Algorithm Problem Solving (APS): Sorting

Niema Moshiri
UC San Diego SPIS 2019
Introduction to Sorting

- Many algorithms require the input data to be sorted
Introduction to Sorting

- Many algorithms require the input data to be sorted
- **Computational Problem:** Given \( n \) “comparable” items, order them such that the \( i \)-th element is less than or equal to the \((i+1)\)-th element
Introduction to Sorting

- Many algorithms require the input data to be sorted

- **Computational Problem:** Given $n$ “comparable” items, order them such that the $i$-th element is less than or equal to the $(i+1)$-th element
  - This is for sorting in *ascending* order
Introduction to Sorting

- Many algorithms require the input data to be sorted

- **Computational Problem:** Given \( n \) “comparable” items, order them such that the \( i \)-th element is less than or equal to the \((i+1)\)-th element
  - This is for sorting in *ascending* order
  - Just change “less than” to “greater than” for *descending* order
Introduction to Sorting

- How do we sort $n$ items?
Introduction to Sorting

- How do we sort $n$ items?
Introduction to Sorting

- How do we sort $n$ items?
- No, not just clicking a button...
Introduction to Sorting

- How do we sort $n$ items?

- No, not just clicking a button...

```
niema@DESKTOP-G7N2912:~$ python3
Python 3.6.8 (default, Jan 14 2019, 11:02:34)
[GCC 8.0.1 20180414 (experimental) [trunk revision 259383]] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> numbers = [93,68,14,0,22,20,20,46,59,35,36,93,68,79,4,55,15]
>>> numbers.sort()
>>> print(numbers)
[0, 4, 14, 15, 20, 20, 22, 35, 36, 46, 55, 59, 68, 68, 79, 93, 93]
```
Introduction to Sorting

- How do we sort $n$ items?
- No, not just clicking a button...
- No, what’s *actually* happening behind the scenes?
Introduction to Sorting

● How do we sort $n$ items?

● No, not just clicking a button...

● No, what’s *actually* happening behind the scenes?

● Let’s discuss some sorting algorithms!
Introduction to Sorting

- How do we sort $n$ items?
- No, not just clicking a button...
- No, what’s actually happening behind the scenes?
- Let’s discuss some sorting algorithms!
- But first, let’s discuss time complexity using Big-O notation
Time Complexity
Describing an Algorithm

- Algorithms can be complicated, but what’s important to the user?
Describing an Algorithm

- Algorithms can be complicated, but what’s important to the user?
  - **Correctness:** Will it give me the right answer?
Describing an Algorithm

- Algorithms can be complicated, but what’s important to the user?
  - **Correctness:** Will it give me the right answer?
  - **Runtime:** How long will it take to run?
Describing an Algorithm

- Algorithms can be complicated, but what’s important to the user?
  - **Correctness:** Will it give me the right answer?
  - **Runtime:** How long will it take to run?

- “Runtime” can be measured using the following:
Describing an Algorithm

- Algorithms can be complicated, but what’s important to the user?
  - **Correctness:** Will it give me the right answer?
  - **Runtime:** How long will it take to run?

- “Runtime” can be measured using the following:
  - Human Time (e.g. seconds)
Describing an Algorithm

- Algorithms can be complicated, but what’s important to the user?
  - **Correctness:** Will it give me the right answer?
  - **Runtime:** How long will it take to run?

- “Runtime” can be measured using the following:
  - Human Time (e.g. seconds)
  - Computer Time (e.g. clock cycles)
Runtime is Implementation-Dependent

- An “algorithm” is a mathematical entity
Runtime is Implementation-Dependent

- An “algorithm” is a mathematical entity
  - A “program” is just an implementation of an algorithm
Runtime is Implementation-Dependent

- An “algorithm” is a mathematical entity
  - A “program” is just an implementation of an algorithm
- “Runtime” measures a program, not an algorithm
Runtime is Implementation-Dependent

