Sorting

Guttag Chapter 12

Goals

Understand why and when is sorting useful

Understand different methods of sorting

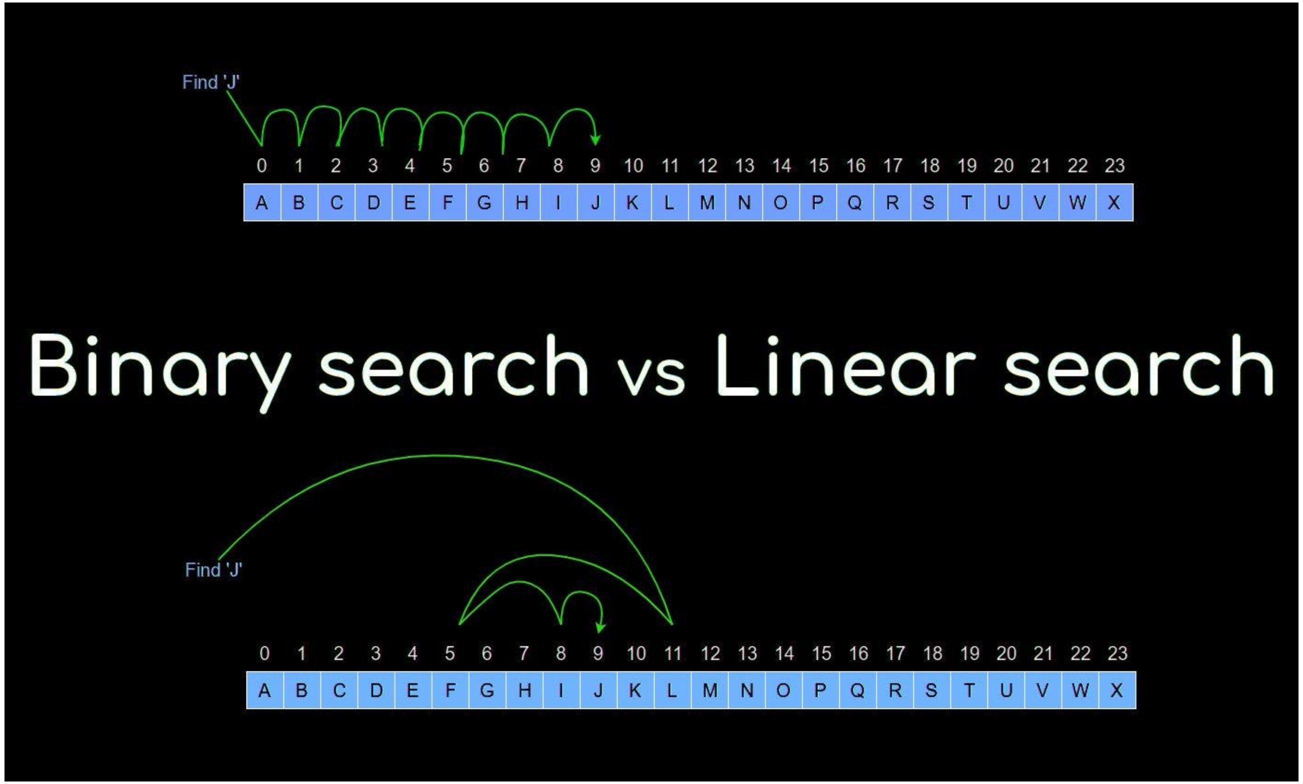
- selection
- bubble
- insert
- merge sort
- python built-in algorithms (quicksort and timsort)

Live coding example

Why and When to Sort

Why is sorting useful?

