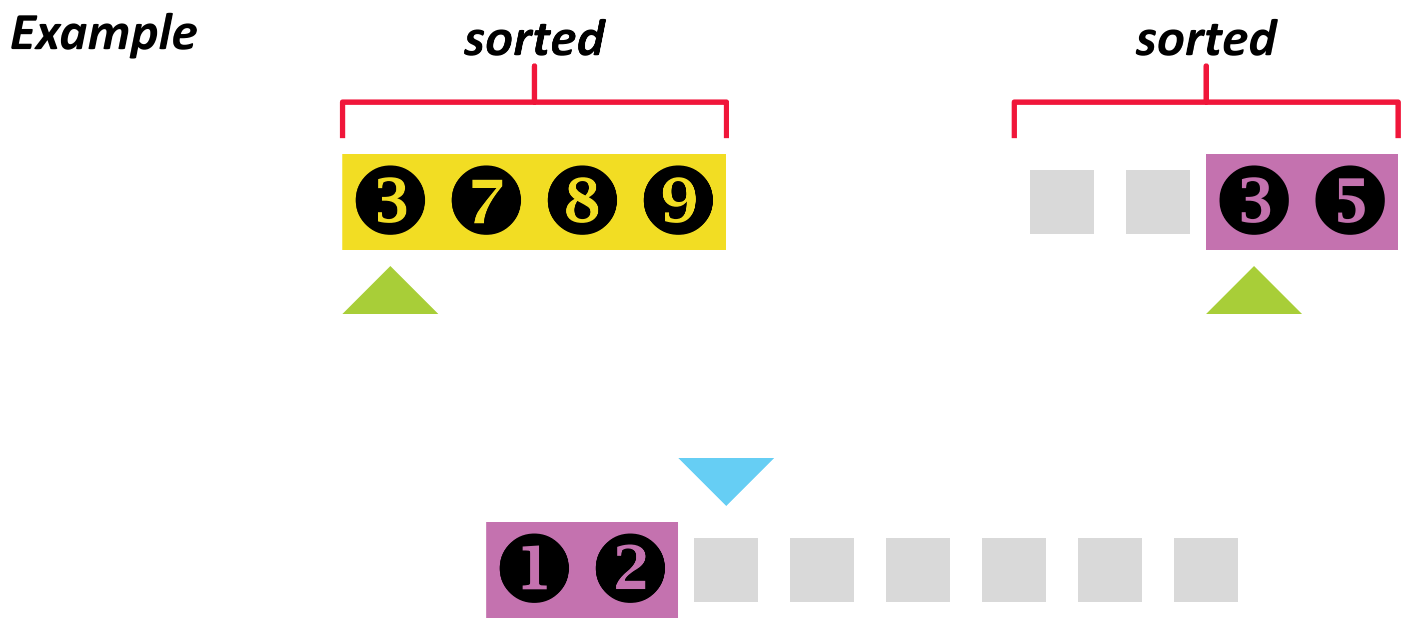
Separate Indices into the **New List** and the **Sorted Sublists**

Copy the **Lesser Sublist Value** and **Move** that **Index**

Leading Question

If you were **Provided Two Sorted Lists** (of lengths x and y)

Could you **Merge** them into a **Single Sorted List** (of length x+y)?



Separate Indices into the **New List** and the **Sorted Sublists**

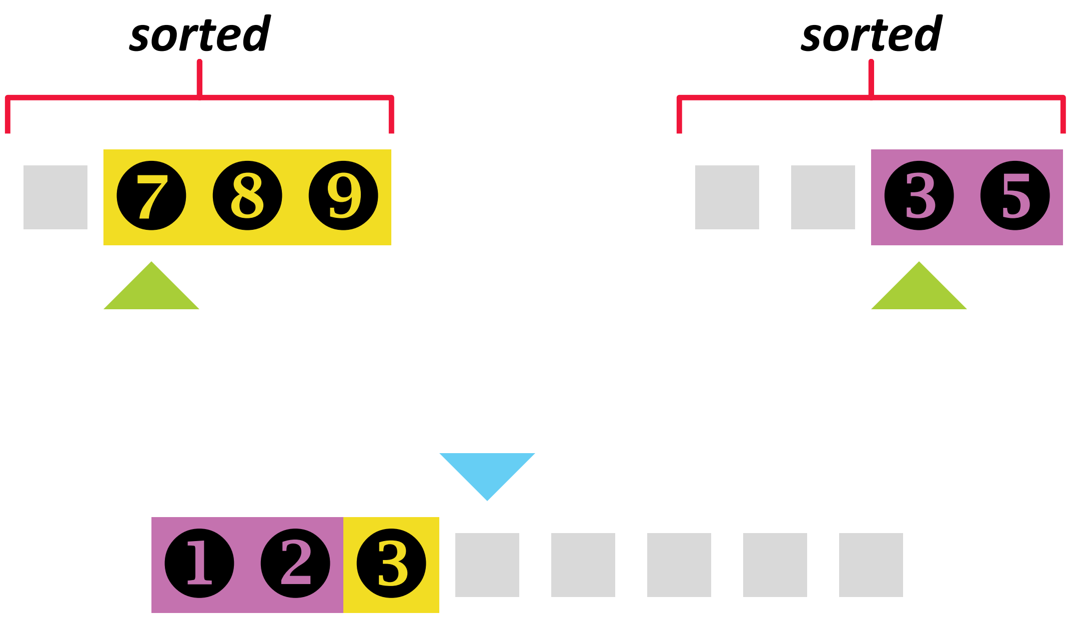
Copy the **Lesser Sublist Value** and **Move** that **Index**

Leading Question

If you were Provided Two Sorted Lists (of lengths x and y)

Could you Merge them into a Single Sorted List (of length x+y)?

Example



Separate Indices into the New List and the Sorted Sublists

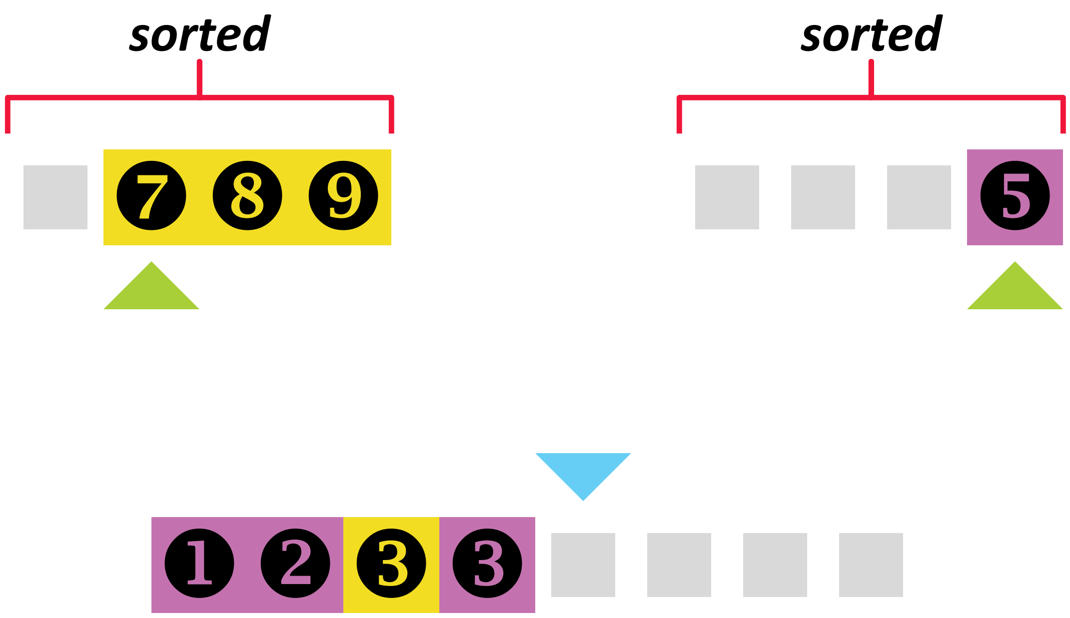
Copy the Lesser Sublist Value and Move that Index

