Recursion

Recursion

Advanced Sorting Algorithms

Bubble Sort

Bubble Sort

- Loop until no swap is done:

Bubble Sort

- Loop until no swap is done:
- In each pass, loop over every index in list

Bubble Sort

- Loop until no swap is done:
- In each pass, loop over every index in list
- Compare to the item to its right

Bubble Sort

- Loop until no swap is done:
- In each pass, loop over every index in list
- Compare to the item to its right
- If bigger, swap

Bubble Sort

- Loop until no swap is done:
- In each pass, loop over every index in list
- Compare to the item to its right
- If bigger, swap
- Otherwise, do nothing

Selection Sort

Selection Sort

Modify Bubble Sort:

- Loop until no swap is done:
- In each pass, loop over every item in list
- Compare to the item to its left
- *If bigger, swap*
- Otherwise, do nothing

Selection Sort

Modify Bubble Sort:

- Loop until no swap is done:
- In each pass, loop over every item in list
- Compare to the item to its left
- **If bigger, save for later**
- Otherwise, do nothing

Selection Sort

Modify Bubble Sort:

- Loop until no swap is done:
- In each pass, loop over every item in list
- Compare to the item to its left
- **If bigger, save for later**
- Otherwise, do nothing
- **Swap biggest item to end of list**

Insertion Sort

Insertion Sort

Insert each item into the already sorted sub-list

Insertion Sort

Insert each item into the already sorted sub-list

- Loop until all items have been inserted:

Insertion Sort

Insert each item into the already sorted sub-list

- Loop until all items have been inserted:
- In iteration i, insert the ith item into the sublist `myList[:i]`

Insertion Sort

Insert each item into the already sorted sub-list

- Loop until all items have been inserted:
- In iteration i, insert the ith item into the sublist `myList[:i]`
    - Easy to insert into an already-sorted sub-list

Insertion Sort

Insert each item into the already sorted sub-list

- Loop until all items have been inserted:
- In iteration i, insert the ith item into the sublist `myList[:i]`
    - Easy to insert into an already-sorted sub-list
    - sub-list `myList[:i]` is already sorted

Insertion Sort

Insert each item into the already sorted sub-list

- Loop until all items have been inserted:
- In iteration i, insert the ith item into the sublist `myList[:i]`
  - Easy to insert into an already-sorted sub-list
  - sub-list `myList[:i]` is already sorted
  - so coding this insertion is *easy*

Merge Sort

Merge Sort

- Break up `myList` into sub-lists of length 1

Merge Sort

- Break up `myList` into sub-lists of length 1
- A sub-lists of length 1 is already sorted

Merge Sort

- Break up `myList` into sub-lists of length 1
- A sub-lists of length 1 is already sorted
- Merge together sorted sub-lists until everything has been merged

Merge Sort

Merge Sort

```
myList = [8, 5, 10, 1, 4]
```

Merge Sort

```
myList = [8, 5, 10, 1, 4]
```

8 5 10 1 4

Merge Sort

```
myList = [8, 5, 10, 1, 4]
```

8 5 10 1 4

5, 8 1, 10 4

Merge Sort

```
myList = [8, 5, 10, 1, 4]
```

8 5 10 1 4

5, 8 1, 10 4

5, 8 1, 4, 10

Merge Sort

```
myList = [8, 5, 10, 1, 4]
```

8   5   10   1   4

5, 8   1, 10   4

5, 8   1, 4, 10

1, 4, 5, 8, 10

Merge Sort

```
myList = [8, 5, 10, 1, 4]
```

8 5 10 1 4

5, 8 1, 10 4

5, 8 1, 4, 10

1, 4, 5, 8, 10

- $\mathcal{O}(\log n)$ levels

Merge Sort

```
myList = [8, 5, 10, 1, 4]
```

8 5 10 1 4

5, 8 1, 10 4

5, 8 1, 4, 10

1, 4, 5, 8, 10

- $\mathcal{O}(\log n)$ levels
- $\mathcal{O}(n)$ work to merge per level

Merge Sort

```
myList = [8, 5, 10, 1, 4]
```

8 5 10 1 4

5, 8 1, 10 4

5, 8 1, 4, 10

1, 4, 5, 8, 10

- $\mathcal{O}(\log n)$ levels
- $\mathcal{O}(n)$ work to merge per level
- $\mathcal{O}(n \log n)$ total work

Coding Merge Sort

Coding Merge Sort

Merge Sort is tricky to code...

Coding Merge Sort

Merge Sort is tricky to code...

We need to learn about *recursion*

Recursion

Recursion

A **recursive** function is a function that calls itself:

Recursion

A **recursive** function is a function that calls itself:

```
def my_func(n):
    if n > 0:
        return (1 + my_func(n-1))
    else:
        return 0
```

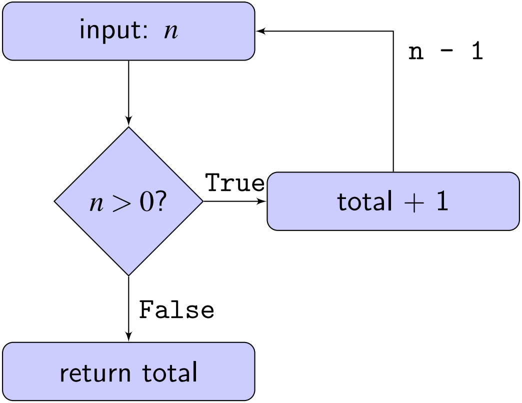
Recursion Loops

Recursion Loops

This is an alternative way to create a loop:

Recursion Loops

This is an alternative way to create a loop:



Anatomy of a Recursive Function

Anatomy of a Recursive Function

```
def my_func(n):
    if n > 0:
        return (1 + my_func(n-1))
    else:
        return 0
```

Anatomy of a Recursive Function

```
def my_func(n):
    if n > 0:
        return (1 + my_func(n-1))
    else:
        return 0
```

Recursive Case/Recursive Call

Anatomy of a Recursive Function

Anatomy of a Recursive Function

```
def my_func(n):
    if n > 0:
        return (1 + my_func(n-1))
    else:
        return 0
```

Anatomy of a Recursive Function

```
def my_func(n):
    if n > 0:
        return (1 + my_func(n-1))
    else:
        return 0
```

Base Case

A Recursive Task

A Recursive Task

$$n! = n \times (n-1) \times (n-2) \times (n-3) \times \ldots \times 1$$

A Recursive Task

$$n! = n \times (n-1) \times (n-2) \times (n-3) \times \ldots \times 1$$

$$n! = n \times (n-1)!$$

Merge Sort

Merge Sort

- Break up `myList` into sub-lists of length 1

Merge Sort

- Break up `myList` into sub-lists of length 1
- A sub-lists of length 1 is already sorted

Merge Sort

- Break up `myList` into sub-lists of length 1
- A sub-lists of length 1 is already sorted
- Merge together sorted sub-lists until everything has been merged

Merge Sort

Merge Sort

- `SortedList = merge(firstHalf, secondHalf)`

Merge Sort

- `SortedList = merge(firstHalf, secondHalf)`
- `firstHalf = merge(firstQuarter, secondQuarter)`

Merge Sort

- `SortedList = merge(firstHalf, secondHalf)`
- `firstHalf = merge(firstQuarter, secondQuarter)`
- `secondHalf = merge(thirdQuarter, fourthQuarter)`