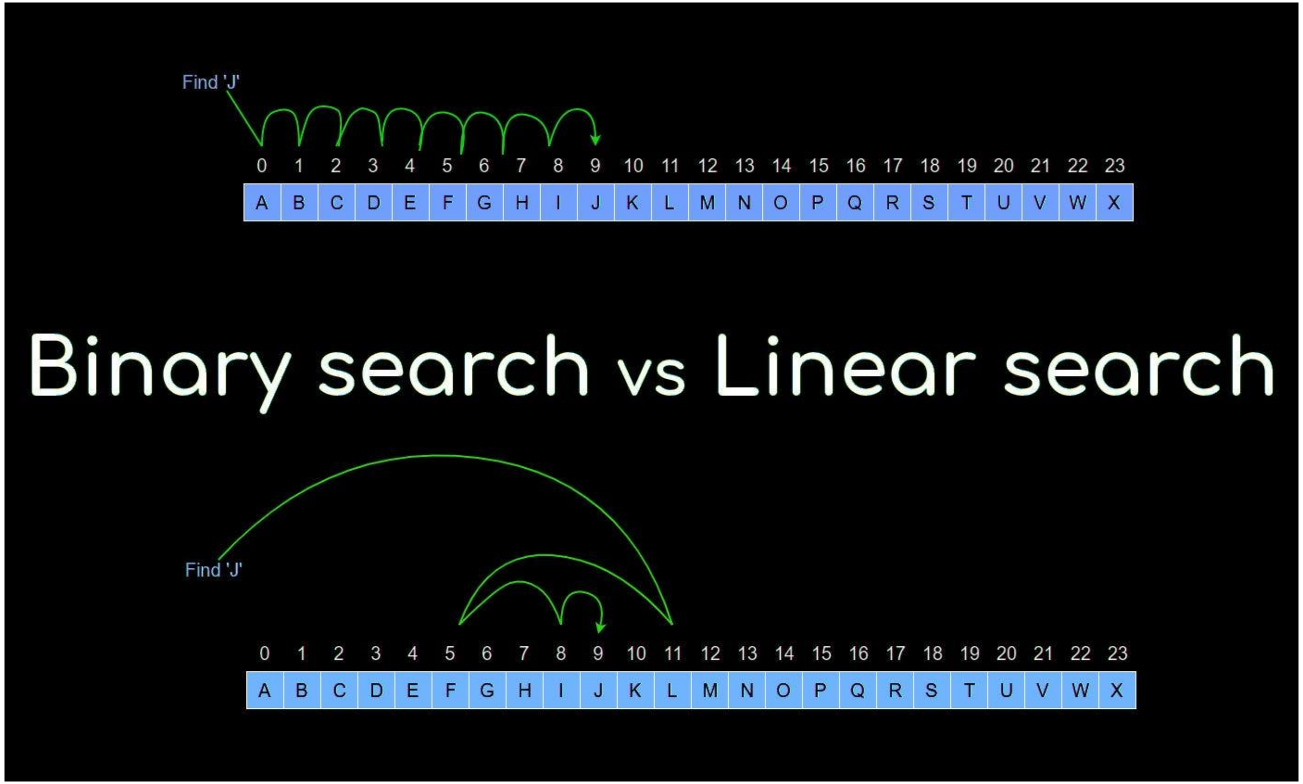
It makes searching easier!



When is sorting useful?

The "cost" can be **amortized**

I.e. sorting can happen just once, but subsequent searching could happen multiple times, diluting the cost.



When is sorting useful?

The "cost" can be **amortized**

- Consider a pair of shoes costing $100
- If you wear the shoes 100 times
- that is $1 per use