Leading Question

If you were **Provided Two Sorted Lists** (of lengths x and y)

Could you **Merge** them into a **Single Sorted List** (of length x+y)?

Example



Separate Indices into the **New List** and the **Sorted Sublists**

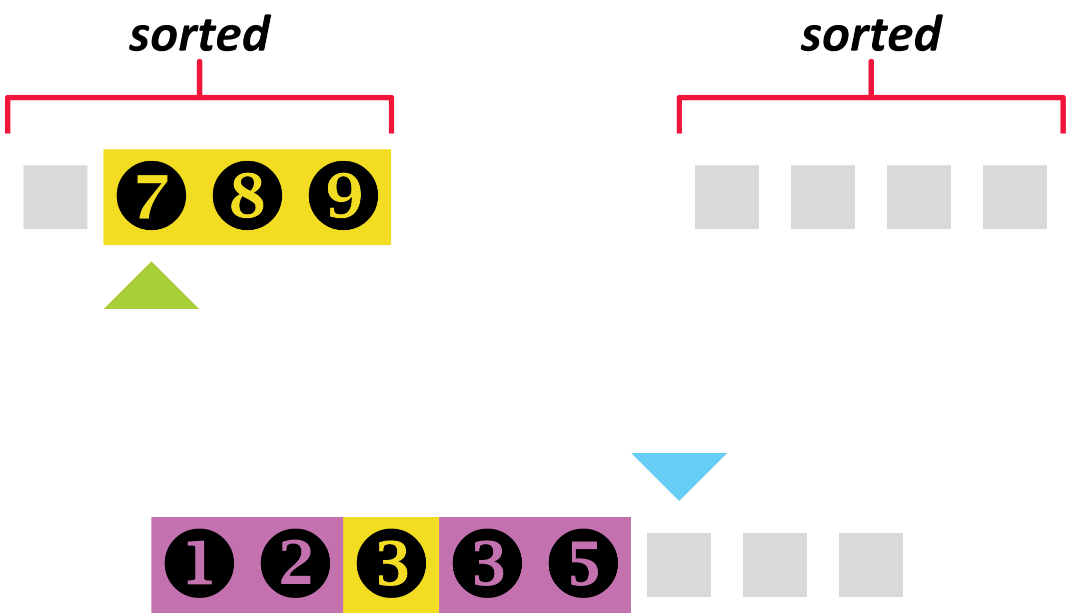
Copy the **Lesser Sublist Value** and **Move** that **Index**

Leading Question

If you were **Provided Two Sorted Lists** (of lengths x and y)

Could you **Merge** them into a **Single Sorted List** (of length x+y)?

Example



Separate Indices into the **New List** and the **Sorted Sublists**

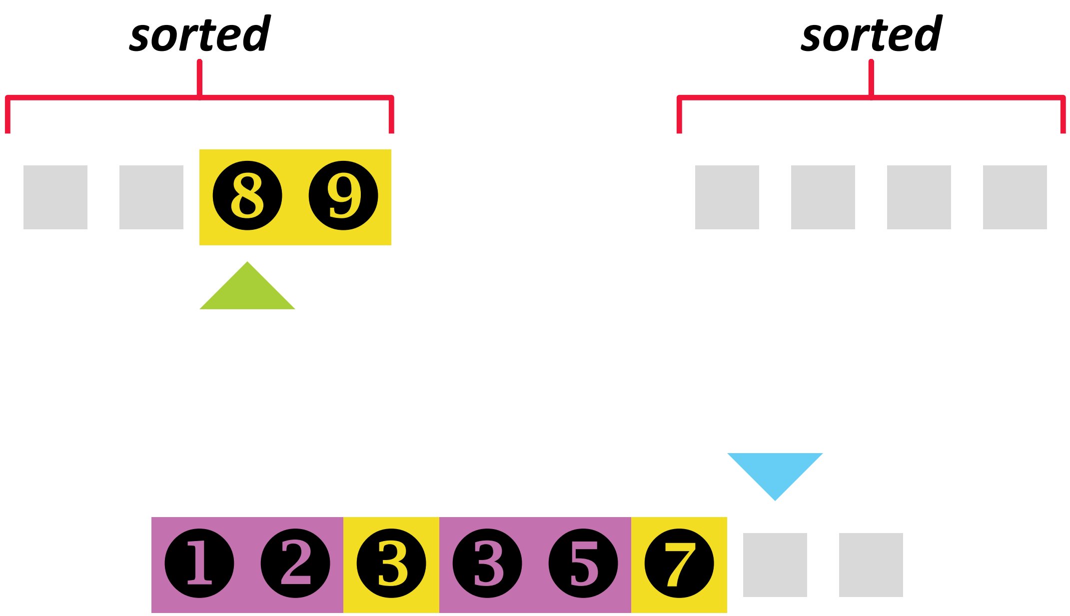
Copy the **Lesser Sublist Value** and **Move** that **Index**

Leading Question

If you were Provided Two Sorted Lists (of lengths x and y)

Could you Merge them into a Single Sorted List (of length x+y)?