- An “algorithm” is a mathematical entity
  - A “program” is just an *implementation* of an algorithm
- “Runtime” measures a *program*, not an *algorithm*
  - The same *program* run on newer hardware can run faster
Runtime is Implementation-Dependent

- An “algorithm” is a mathematical entity
  - A “program” is just an *implementation* of an algorithm

- “Runtime” measures a *program*, not an *algorithm*
  - The same *program* run on newer hardware can run faster
  - Thus, “runtime” may not be the best way to describe an algorithm
Runtime is Implementation-Dependent

● An “algorithm” is a mathematical entity
  ○ A “program” is just an *implementation* of an algorithm

● “Runtime” measures a *program*, not an *algorithm*
  ○ The same *program* run on newer hardware can run faster
  ○ Thus, “runtime” may not be the best way to describe an algorithm
  ○ Can we describe an algorithm independently of implementation?
Time Complexity

- We can use “time complexity” to directly describe an algorithm
Time Complexity

- We can use “time complexity” to directly describe an algorithm.
- Time complexity describes how an algorithm scales.
Time Complexity

- We can use “time complexity” to directly describe an algorithm.
- Time complexity describes how an algorithm scales.
  - It describes the number of operations performed by an algorithm.
Time Complexity

- We can use “time complexity” to directly describe an algorithm.
- Time complexity describes how an algorithm *scales*.
  - It describes the number of operations performed by an algorithm.
  - But with what input data?
The Best, the Worst, and the Average

- To describe an algorithm, we need to think of the input “case”
The Best, the Worst, and the Average

● To describe an algorithm, we need to think of the input “case”
  ○ The best case is the best possible scenario for the algorithm
The Best, the Worst, and the Average

- To describe an algorithm, we need to think of the input “case”
  - The **best case** is the best possible scenario for the algorithm
  - The **worst case** is the worst possible scenario for the algorithm
The Best, the Worst, and the Average

- To describe an algorithm, we need to think of the input “case”
  - The **best case** is the best possible scenario for the algorithm
  - The **worst case** is the worst possible scenario for the algorithm
  - The **average case** is the theoretical expectation
The Best, the Worst, and the Average

● To describe an algorithm, we need to think of the input “case”
  ○ The **best case** is the best possible scenario for the algorithm
  ○ The **worst case** is the worst possible scenario for the algorithm
  ○ The **average case** is the theoretical expectation

● People typically mainly care about the worst case
The Best, the Worst, and the Average

- To describe an algorithm, we need to think of the input “case”
  - The **best case** is the best possible scenario for the algorithm
  - The **worst case** is the worst possible scenario for the algorithm
  - The **average case** is the theoretical expectation

- People typically mainly care about the worst case
  - “Your package will arrive in around 1 to 100 days”
Big-O, Big-Ω, and Big-Θ

- We first need to pick a case (worst, best, or average)
Big-O, Big-Ω, and Big-Θ

- We first need to pick a case (worst, best, or average)
  - What do we do next to describe how our algorithm scales?
Big-O, Big-Ω, and Big-Θ

- We first need to pick a case (worst, best, or average)
  - What do we do next to describe how our algorithm scales?
  - We can describe the number of operations our algorithm performs
Big-O, Big-Ω, and Big-Θ

- We first need to pick a case (worst, best, or average)
  - What do we do next to describe how our algorithm scales?
  - We can describe the number of operations our algorithm performs

- **Big-O**: A function that is an *upper* bound on the number of operations
Big-O, Big-Ω, and Big-Θ

- We first need to pick a case (worst, best, or average)
  - What do we do next to describe how our algorithm scales?
  - We can describe the number of operations our algorithm performs

- **Big-O**: A function that is an *upper* bound on the number of operations

- **Big-Ω**: A function that is a *lower* bound on the number of operations
Big-O, Big-Ω, and Big-Θ

- We first need to pick a case (worst, best, or average)
  - What do we do next to describe how our algorithm scales?
  - We can describe the number of operations our algorithm performs

