CSCI 241
Scott Wehrwein
Merge Sort: Runtime Analysis
Goals

Know how to derive the worst-case runtime of mergesort.
Mergesort: Runtime

A strategy for analyzing recursive methods:

1. Count work done in a call 
   excluding recursive calls.
2. Multiply by overall number of calls made

```python
def fact(n):
    if n <= 1:
        return n
    return n * fact(n-1)
```

1. O(1) work per call
2. Called once per value in 
   1..n+1 for a total of O(n) work
Mergesort: Runtime

A strategy for analyzing recursive methods:

1. Count work done in a call, *excluding recursive calls*.

2. Multiply by overall number of calls made

```python
/** sort A[start..end] */
def mergeSort(A, start, end):
    if (end-start < 2):
        return
    mid = (end+start)/2
    mergeSort(A,start,mid)
    mergeSort(A,mid, end)
    merge(A, start, mid, end)
```
Mergesort: Runtime

A strategy for analyzing recursive methods:

1. Count work done in a call, ensuring recursive calls.

2. Multiply by overall number of calls made.

```c
/** sort A[start..end] */
mergeSort(A, start, end):
    if (end-start < 2):
        return
    mid = (end+start)/2
    mergeSort(A,start,mid)
    mergeSort(A,mid, end)
merge(A, start, mid, end)
```

O(1) (excluded)

O(??)
Merge: Runtime

Let $n = \text{end} - \text{start}$

\[
\text{merge}(A, \text{start}, \text{mid}, \text{end}) : \\
B = \text{deep copy of } A \\
\text{initialize } i, j, \text{ and } k \\
\text{while neither half is empty} \\
\text{copy the smaller} \\
\text{“front” element into } A \\
copy \text{ any remaining} \\
\text{left half elements} \\
copy \text{ any remaining} \\
\text{right half elements}
Mergesort: Runtime

A strategy for analyzing recursive methods:

1. Count work done in a call, *excluding recursive calls*.

2. Multiply by overall number of calls made

```python
/** sort A[start..end] */
mergeSort(A, start, end):
    if (end-start < 2):
        return
    mid = (end+start)/2
    mergeSort(A,start,mid)
    mergeSort(A,mid, end)
    merge(A, start, mid, end)
```
Mergesort: Runtime

A strategy for analyzing recursive methods:

1. Count work done in a call excluding recursive calls.
2. Multiply by overall number of calls made

**Problem:** sometimes work depends on n, which varies from call to call.
Mergesort: Runtime

/** sort A[start..end] */
mergeSort(A, start, end):
    if (end-start < 2):
        return
        O(1)
        mid = (end+start)/2
        n/2 \rightarrow mergeSort(A, start, mid)
        n/2 \rightarrow mergeSort(A, mid, end)
        O(n)
        merge(A, start, mid, end)
Overall work: $O(n \cdot \log n)$