Example



Separate Indices into the New List and the Sorted Sublists

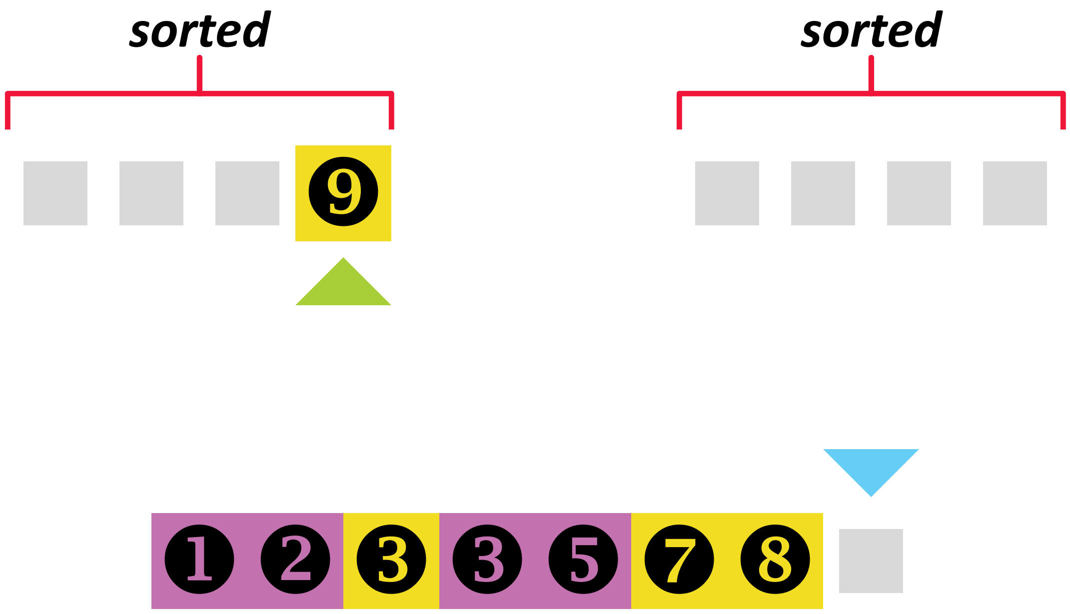
Copy the Lesser Sublist Value and Move that Index

Leading Question

If you were Provided Two Sorted Lists (of lengths x and y)

Could you Merge them into a Single Sorted List (of length x+y)?

Example



Separate Indices into the New List and the Sorted Sublists

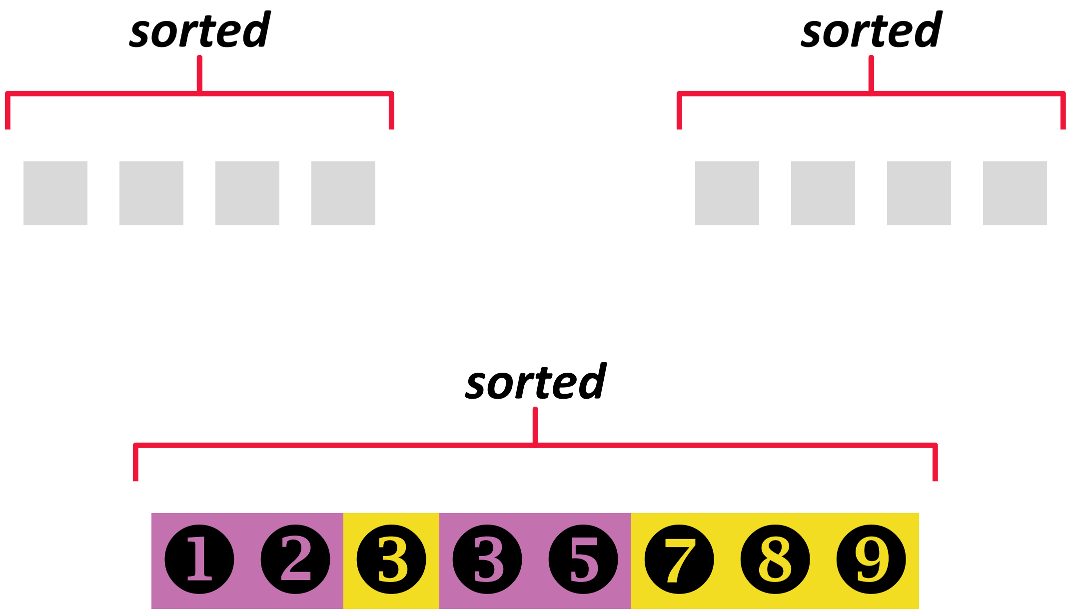
Copy the Lesser Sublist Value and Move that Index

Leading Question

If you were **Provided Two Sorted Lists** (of lengths x and y)

Could you **Merge** them into a **Single Sorted List** (of length x+y)?

Example



Separate Indices into the **New List** and the **Sorted Sublists**

Copy the **Lesser Sublist Value** and **Move** that **Index**

Additional Leading Questions

Two Unsorted Lists X and Y and the Length of X > the Length of Y

Which of the Two Lists is (in General) Easier to Sort?

How do you Sort a Singleton List (i.e., a List of Length 1)?

Additional Leading Questions

Two Unsorted Lists X and Y and the Length of X > the Length of Y

Which of the Two Lists is (in General) Easier to Sort?

How do you Sort a Singleton List (i.e., a List of Length 1)?

Procedure for Sorting a List:

Divide the Unsorted List (of Length L)

into Two Lists (of Length $\lceil L/2 \rceil$ and $\lfloor L/2 \rfloor$ respectively)

Sort the Sublists and then Merge them into a Single Sorted List

Additional Leading Questions

Two Unsorted Lists X and Y and the Length of X > the Length of Y

Which of the Two Lists is (in General) Easier to Sort?

How do you Sort a Singleton List (i.e., a List of Length 1)?

Procedure for Sorting a List:

Divide the Unsorted List (of Length L)

into Two Lists (of Length $\lceil L/2 \rceil$ and $\lfloor L/2 \rfloor$ respectively)

Sort the Sublists and then Merge them into a Single Sorted List

Can the Procedure Itself be Used to Solve this Subproblem?

Recursive Approach

Merge Sort

If ***the*** ***Unsorted List*** ***is of*** ***Length 1***, ***Return***

Otherwise, ***Divide*** ***the list in*** ***Half*** ***(approximately) into*** ***Two Sublists***

Recursively Call Merge Sort ***on*** ***Each Sublist*** ***and*** ***Merge*** ***the*** ***Return Values***

```
sort(item):
    if (len(item) ≤ 1):
        return item
    else
        left  ← sort(item[0:len(item)/2])
        right ← sort(item[len(item)/2:len(item)])
        return merge(left, right)
```

```
merge(left, right):
    j = 0, k = 0
    for i ∊ [0, len(left)+len(right)):
        if left[j] < right[k]:
            item[i] ← left[j]  ; j++
        else:
            item[i] ← right[k] ; k++
    return item
```

Merge Sort

Example ❼ ❸ ❾ ❽ ❺ ❷ ❸ ❶

```
sort(item[0:len(item)/2])
sort(item[len(item)/2:len(item)])
merge(left, right)
```

`sort(` ❼ ❸ ❾ ❽ ❺ ❷ ❸ ❶ `)`

Merge Sort

Example ❼ ❸ ❾ ❽ ❺ ❷ ❸ ❶

```
sort(item[0:len(item)/2])

sort(item[len(item)/2:len(item)])

merge(left, right)
```

```
sort( ❼ ❸ ❾ ❽ ❺ ❷ ❸ ❶ )

merge( sort( ❼ ❸ ❾ ❽ ),sort( ❺ ❷ ❸ ❶ ) )
```