- **Big-O**: A function that is an *upper* bound on the number of operations

- **Big-Ω**: A function that is a *lower* bound on the number of operations

- **Big-Θ**: A function that is both an upper *and* lower bound
Big-O, Big-Ω, and Big-Θ

• We first need to pick a case (worst, best, or average)
  ○ What do we do next to describe how our algorithm scales?
  ○ We can describe the number of operations our algorithm performs

• **Big-O**: A function that is an *upper* bound on the number of operations

• **Big-Ω**: A function that is a *lower* bound on the number of operations

• **Big-Θ**: A function that is both an upper *and* lower bound
Example: Big-O, Big-Ω, and Big-Θ

- Number of Operations = \( f(n) = 2n^2 + 3n + 1 \)
Example: Big-O, Big-Ω, and Big-Θ

- Number of Operations = \( f(n) = 2n^2 + 3n + 1 \)
Example: Big-O, Big-Ω, and Big-Θ

- Number of Operations = \( f(n) = 2n^2 + 3n + 1 \)

\( f(n) \) is \( \Theta(n^2) \)

\( g(n) = 3n^2 \)

\( f(n) = 2n^2 + 3n + 1 \)
Example: Big-O, Big-Ω, and Big-Θ

- Number of Operations = $f(n) = 2n^2 + 3n + 1$
Example: Big-O, Big-Ω, and Big-Θ

- Number of Operations = \( f(n) = 2n^2 + 3n + 1 \)
Example: Big-O, Big-Ω, and Big-Θ

- Number of Operations = \( f(n) = 2n^2 + 3n + 1 \)

\( f(n) \) is both \( O(n^2) \) and \( \Omega(n^2) \) therefore...

\( f(n) \) is \( \Theta(n^2) \)
Finding the Big-O Time Complexity

- Imagine we have a function $f(n)$ denoting the number of operations
Finding the Big-O Time Complexity

- Imagine we have a function $f(n)$ denoting the number of operations
  - First, drop all lower terms of $n$ in the addition
Finding the Big-O Time Complexity

- Imagine we have a function $f(n)$ denoting the number of operations
  - First, drop all lower terms of $n$ in the addition
  - Second, drop all constant coefficients
Finding the Big-O Time Complexity

- Imagine we have a function $f(n)$ denoting the number of operations
  - First, drop all lower terms of $n$ in the addition
  - Second, drop all constant coefficients
- Example: $f(n) = 5n \log n + 2n + 27$
Finding the Big-O Time Complexity

- Imagine we have a function $f(n)$ denoting the number of operations
  - First, drop all lower terms of $n$ in the addition
  - Second, drop all constant coefficients
- Example: $f(n) = 5n \log n + 2n + 27$
  - $5n \log n + 2n + 27 \rightarrow 5n \log n$
Finding the Big-O Time Complexity

- Imagine we have a function $f(n)$ denoting the number of operations
  - First, drop all lower terms of $n$ in the addition
  - Second, drop all constant coefficients

- Example: $f(n) = 5n \log n + 2n + 27$
  - $5n \log n + 2n + 27 \rightarrow 5n \log n$
  - $5n \log n \rightarrow n \log n \rightarrow O(n \log n)$
Selection Sort
Selection Sort

Algorithm `selection_sort(X)`:

1. **output** ← empty list
2. Repeat \(|X|\) times:
   1. `y` ← smallest item in `X`
   2. Remove `y` from `X`
   3. Add `y` to `output`
3. Return `output`
Selection Sort

<table>
<thead>
<tr>
<th>7</th>
<th>25</th>
<th>0</th>
<th>42</th>
<th>-9</th>
</tr>
</thead>
</table>


Selection Sort

<table>
<thead>
<tr>
<th>7</th>
<th>25</th>
<th>0</th>
<th>42</th>
<th>-9</th>
</tr>
</thead>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>
Selection Sort

| 7 | 25 | 0 | 42 | -9 |

-9
Selection Sort

7  25  0  42  -9

-9
### Selection Sort

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>25</td>
<td>0</td>
<td>42</td>
<td>-9</td>
<td>-9</td>
</tr>
</tbody>
</table>
Selection Sort