You should probably only get the shoes if they will be used a lot

You should only sort if the sorted material will be searched a lot

How to Sort

Algorithm: Selection Sort

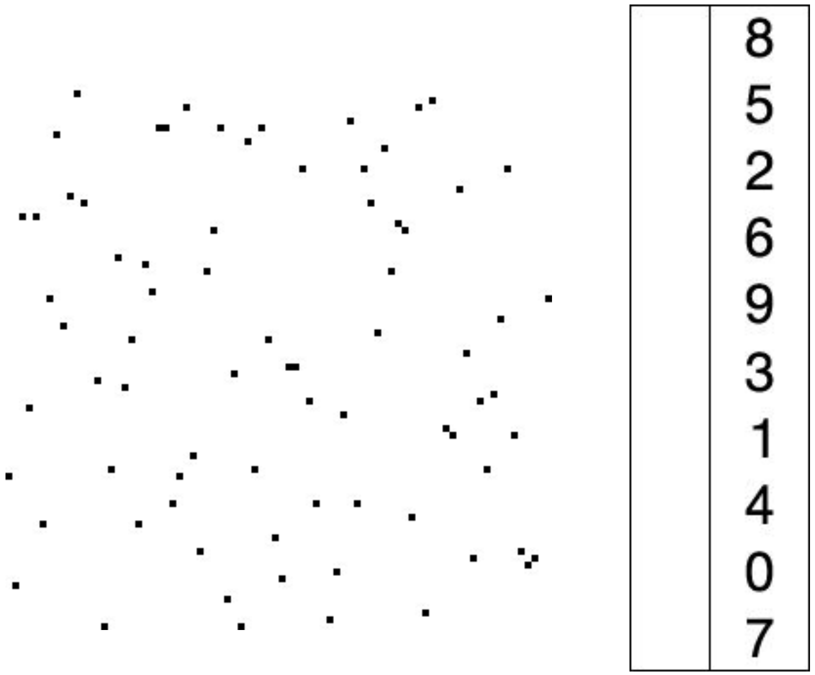
Conceptually, the list can have two parts: **sorted prefix** and **unsorted suffix**

game: select the smallest element in the suffix, and move it to the prefix ending in a loop

- `[8, 3, 8, 9, 6, 4, 8, 2, 2, 1] # unsorted suffix`
- `[8, 3, 8, 9, 6, 4, 8, 2, 2, 1] # locate smallest`
- `[1, 8, 3, 8, 9, 6, 4, 8, 2, 2] # move to prefix ending`
- `[1, 8, 3, 8, 9, 6, 4, 8, 2, 2] # locate smallest`
- `[1, 2, 8, 3, 8, 9, 6, 4, 8, 2] # move to prefix ending`
- `[1, 2, 8, 3, 8, 9, 6, 4, 8, 2] # locate smallest`
- `[1, 2, 2, 8, 3, 8, 9, 6, 4, 8] # move to prefix ending`
- `[1, 2, 2, 8, 3, 8, 9, 6, 4, 8] # locate smallest…etc`

Algorithm: Selection Sort

Repeatedly, **select** smallest and build up prefix



Characteristics

- selecting the smallest takes O(n)
- selection repeats n times!
- if sorting occurs in place, all the shuffling takes even more time!
- Complexity: $O(n^2)$

Algorithm: Bubble Sort

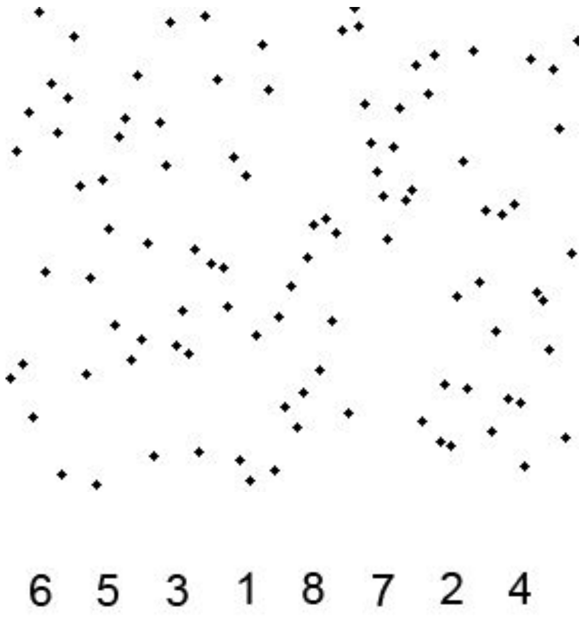
Conceptually, the list can have two parts: **unsorted prefix** and **sorted suffix**

game: find the largest element in prefix via "bubbling" through to the end, in a loop