Merge Sort

Example ❼ ❸ ❾ ❽ ❺ ❷ ❸ ❶

```
sort(item[0:len(item)/2])
sort(item[len(item)/2:len(item)])
merge(left, right)
```

sort(❼ ❸ ❾ ❽ ❺ ❷ ❸ ❶)

merge(sort(❼ ❸ ❾ ❽), sort(❺ ❷ ❸ ❶))

merge(sort(❼ ❸), sort(❾ ❽))

Merge Sort

Example ❼ ❸ ❾ ❽ ❺ ❷ ❸ ❶

```
sort(item[0:len(item)/2])
sort(item[len(item)/2:len(item)])
merge(left, right)
```

```
sort( ❼ ❸ ❾ ❽ ❺ ❷ ❸ ❶ )
merge( sort( ❼ ❸ ❾ ❽ ),sort( ❺ ❷ ❸ ❶ ) )
    merge( sort( ❼ ❸ ),sort( ❾ ❽ ) )
        merge( sort( ❼ ),sort( ❸ ) )
```

Merge Sort

Example ❼ ❸ ❾ ❽ ❺ ❷ ❸ ❶

```
sort( ❼ ❸ ❾ ❽ ❺ ❷ ❸ ❶ )

merge( sort( ❼ ❸ ❾ ❽ ),sort( ❺ ❷ ❸ ❶ ) )

    merge( sort( ❼ ❸ ),sort( ❾ ❽ ) )

        merge( sort( ❼ ),sort( ❸ ) )

        merge( ❼ , ❸ )
```

```
sort(item[0:len(item)/2])

sort(item[len(item)/2:len(item)])

merge(left, right)
```

Merge Sort

Example ❼ ❸ ❾ ❽ ❺ ❷ ❸ ❶

```
sort(item[0:len(item)/2])
sort(item[len(item)/2:len(item)])
merge(left, right)
```

```
sort( ❼ ❸ ❾ ❽ ❺ ❷ ❸ ❶ )

merge( sort( ❼ ❸ ❾ ❽ ),sort( ❺ ❷ ❸ ❶ ) )

    merge( sort( ❼ ❸ ),sort( ❾ ❽ ) )

        merge( sort( ❼ ),sort( ❸ ) )

        merge( ❼ , ❸ )

        ❸ ❼
```

Merge Sort

Example ❼ ❸ ❾ ❽ ❺ ❷ ❸ ❶

```
sort(item[0:len(item)/2])

sort(item[len(item)/2:len(item)])

merge(left, right)
```

sort(❼ ❸ ❾ ❽ ❺ ❷ ❸ ❶)

merge(sort(❼ ❸ ❾ ❽), sort(❺ ❷ ❸ ❶))

merge(sort(❼ ❸), sort(❾ ❽))

merge(sort(❼), sort(❸)), merge(sort(❾), sort(❽))

merge(❼ , ❸)

❸ ❼

Merge Sort

Example ❼ ❸ ❾ ❽ ❺ ❷ ❸ ❶

```
sort(item[0:len(item)/2])

sort(item[len(item)/2:len(item)])

merge(left, right)
```

sort(❼ ❸ ❾ ❽ ❺ ❷ ❸ ❶)

merge(sort(❼ ❸ ❾ ❽), sort(❺ ❷ ❸ ❶))

merge(sort(❼ ❸), sort(❾ ❽))

merge(sort(❼), sort(❸)), merge(sort(❾), sort(❽))

merge(❼, ❸), merge(❾, ❽)

❸ ❼

Merge Sort

Example 7 3 9 8 5 2 3 1

```
sort(item[0:len(item)/2])
sort(item[len(item)/2:len(item)])
merge(left, right)
```

```
sort( 7 3 9 8 5 2 3 1 )

merge( sort( 7 3 9 8 ), sort( 5 2 3 1 ) )

    merge( sort( 7 3 ), sort( 9 8 ) )

        merge( sort( 7 ), sort( 3 ) ), merge( sort( 9 ), sort( 8 ) )

        merge( 7 , 3 ), merge( 9 , 8 )

        3 7  8 9
```

Merge Sort

Example ❼ ❸ ❾ ❽ ❺ ❷ ❸ ❶

```
sort(item[0:len(item)/2])

sort(item[len(item)/2:len(item)])

merge(left, right)
```

```
sort( ❼ ❸ ❾ ❽ ❺ ❷ ❸ ❶ )

merge( sort( ❼ ❸ ❾ ❽ ),sort( ❺ ❷ ❸ ❶ ) )

merge( merge( sort( ❼ ❸ ),sort( ❾ ❽ ) )                                    )

    merge( merge( sort( ❼ ),sort( ❸ ) ),merge( sort( ❾ ),sort( ❽ ) ) )

    merge( merge( ❼ , ❸ ),merge( ❾ , ❽ ) )

    merge( ❸ ❼ , ❽ ❾ )

    ❸ ❼ ❽ ❾
```