<table>
<thead>
<tr>
<th>7</th>
<th>25</th>
<th>0</th>
<th>42</th>
<th>-9</th>
</tr>
</thead>
<tbody>
<tr>
<td>-9</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>25</td>
<td>0</td>
<td>42</td>
<td>-9</td>
</tr>
<tr>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

-9  | 0  |    |    |    |
Selection Sort

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>25</td>
<td>0</td>
<td>42</td>
<td>-9</td>
<td>-9</td>
</tr>
<tr>
<td>-9</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Selection Sort

<table>
<thead>
<tr>
<th>7</th>
<th>25</th>
<th>0</th>
<th>42</th>
<th>-9</th>
</tr>
</thead>
<tbody>
<tr>
<td>-9</td>
<td>0</td>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Selection Sort

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>25</td>
<td>0</td>
<td>42</td>
<td>-9</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-9</td>
</tr>
<tr>
<td>-9</td>
<td>0</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Selection Sort

<table>
<thead>
<tr>
<th>7</th>
<th>25</th>
<th>0</th>
<th>42</th>
<th>-9</th>
</tr>
</thead>
<tbody>
<tr>
<td>-9</td>
<td>0</td>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Selection Sort

<table>
<thead>
<tr>
<th></th>
<th>25</th>
<th>0</th>
<th>42</th>
<th>-9</th>
</tr>
</thead>
<tbody>
<tr>
<td>-9</td>
<td>0</td>
<td>7</td>
<td>25</td>
<td></td>
</tr>
</tbody>
</table>
Selection Sort

7  25  0  42  -9

-9  0  7  25
Selection Sort

7  25  0  42  -9

-9  0  7  25
### Selection Sort

<table>
<thead>
<tr>
<th>7</th>
<th>25</th>
<th>0</th>
<th>42</th>
<th>-9</th>
</tr>
</thead>
<tbody>
<tr>
<td>-9</td>
<td>0</td>
<td>7</td>
<td>25</td>
<td>42</td>
</tr>
</tbody>
</table>
Selection Sort

<table>
<thead>
<tr>
<th></th>
<th>7</th>
<th>25</th>
<th>0</th>
<th>42</th>
<th>-9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-9</td>
<td>0</td>
<td>7</td>
<td>25</td>
<td>42</td>
</tr>
</tbody>
</table>
## Selection Sort

<table>
<thead>
<tr>
<th>7</th>
<th>25</th>
<th>0</th>
<th>42</th>
<th>-9</th>
</tr>
</thead>
</table>

Can we do it in-place?

| -9 | 0 | 7 | 25 | 42 |
Selection Sort (In-Place)

| 7  | 25 | 0  | 42 | -9 |
Selection Sort (In-Place)

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>25</td>
<td>0</td>
<td>42</td>
<td>-9</td>
<td></td>
</tr>
</tbody>
</table>
Selection Sort (In-Place)

| 7 | 25 | 0 | 42 | -9 |
Selection Sort (In-Place)
Selection Sort (In-Place)

<table>
<thead>
<tr>
<th></th>
<th>-9</th>
<th>25</th>
<th>0</th>
<th>42</th>
<th>7</th>
</tr>
</thead>
</table>
Selection Sort (In-Place)

| -9 | 25 | 0 | 42 | 7 |
## Selection Sort (In-Place)

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-9</td>
<td>25</td>
<td>0</td>
<td>42</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>
Selection Sort (In-Place)

<table>
<thead>
<tr>
<th></th>
<th>25</th>
<th>0</th>
<th>42</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>-9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Selection Sort (In-Place)

-9  0  25  42  7
Selection Sort (In-Place)

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-9</td>
<td>0</td>
<td>25</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>-9</td>
<td>0</td>
<td>25</td>
<td>42</td>
</tr>
</tbody>
</table>

**Selection Sort (In-Place)**
Selection Sort (In-Place)

-9  0  25  42  7
Selection Sort (In-Place)

