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

```
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

```
/** 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

```
                /** sort A[start..end] */
                mergeSort(A, start, end):
     O(1)         if (end-start < 2):
                    return
     O(1)         mid = (end+start)/2
(excluded)        mergeSort(A,start,mid)
                  mergeSort(A,mid, end)
     O(??)        merge(A, start, mid, end)
```

Merge: Runtime

Let n = `end - start`

O(end-start)

```
merge(A, start, mid, end):
  B = deep copy of A                      O(n)
  initialize i, j, and k                  O(1)
  while neither half is empty
    copy the smaller
    "front" element into A

  copy any remaining
  left half elements

  copy any remaining
  right half elements
```

O(n)

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

```
                   /** sort A[start..end] */
                   mergeSort(A, start, end):
                     if (end-start < 2):
      O(1)             return
      O(1)           mid = (end+start)/2
 (excluded)          mergeSort(A,start,mid)
                     mergeSort(A,mid, end)
O(end - start)       merge(A, start, mid, end)
```

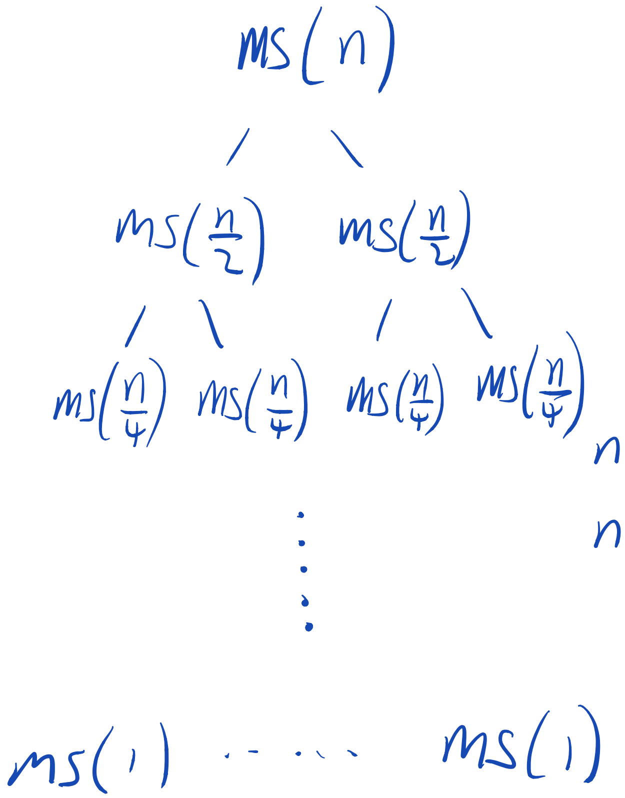
Mergesort: Runtime

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



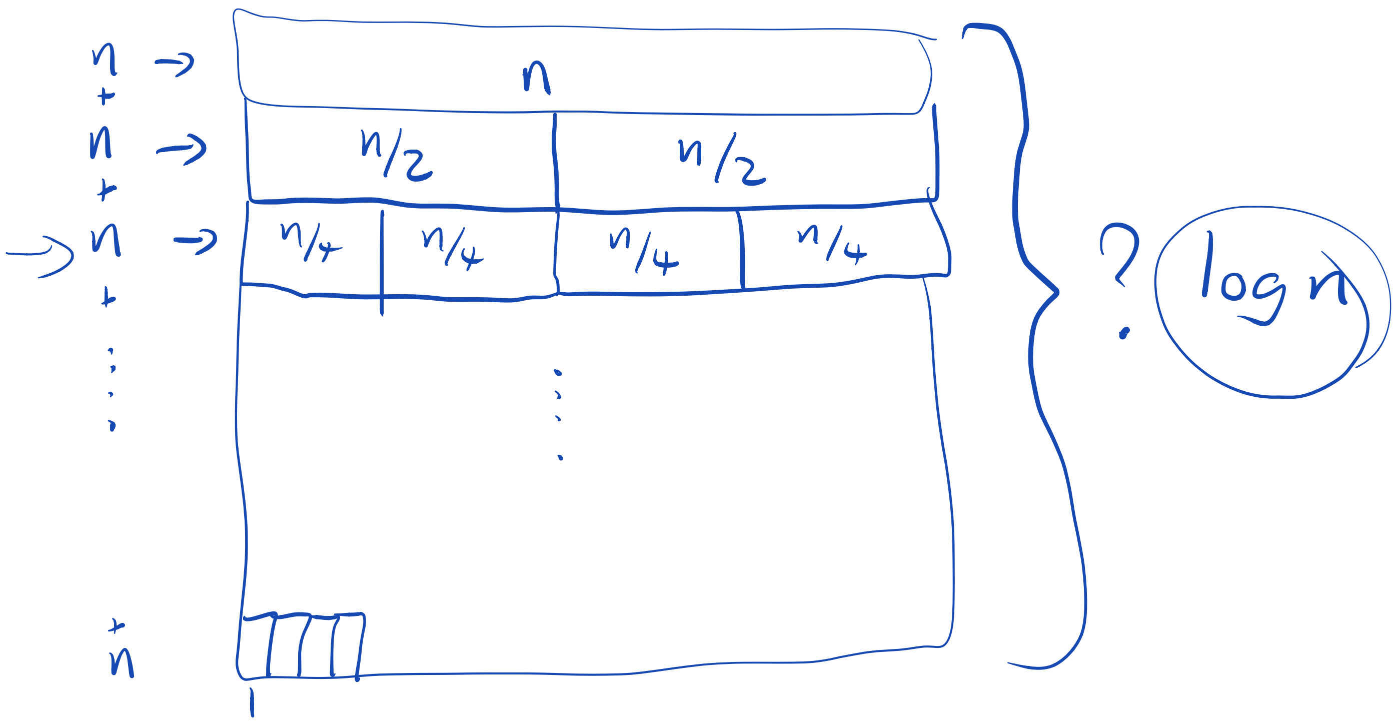
```
/** sort A[start..end] */
mergeSort(A, start, end):
  if (end-start < 2):
    return
  mid = (end+start)/2
  mergeSort(A,start,mid)
  mergeSort(A,mid, end)
  merge(A, start, mid, end)
```

$O(1)$

$n/2 \rightarrow$

$n/2 \rightarrow$

$O(n)$



Overall work: $O(n \cdot \log n)$