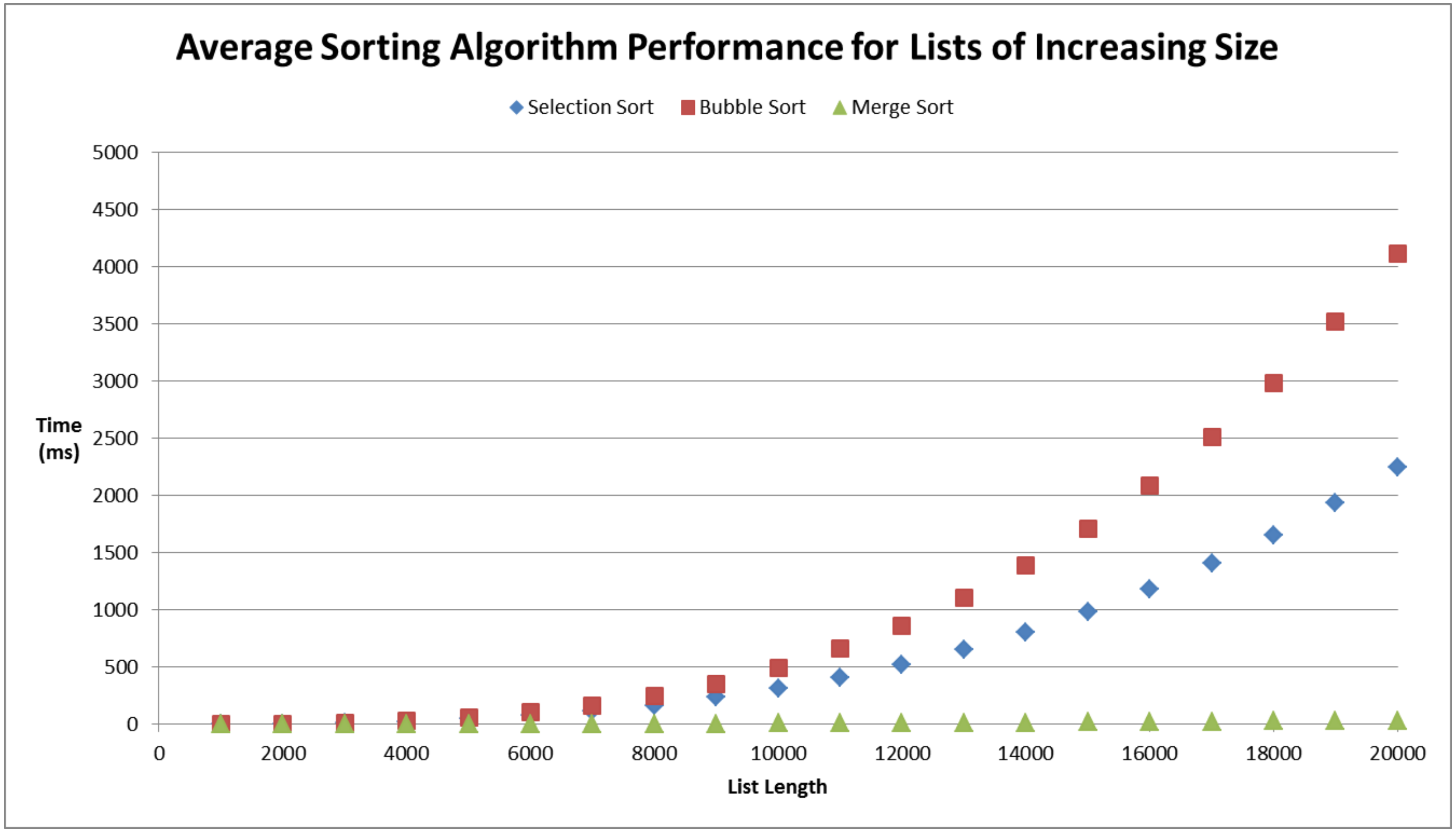
Merge Sort

Example 7 3 9 8 5 2 3 1

```
sort(item[0:len(item)/2])

sort(item[len(item)/2:len(item)])

merge(left, right)
```

```
sort( 7 3 9 8 5 2 3 1 )

merge( sort( 7 3 9 8 ), sort( 5 2 3 1 ) )

merge( merge( sort( 7 3 ), sort( 9 8 ) ), merge( sort( 5 2 ), sort( 3 1 ) ) )

merge( merge( merge( sort( 7 ), sort( 3 ) ), merge( sort( 9 ), sort( 8 ) ) ),
        merge( merge( sort( 5 ), sort( 2 ) ), merge( sort( 3 ), sort( 1 ) ) ) )

merge( merge( merge( 7 , 3 ), merge( 9 , 8 ) ), merge( merge( 5 , 2 ), merge( 3 , 1 ) ) )

merge( merge( 3 7 , 8 9 ), merge( 2 5 , 1 3 ) )

merge( 3 7 8 9 , 1 2 3 5 )

1 2 3 3 5 7 8 9
```

Measuring Real-Time Performance



Merge Sort Outclasses both Selection and Bubble Sort

(n.b., Not Just Faster; the Curve is Different!)

Analyzing Merge Sort Performance

for an array of Length n *(to simplify assume $n = 2^m$ for $m \in \mathbb{Z}^+$)*

```
sort(item):
    if (len(item) ≤ 1):
        return item
    else
        left  ← sort(item[...
        right ← sort(item[...
        return merge(left, right)
```

```
merge(left, right):
    j = 0, k = 0
    for i ∈ [0, len(left)+len(right)):
        if left[j] < right[k]:
            item[i] ← left[j]  ; j++
        else:
            item[i] ← right[k] ; k++
    return item
```

How Many Accesses to Split an Array of Length n in two? n

How Many Accesses to Merge Two Arrays of Length $^n/_2$? n

If Merge Sorting an array of Length n Required T Accesses

How Many Accesses to Merge Sort array of Length $^n/_2$?

Analyzing Selection Sort Performance

let `T(n)` be the Number of Accesses to Sort array of Length n

```
sort(item):
    if (len(item) ≤ 1):
        return item
    else
        left  ← sort(item[...
        right ← sort(item[...
        return merge(left, right)
```

```
merge(left, right):
    j = 0, k = 0
    for i ∈ [0, len(left)+len(right)):
        if left[j] < right[k]:
            item[i] ← left[j]  ; j++
        else:
            item[i] ← right[k] ; k++
    return item
```

$$T(n) = T(n/2) + T(n/2) + 2n$$
$$= 2(T(n/2)) + 2n$$

Analyzing Selection Sort Performance

let `T(n)` be the Number of Accesses to Sort array of Length n

```
sort(item):
    if (len(item) ≤ 1):
        return item
    else
        left  ← sort(item[...
        right ← sort(item[...
        return merge(left, right)
```

```
merge(left, right):
    j = 0, k = 0
    for i ∊ [0, len(left)+len(right)):
        if left[j] < right[k]:
            item[i] ← left[j]   ; j++
        else:
            item[i] ← right[k] ; k++
    return item
```

$$T(n) = T(n/2) + T(n/2) + 2n$$
$$= 2(T(n/2)) + 2n$$

$$T(n/2) = T(n/4) + T(n/4) + 2(n/2)$$
$$= 2(T(n/4)) + 2(n/2)$$

Analyzing Selection Sort Performance

let $T(n)$ be the Number of Accesses to Sort array of Length n

```
sort(item):
    if (len(item) ≤ 1):
        return item
    else
        left  ← sort(item[...
        right ← sort(item[...
        return merge(left, right)
```

```
merge(left, right):
    j = 0, k = 0
    for i ∈ [0, len(left)+len(right)):
        if left[j] < right[k]:
            item[i] ← left[j]  ; j++
        else:
            item[i] ← right[k] ; k++
    return item
```

$$\begin{aligned} T(n) &= T(n/2) + T(n/2) + 2n \\ &= 2(T(n/2)) + 2n \\ &= 2(2(T(n/4)) + (n/2)) + 2n \end{aligned}$$

$$\begin{aligned} T(n/2) &= T(n/4) + T(n/4) + 2(n/2) \\ &= 2(T(n/4)) + 2(n/2) \end{aligned}$$

Analyzing Selection Sort Performance

let `T(n)` be the Number of Accesses to Sort array of Length n

```
sort(item):
    if (len(item) ≤ 1):
        return item
    else
        left  ← sort(item[...
        right ← sort(item[...
        return merge(left, right)
```

```
merge(left, right):
    j = 0, k = 0
    for i ∈ [0, len(left)+len(right)):
        if left[j] < right[k]:
            item[i] ← left[j]  ; j++
        else:
            item[i] ← right[k] ; k++
    return item
```

$$
\begin{aligned}
T(n) &= T(n/2) + T(n/2) + 2n \\
&= 2(T(n/2)) + 2n \\
&= 2(2(T(n/4)) + 2(n/2)) + 2n \\
&= 2(2(T(n/4))) + 2n + 2n
\end{aligned}
$$