- `[8, 3, 8, 9, 6, 4, 8, 2, 2, 1] # unsorted prefix`
- `[8, 3, 8, 9, 6, 4, 8, 2, 2, 1] # locate largest`
- `[8, 3, 8, 6, 4, 8, 2, 2, 1, 9] # bubble to the suffix beginning`
- `[8, 3, 8, 6, 4, 8, 2, 2, 1, 9] # locate largest`
- `[3, 8, 6, 4, 8, 2, 2, 1, 8, 9] # bubble to the suffix beginning`
- `[3, 8, 6, 4, 8, 2, 2, 1, 8, 9] # locate largest`
- `[3, 6, 4, 8, 2, 2, 1, 8, 8, 9] # bubble to the suffix beginning`
- `[3, 6, 4, 8, 2, 2, 1, 8, 8, 9] # locate largest…etc`

Algorithm: Bubble Sort

Repeatedly, **bubble** up the largest to suffix



Characteristics

- bubbling requires traversing the list to make decisions about what is largest → O(n)
- bubbling repeats n times!
- Complexity: O(n^2)

Algorithm: Insert Sort

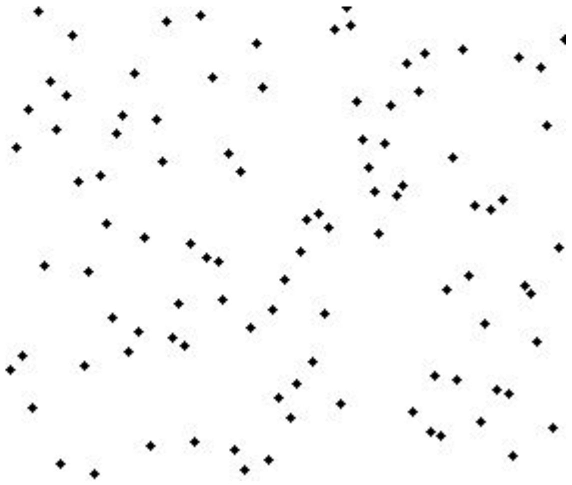
Conceptually, the list can have two parts: **sorted prefix** and **unsorted suffix**

game: get the next element and insert it into the prefix at correct location in a loop

- `[8, 3, 8, 9, 6, 4, 8, 2, 2, 1] # unsorted suffix`
- `[8, 3, 8, 9, 6, 4, 8, 2, 2, 1] # get next element`
- `[8, 3, 8, 9, 6, 4, 8, 2, 2, 1] # insert in prefix after spot found`
- `[8, 3, 8, 9, 6, 4, 8, 2, 2, 1] # get next element`
- `[3, 8, 8, 9, 6, 4, 8, 2, 2, 1] # insert in prefix after spot found`
- `[3, 8, 8, 9, 6, 4, 8, 2, 2, 1] # get next element`
- `[3, 8, 8, 9, 6, 4, 8, 2, 2, 1] # insert in prefix after spot found`
- `[3, 8, 8, 9, 6, 4, 8, 2, 2, 1] # get next element…etc`

Algorithm: Insert Sort

Repeatedly, get next element and **insert** it in prefix



Characteristics

- inserting requires traversing the prefix to make decision about correct location → O(n)
- inserting repeats n times!
- Complexity: O(n^2)

Algorithm: Merge Sort

RECURSIVE!

but first….

Think about whether two sorted lists could be merged easily computationally.

[1,4,6,8,10,11] and [2,3,4,7,9,10]

Algorithm: Merge Sort

RECURSIVE!