-9  0  7  42  25
Selection Sort (In-Place)
## Selection Sort (In-Place)

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-9</td>
<td>0</td>
<td>7</td>
<td>42</td>
<td>25</td>
</tr>
</tbody>
</table>
Selection Sort (In-Place)
Selection Sort (In-Place)

-9  0  7  25  42
Selection Sort (In-Place)

-9  0  7  25  42
Selection Sort (In-Place)

-9  0  7  25  42
Selection Sort (In-Place)

-9  0  7  25  42
Selection Sort (In-Place)

-9  0  7  25  42
Selection Sort (In-Place)

What type of algorithm is this?
Selection Sort: Worst-Case Time Complexity
Selection Sort: Worst-Case Time Complexity

- For each of our $n$ iterations:
Selection Sort: Worst-Case Time Complexity

- For each of our \( n \) iterations:
  - Find the smallest remaining item
Selection Sort: Worst-Case Time Complexity

- For each of our $n$ iterations:
  - Find the smallest remaining item
    - In the $i$-th iteration (0-based counting), we check $n - i$ items
Selection Sort: Worst-Case Time Complexity

- For each of our $n$ iterations:
  - Find the smallest remaining item
    - In the $i$-th iteration (0-based counting), we check $n - i$ items
- Total number of operations $= n + (n-1) + (n-2) + \ldots + 3 + 2 + 1$
Selection Sort: Worst-Case Time Complexity

- For each of our $n$ iterations:

  - Find the smallest remaining item

    - In the $i$-th iteration (0-based counting), we check $n - i$ items

- Total number of operations = $n + (n-1) + (n-2) + ... + 3 + 2 + 1$

  - This is the sum of the integers from 1 to $n$, which is $n(n+1)/2$
Selection Sort: Worst-Case Time Complexity

- For each of our $n$ iterations:
  - Find the smallest remaining item
    - In the $i$-th iteration (0-based counting), we check $n - i$ items
  - Total number of operations = $n + (n-1) + (n-2) + \ldots + 3 + 2 + 1$
    - This is the sum of the integers from 1 to $n$, which is $n(n+1)/2$
    - $n(n+1)/2 = n^2 + n \rightarrow O(n^2)$
Selection Sort: Worst-Case Time Complexity

- For each of our $n$ iterations:
  - Find the smallest remaining item

  - Total number of operations $= n + (n-1) + (n-2) + \ldots + 3 + 2 + 1$
    - This is the sum of the integers from 1 to $n$, which is $n(n+1)/2$
    - $n(n+1)/2 = n^2 + n \rightarrow O(n^2)$
Merge Sort
Merge Sort

Algorithm merge_sort(X):

If |X| only has 1 item:

    Return |X|

left ← merge_sort(left half of X)

right ← merge_sort(right half of X)

Return the result of merging left and right
Merge Sort

| -9 | 0 | 7 | 25 | 42 | 5 | -2 | 12 |
Merge Sort

-9 0 7 25

42 5 -2 12
Merge Sort

-9 0 7 25

42 5 -2 12

-9 0 7 25

42 5 -2 12
Merge Sort

-9 0 7 25

42 5 -2 12

-9 0 7 25

42 5 -2 12
Merge Sort

-9 0 7 25

42 5 -2 12

-9 0 7 25

42 5 -2 12
Merge Sort
Merge Sort

-9 0 7 25
-9 0
-9 0

42 5 -2 12
42 5
42 5
5 42
-2 12
Merge Sort
Merge Sort
Merge Sort

What type of algorithm is this?
Merging Two Sorted Lists

Algorithm `merge(X,Y)`:  

```plaintext
output ← empty list;  i,j ← 0
While i < |X| and j < |Y|:
    If X[i] < Y[j]:
        Add X[i] to output and increment i
    Else:
        Add Y[j] to output and increment j
Add remaining items to output
Return output
```
Merging Two Sorted Lists