Analyzing Selection Sort Performance

let `T(n)` be the Number of Accesses to Sort array of Length n

```
sort(item):
    if (len(item) ≤ 1):
        return item
    else
        left  ← sort(item[...
        right ← sort(item[...
        return merge(left, right)
```

```
merge(left, right):
    j = 0, k = 0
    for i ∈ [0, len(left)+len(right)):
        if left[j] < right[k]:
            item[i] ← left[j]   ; j++
        else:
            item[i] ← right[k]  ; k++
    return item
```

$$
\begin{aligned}
T(n) &= T(^n/_2) + T(^n/_2) + 2n \\
&= 2(T(^n/_2)) + 2n \\
&= 2(2(T(^n/_4)) + 2(^n/_2)) + 2n \\
&= 2(2(T(^n/_4))) + 2n + 2n
\end{aligned}
$$

$$
\begin{aligned}
T(^n/_4) &= T(^n/_8) + T(^n/_8) + 2(^n/_4) \\
&= 2(T(^n/_8)) + 2(^n/_4)
\end{aligned}
$$

Analyzing Selection Sort Performance

let $T(n)$ be the Number of Accesses to Sort array of Length n

```
sort(item):
    if (len(item) ≤ 1):
        return item
    else
        left  ← sort(item[...
        right ← sort(item[...
        return merge(left, right)
```

```
merge(left, right):
    j = 0, k = 0
    for i ∈ [0, len(left)+len(right)):
        if left[j] < right[k]:
            item[i] ← left[j]  ; j++
        else:
            item[i] ← right[k] ; k++
    return item
```

$$\begin{aligned} T(n) &= T(n/2) + T(n/2) + 2n \\ &= 2(T(n/2)) + 2n \\ &= 2(2(T(n/4)) + 2(n/2)) + 2n \\ &= 2(2(T(n/4))) + 2n + 2n \\ &= 2(2(2(T(n/8))) + 2(n/4))) + 2n + 2n \end{aligned}$$

$$\begin{aligned} T(n/4) &= T(n/8) + T(n/8) + 2(n/4) \\ &= 2(T(n/8)) + 2(n/4) \end{aligned}$$

Analyzing Selection Sort Performance

let $T(n)$ be the Number of Accesses to Sort array of Length n

```
sort(item):
    if (len(item) ≤ 1):
        return item
    else
        left  ← sort(item[...
        right ← sort(item[...
        return merge(left, right)
```

```
merge(left, right):
    j = 0, k = 0
    for i ∊ [0, len(left)+len(right)):
        if left[j] < right[k]:
            item[i] ← left[j]   ; j++
        else:
            item[i] ← right[k] ; k++
    return item
```

$$
\begin{aligned}
T(n) &= T(^n/_2) + T(^n/_2) + 2n \\
&= 2(T(^n/_2)) + 2n \\
&= 2(2(T(^n/_4)) + 2(^n/_2)) + 2n \\
&= 2(2(T(^n/_4))) + 2n + 2n \\
&= 2(2(2(T(^n/_8))) + 2(^n/_4))) + 2n + 2n \\
&= 2(2(2(T(^n/_8)))) + 2n + 2n + 2n
\end{aligned}
$$

Analyzing Selection Sort Performance

let `T(n)` be the Number of Accesses to Sort array of Length n

```
sort(item):
    if (len(item) ≤ 1):
        return item
    else
        left  ← sort(item[...
        right ← sort(item[...
        return merge(left, right)
```

```
merge(left, right):
    j = 0, k = 0
    for i ∈ [0, len(left)+len(right)):
        if left[j] < right[k]:
            item[i] ← left[j]  ; j++
        else:
            item[i] ← right[k] ; k++
    return item
```

$$\begin{aligned} T(n) &= T(^n/_2) + T(^n/_2) + 2n \\ &= 2(T(^n/_2)) + 2n \\ &= 2(2(T(^n/_4)) + 2(^n/_2)) + 2n \\ &= 2(2(T(^n/_4))) + 2n + 2n \\ &= 2(2(2(T(^n/_8))) + 2(^n/_4))) + 2n + 2n \\ &= 2(2(2(T(^n/_8)))) + 2n + 2n + 2n \end{aligned}$$

...after k steps...

$$= 2^k(T(^n/_{2^k})) + 2nk$$

Analyzing Selection Sort Performance

let `T(n)` be the Number of Accesses to Sort array of Length n

```
sort(item):
    if (len(item) ≤ 1):
        return item
    else
        left  ← sort(item[...
        right ← sort(item[...
        return merge(left, right)
```

```
merge(left, right):
    j = 0, k = 0
    for i ∈ [0, len(left)+len(right)):
        if left[j] < right[k]:
            item[i] ← left[j]   ; j++
        else:
            item[i] ← right[k] ; k++
    return item
```

$$T(n) = 2^k(T(n/2^k)) + 2nk$$

...Recall the Assumption that $n = 2^m$ for Integer m...

...How Many Steps (k) Until we Evaluate `T(1)`*?* *m*

$$= 2^m(T(n/2^m)) + 2nm$$

$n = 2^m$

$$= n(T(1)) + 2nm$$

$m = \log_2 n$

$$= n(T(1)) + 2n(\log_2(n))$$

...and this is actually 0!

Comparing Theoretical Performances

Complexity Analyses for the Sorting Algorithms can be Compared

Selection Sort	=	$n^2 + 2n - 3$
Bubble Sort	=	${}^5/_2(n^2 - n)$
Merge Sort	=	$2(n)(\log_2 n)$

at Very Large Values of n (i.e., very large lists)

Higher-Order Terms Dominate (Grow Faster) than Lower Order

(at Very Large Values of n: $n^2 \gg n \log n \gg n \gg \log n$)

Remove the Lower Order Terms

Comparing Theoretical Performances

Complexity Analyses for the Sorting Algorithms can be Compared

Selection Sort	=	$n^2 + 2n - 3$
Bubble Sort	=	$^5/_2(n^2 - n)$
Merge Sort	=	$2(n)(\log_2 n)$

Constants and Constant Co-efficients can also be Removed

Why?

Comparing Theoretical Performances

Complexity Analyses for the Sorting Algorithms can be Compared

Selection Sort	=	$n^2 + 2n - 3$
Bubble Sort	=	$^5/_2(n^2 - n)$
Merge Sort	=	$2(n)(\log_2 n)$

Constants and Constant Co-efficients can also be Removed

Why?