Conceptually

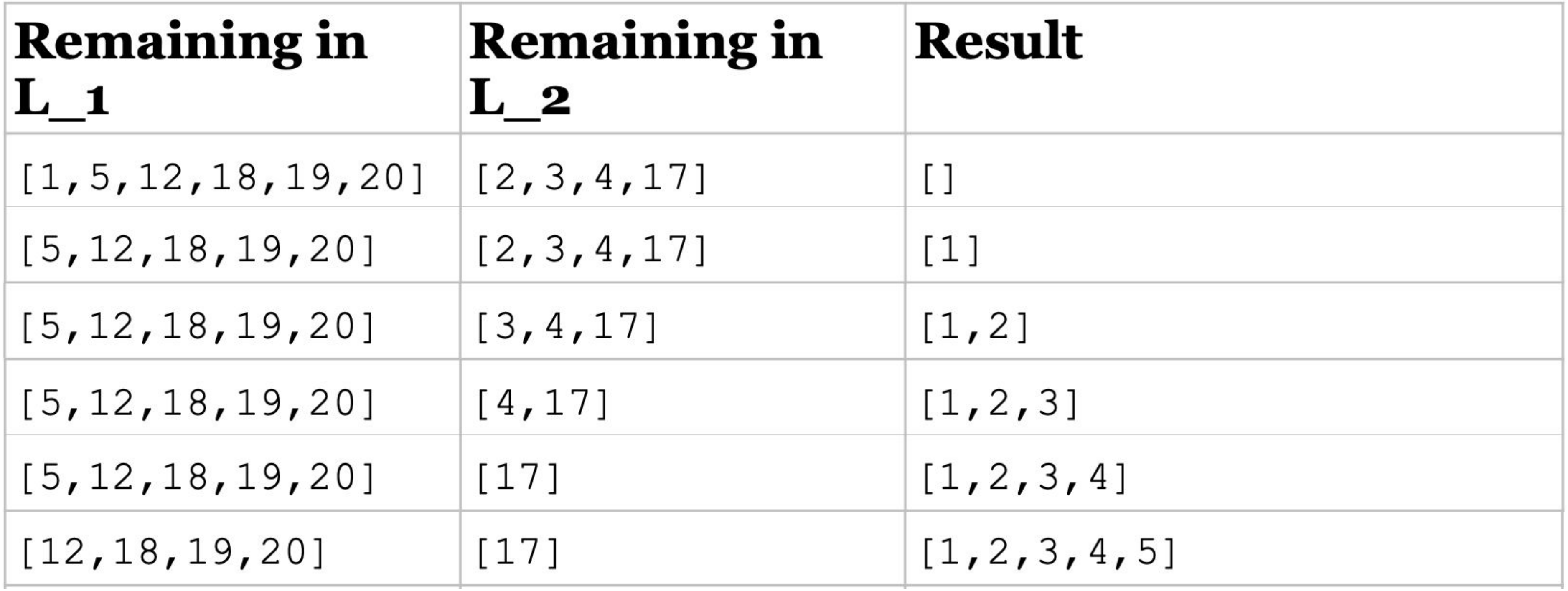
#1. if a list is of length 0 or 1, it is already sorted.

#2. merging two sorted lists is fast

game: split large lists in half repeatedly until sorted, then sort and merge neighboring lists, repeatedly

Algorithm: Merge Sort

Merging two sorted lists:

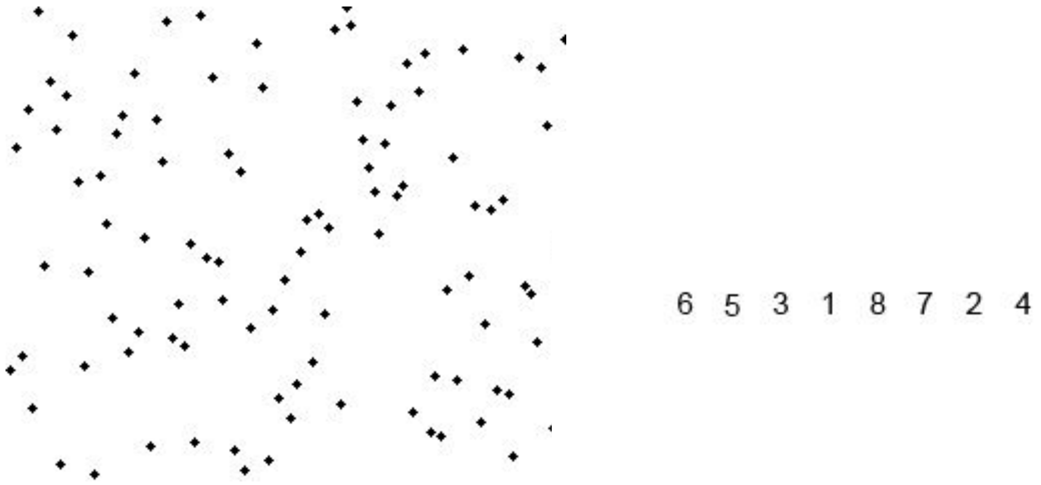


Remaining in L_1	Remaining in L_2	Result
[1,5,12,18,19,20]	[2,3,4,17]	[]
[5,12,18,19,20]	[2,3,4,17]	[1]
[5,12,18,19,20]	[3,4,17]	[1,2]
[5,12,18,19,20]	[4,17]	[1,2,3]
[5,12,18,19,20]	[17]	[1,2,3,4]
[12,18,19,20]	[17]	[1,2,3,4,5]

Algorithm: Merge Sort

RECURSIVE!

split up until **sorted**, then **sort** and **merge** neighboring pairs



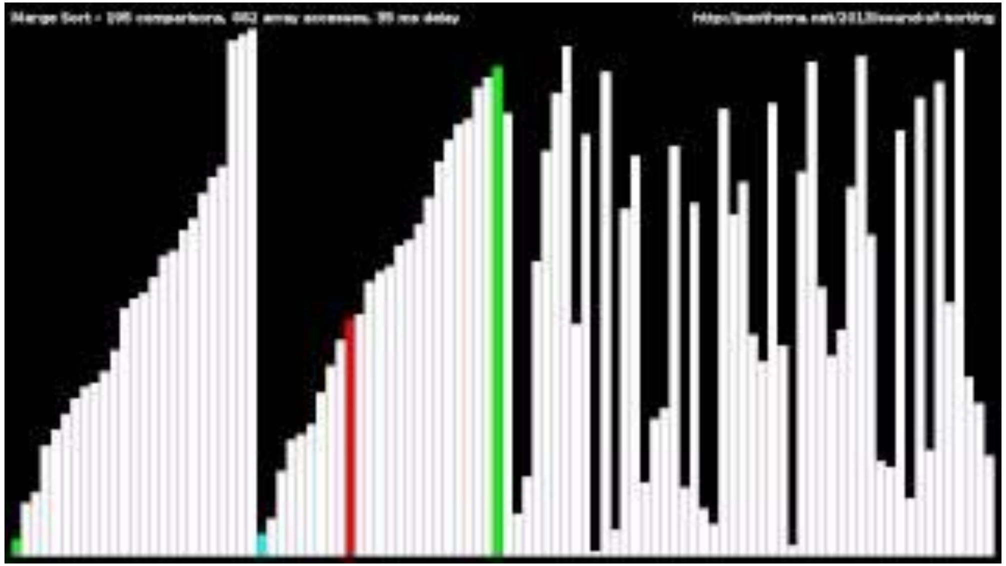
Characteristics

- merging lists takes O(n) time
- merging reduces the number of lists by factor of 2
- total number of merges is log(n)
- Complexity is O(n log(n))

Algorithm: Merge Sort

RECURSIVE!