-9 0 7 25

-2 5 12
Merging Two Sorted Lists

-9  0  7  25
-2  5  12
Merging Two Sorted Lists

-9 0 7 25

-2 5 12

-9
Merging Two Sorted Lists

-9 0 7 25

-2 5 12

-9
Merging Two Sorted Lists

-9 0 7 25

-2 5 12

-9 -2
Merging Two Sorted Lists

-9 0 7 25

-2 5 12

-9 -2
Merging Two Sorted Lists

-9 0 7 25

-2 5 12

-9 -2 0
Merging Two Sorted Lists
Merging Two Sorted Lists

-9 0 7 25

-2 5 12

-9 -2 0 5
Merging Two Sorted Lists

-9 0 7 25

-2 5 12

-9 -2 0 5
Merging Two Sorted Lists

-9 0 7 25

-2 5 12

-9 -2 0 5 7
Merging Two Sorted Lists

-9 0 7 25

-2 5 12

-9 -2 0 5 7
Merging Two Sorted Lists

-9 0 7 25

-2 5 12

-9 -2 0 5 7 12
Merging Two Sorted Lists

-9 0 7 25

-2 5 12

-9 -2 0 5 7 12
Merging Two Sorted Lists

-9 0 7 25

-2 5 12

-9 -2 0 5 7 12 25
Merging Two Sorted Lists

-9 0 7 25

-9 -2 0 5 7 12 25

-2 5 12
Merge Sort: Worst-Case Time Complexity
Merge Sort: Worst-Case Time Complexity

- For each of our $\log_2 n$ levels of merging:
Merge Sort: Worst-Case Time Complexity

- For each of our $\log_2 n$ levels of merging:
  - Merge pairs of sorted lists ($n$ items total)
Merge Sort: Worst-Case Time Complexity

- For each of our $\log_2 n$ levels of merging:
  - Merge pairs of sorted lists ($n$ items total)
    - In each level of merging, each item is checked only once
Merge Sort: Worst-Case Time Complexity

- For each of our $\log_2 n$ levels of merging:
  - Merge pairs of sorted lists ($n$ items total)
    - In each level of merging, each item is checked only once
  - Total number of operations = $n + n + ... + n$ (once per row of merging)
Merge Sort: Worst-Case Time Complexity

- For each of our $\log_2 n$ levels of merging:
  - Merge pairs of sorted lists ($n$ items total)
    - In each level of merging, each item is checked only once
  - Total number of operations = $n + n + \ldots + n$ (once per row of merging)
    - We have $\log_2 n$ rows of merging, so $n \log_2 n$ total, $\times 2$ for dividing
Merge Sort: Worst-Case Time Complexity

- For each of our $\log_2 n$ levels of merging:
  - Merge pairs of sorted lists ($n$ items total)
    - In each level of merging, each item is checked only once
  - Total number of operations = $n + n + \ldots + n$ (once per row of merging)
    - We have $\log_2 n$ rows of merging, so $n \log_2 n$ total, $\times 2$ for dividing
    - $2n \log_2 n \rightarrow O(n \log n)$
Merge Sort: Worst-Case Time Complexity

- For each of our $\log_2 n$ levels of merging:
  - Merge pairs of sorted lists ($n$ items total)

Can we do better?

- Total number of operations = $n + n + ... + n$ (once per row of merging)
  - We have $\log_2 n$ rows of merging, so $n \log_2 n$ total, $\times 2$ for dividing
  - $2n \log_2 n \rightarrow O(n \log n)$
Merge Sort: Worst-Case Time Complexity

- For each of our \( \log_2 n \) levels of merging:
  - Merge pairs of sorted lists (\( n \) items total)
    - In each level of merging, each item is checked only once
  - Total number of operations = \( n + n + \ldots + n \) (once per row of merging)
    - We have \( \log_2 n \) rows of merging, so \( n \log_2 n \) total, \( \times 2 \) for dividing
    - \( 2n \log_2 n \rightarrow O(n \log n) \)

Can we do better?

Probably not!