$3n^2 > 2n^2$ but Difference is Relatively Insignificant

(more Concerned with Changing Order than constants)

Complexity Analysis and Big-Oh Notation

Algorithms are typically Compared using Big-Oh Notation

$f(n) = O(g(n))$ "*Function f(n) is Big-Oh g(n)*"

$\updownarrow$ $\updownarrow$

there are Two Constants c and k such that

$0 \leq f(n) \leq cg(n)$ **for all** $n \geq k$

Complexity Analysis and Big-Oh Notation

Algorithms are typically Compared using Big-Oh Notation

`f(n) = O(g(n))` ↕ "*Function f(n) is Big-Oh g(n)*" ↕

there are Two Constants c and k such that

$0 \le f(n) \le cg(n)$ **for all** $n \ge k$

"***Function*** `f` ***Does Not Grow Any Faster than Function*** `g`"

to Convert a Formula into it's Big-Oh Notation

Remove all Constants and Constant Co-efficients

and Remove Everything Except the Highest-Order Term

Search Algorithms Revisited

given an Telephone Directory (i.e., Phonebook)

there are Several Approaches for Searching it (since it is Sorted)

Naïve Approach

starting from the first page of the phonebook, check for the name "Collier" on each page and, if you don't find it, move on to the next page and continue the search until you check the whole book

Alternative Approach

open the phonebook to the center and compare the name on that page with "Collier" and, if "Collier" isn't there, at least you know which half will contain it, so you start again with only half the book

Describe Each Approach in Pseudocode

Express the Time Complexity of each (using Big-Oh)

Linear Search Pseudocode

```
linearsearch(list, key, low, high):
    for i ∈ [0, len(list)):
        if (list[i] == key):
            return i
    return "key not found"
```

Binary Search Pseudocode

```
binarysearch(list, key, low, high):
    if (high < low):
        return "key not found"
    else
        center  ← (low + high) / 2
        if (list[center] > key):
            return binarysearch(list, key, low, center-1)
        else if (list[center] < key):
            return binarysearch(list, key, center+1, high)
        else
            return center
```

Best / Worst / Average Case Scenarios

given an Telephone Directory in which you must Search for a Key

if you Use the Linear Search Algorithm:

What is the Best-Case Scenario?

(i.e., What would be the Easiest to Find?)

If the Directory has n Pages,

What is the Worst-Case Scenario?

Best / Worst / Average Case Scenarios

given an Telephone Directory in which you must Search for a Key

if you Use the Linear Search Algorithm:

What is the Best-Case Scenario?

(i.e., What would be the Easiest to Find?)

on the First Page

1 Page Searched

If the Directory has n Pages,

What is the Worst-Case Scenario?

Best / Worst / Average Case Scenarios

given an Telephone Directory in which you must Search for a Key

if you Use the Linear Search Algorithm:

What is the Best-Case Scenario?

(i.e., What would be the Easiest to Find?)

on the First Page

1 Page Searched

If the Directory has n Pages,

What is the Worst-Case Scenario?

on the Last Page

n Pages Searched

Best / Worst / Average Case Scenarios

given an Telephone Directory in which you must Search for a Key

if you Use the Linear Search Algorithm:

What is the Best-Case Scenario?

(i.e., What would be the Easiest to Find?)

on the First Page

1 Page Searched

If the Directory has n Pages,

What is the Worst-Case Scenario?

on the Last Page

n Pages Searched

What is the Average-Case Scenario?

Best / Worst / Average Case Scenarios

given an Telephone Directory in which you must Search for a Key

if you Use the Linear Search Algorithm:

What is the Best-Case Scenario?

(i.e., What would be the Easiest to Find?)

on the First Page

1 Page Searched

If the Directory has n Pages,

What is the Worst-Case Scenario?

on the Last Page

n Pages Searched

What is the Average-Case Scenario?

on Average you must Search Half (i.e., n/2)

Best / Worst / Average Case Scenarios

given an Telephone Directory in which you must Search for a Key

if you Use the Binary Search Algorithm:

What is the Best-Case Scenario?

(i.e., What would be the Easiest to Find?)

If the Directory has n Pages,

What is the Worst-Case Scenario?

Best / Worst / Average Case Scenarios

given an Telephone Directory in which you must Search for a Key

if you Use the Binary Search Algorithm:

What is the Best-Case Scenario?

(i.e., What would be the Easiest to Find?)

on the First Page

1 Page Searched

If the Directory has n Pages,

What is the Worst-Case Scenario?

Best / Worst / Average Case Scenarios

given an Telephone Directory in which you must Search for a Key

if you Use the Binary Search Algorithm:

What is the Best-Case Scenario?

(i.e., What would be the Easiest to Find?)

on the First Page

1 Page Searched

If the Directory has n Pages,

What is the Worst-Case Scenario?

?

How can we Determine the Worst-Case

Time Complexity of Binary Search?

Analyzing Binary Search Performance

let `T(n)` be the Number of Accesses to Search Array of Length n

```
binarysearch(list, key, low, high):
     ...
     center  ← (low + high) / 2
     if (list[center] > key):
          return binarysearch(list, key, low, center-1)
     else if (list[center] < key):
          return binarysearch(list, key, center+1, high)
     else
          return center
```

Access Center and then Search an Array of Length $^n/_2$

$$T(n) = 1 + T(^n/_2)$$
$$T(n) = 1 + 1 + T(^n/_4) = 2 + T(^n/_4)$$
$$T(n) = 1 + 1 + 1 + \ldots + T(^n/_{2^k})) = k + T(^n/_{2^k})$$

...Assume that $n = 2^m$ for Integer m...

...How Many Steps (k) Until we Evaluate `T(1)`*?* *m*

$$T(n) = m + T(^n/_{2^m})$$
$$T(n) = \log_2(n) + T(1)$$

Comparing Theoretical Performances

Complexity Analyses for the Searching Algorithms

(assuming the Worst-Case Scenario)