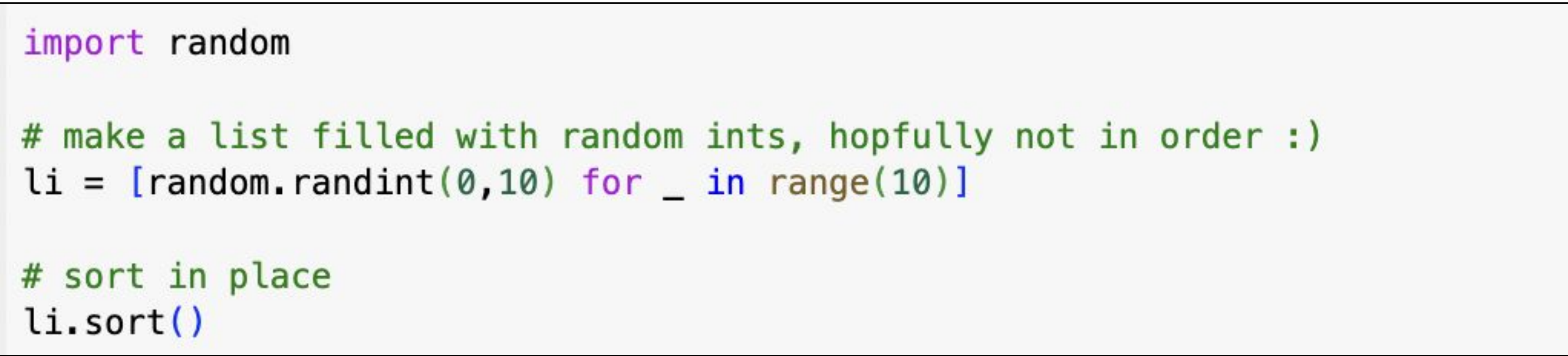
split up until **sorted**, then **sort** and **merge** neighboring pairs



Characteristics

- merging lists takes O(n) time
- merging reduces the number of lists by factor of 2
- total number of merges is log(n)
- Complexity is O(n log(n))

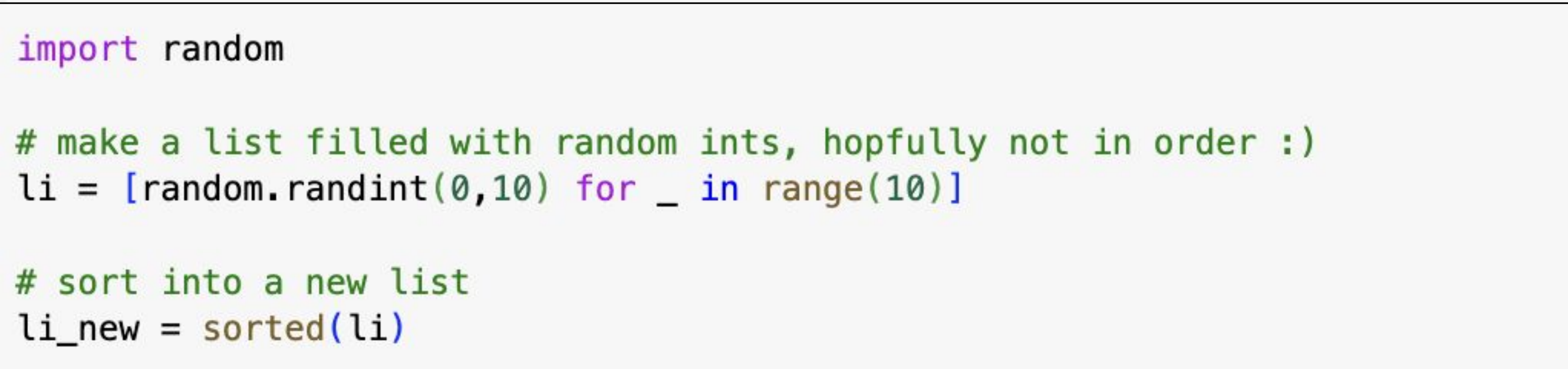
Algorithm: built-in sorting algorithm



```python
import random

# make a list filled with random ints, hopfully not in order :)
li = [random.randint(0,10) for _ in range(10)]

# sort in place
li.sort()
```



```python
import random

# make a list filled with random ints, hopfully not in order :)
li = [random.randint(0,10) for _ in range(10)]

# sort into a new list
li_new = sorted(li)
```

Critical Thinking

- Why is sorting useful?
- What are commonalities in the sorting algorithms?
- What are differences between the algorithms?