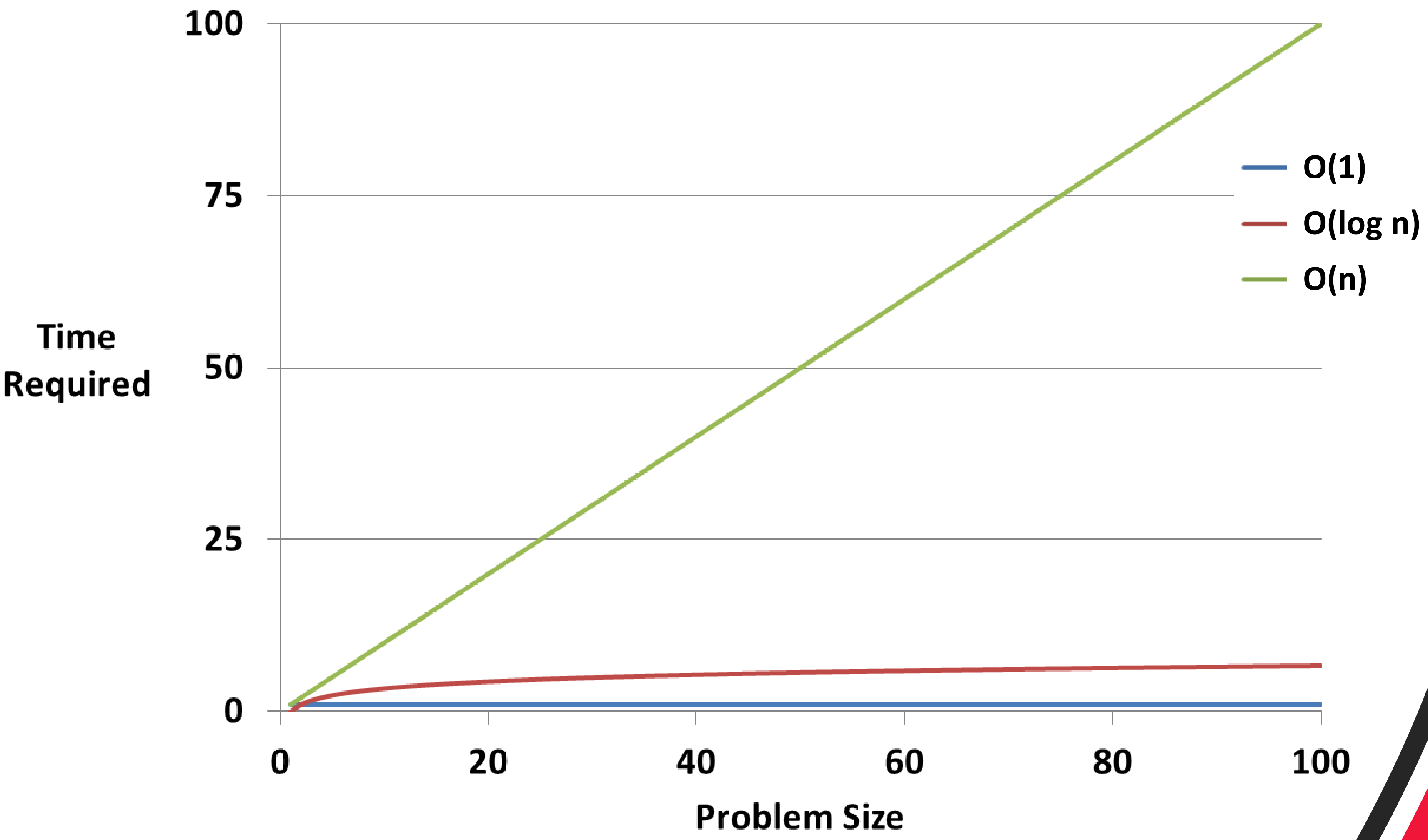
Linear Search	=	`n`
Binary Search	=	$\log_2 n$

If the Array to be Searched Doubles in Size

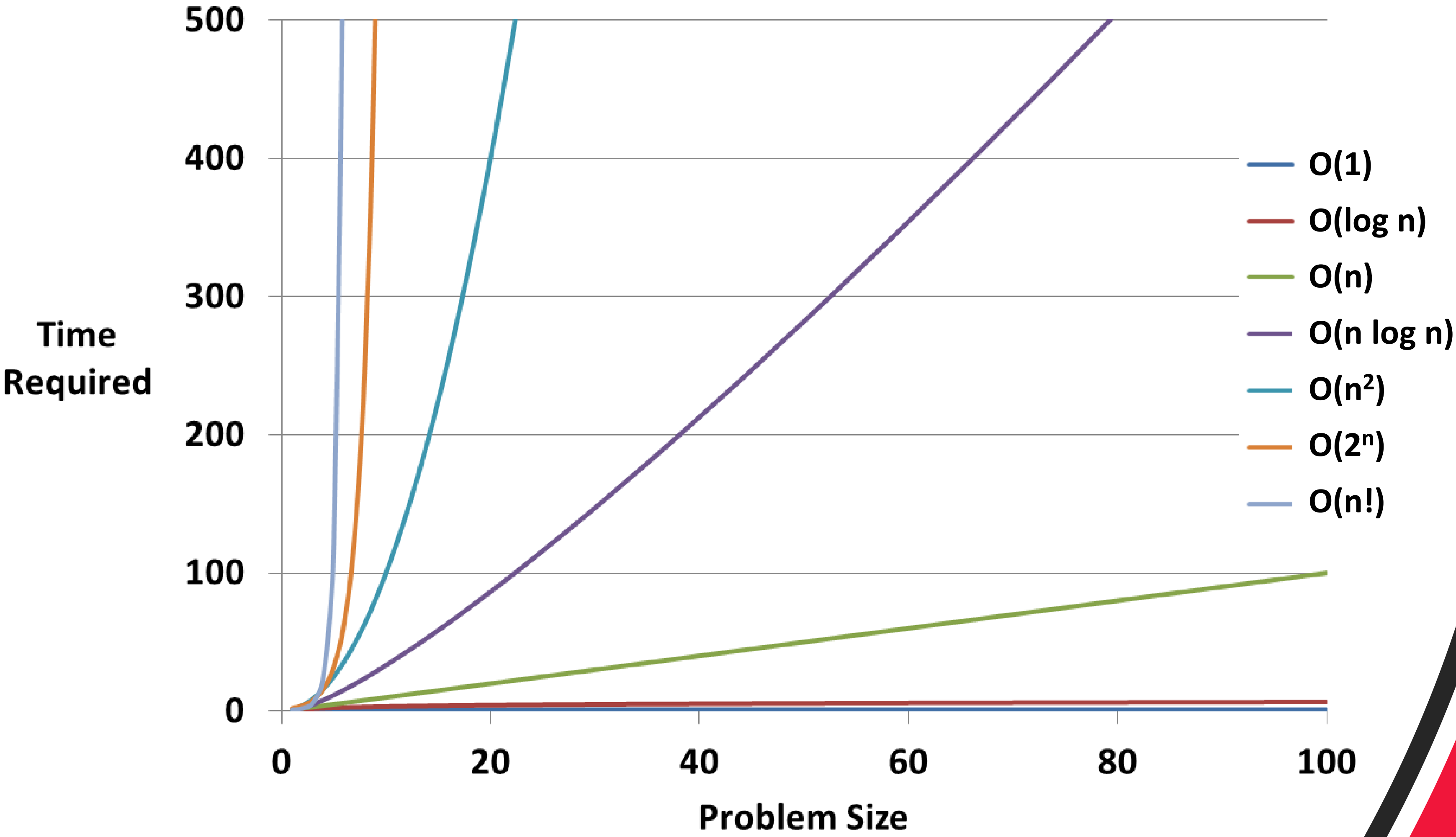
How Much Worse is Linear Search? Binary Search?

n	vs.	$2n$
$\log_2 (n)$	vs.	$\log_2 (2n)$
		$1 + \log_2 (n)$

Comparing Theoretical Performances



Comparing Theoretical Performances



Time Complexity Terms

Time Complexity	Term	Reaction
O(1)	"Constant"	
O(log n)	"Logarithmic"	
O(n)	"Linear"	
O(n log n)	"Quasilinear"	
$O(n^2)$	"Polynomial"	
$O(2^n)$	"Exponential"	
O(n!)	"Factorial"