10.6.3

Running time analysis of merge-sort: Recursion tree method
Recursion tree

MergeSort(A[1..16])
Recursion tree

MergeSort(A[1..16])

MergeSort(A[1..8])

MergeSort(A[9..16])
Recursion tree

MergeSort(A[1..16])

MergeSort(A[1..8])
  MS(1..4)
  MS(5..8)

MergeSort(A[9..16])
  MS(9..12)
  MS(13..16)
Recursion tree

MergeSort(A[1..16])

MergeSort(A[1..8])

MergeSort(A[9..16])

MergeSort(A[1..4])

MergeSort(A[5..8])

MergeSort(A[9..12])

MergeSort(A[13..16])

MergeSort(A[1..2])

MergeSort(A[3..4])

MergeSort(A[5..6])

MergeSort(A[7..8])

MergeSort(A[9..10])

MergeSort(A[11..12])

MergeSort(A[13..14])

MergeSort(A[15..16])

Har-Peled (UIUC)
Recursion tree: subproblem sizes

MergeSort(A[1..16])

16
Recursion tree: subproblem sizes
Recursion tree: subproblem sizes

```
MergeSort(A[1..16])
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Har-Peled (UIUC) CS374 Fall 2020 67 / 102
Recursion tree: subproblem sizes
Recursion tree: subproblem sizes
Recursion tree: Total work?

16
8 8
4 4 4 4
2 2 2 2 2 2 2 2
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Har-Peled (UIUC)  
CS374  Fall 2020  68 / 102
Running Time

\( T(n) \): time for merge sort to sort an \( n \) element array

\[
T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + cn
\]

What do we want as a solution to the recurrence?

Almost always only an asymptotically tight bound. That is we want to know \( f(n) \) such that \( T(n) = \Theta(f(n)) \).

1. \( T(n) = O(f(n)) \) - upper bound
2. \( T(n) = \Omega(f(n)) \) - lower bound
Running Time

\( T(n) \): time for merge sort to sort an \( n \) element array

\[
T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + cn
\]

What do we want as a solution to the recurrence?

Almost always only an asymptotically tight bound. That is we want to know \( f(n) \) such that \( T(n) = \Theta(f(n)) \).

1. \( T(n) = O(f(n)) \) - upper bound
2. \( T(n) = \Omega(f(n)) \) - lower bound
Running Time

$T(n)$: time for merge sort to sort an $n$ element array

\[ T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + cn \]

What do we want as a solution to the recurrence?

Almost always only an \underline{asymptotically} tight bound. That is we want to know $f(n)$ such that $T(n) = \Theta(f(n))$.

1. $T(n) = O(f(n))$ - upper bound
2. $T(n) = \Omega(f(n))$ - lower bound
Solving Recurrences: Some Techniques

1. Know some basic math: geometric series, logarithms, exponentials, elementary calculus
2. Expand the recurrence and spot a pattern and use simple math
3. Recursion tree method — imagine the computation as a tree
4. Guess and verify — useful for proving upper and lower bounds even if not tight bounds

Albert Einstein: “Everything should be made as simple as possible, but not simpler.”

Know where to be loose in analysis and where to be tight. Comes with practice, practice, practice!

Review notes on recurrence solving.
Solving Recurrences: Some Techniques

1. Know some basic math: geometric series, logarithms, exponentials, elementary calculus
2. Expand the recurrence and spot a pattern and use simple math
3. Recursion tree method — imagine the computation as a tree
4. Guess and verify — useful for proving upper and lower bounds even if not tight bounds

Albert Einstein: “Everything should be made as simple as possible, but not simpler.”

Know where to be loose in analysis and where to be tight. Comes with practice, practice, practice!

Review notes on recurrence solving.
Recursion Trees

MergeSort: \( n \) is a power of 2

Unroll the recurrence. \( T(n) = 2T(n/2) + cn \)
Recursion Trees

MergeSort: n is a power of 2

1. Unroll the recurrence.  \( T(n) = 2T(n/2) + cn \)
2. Identify a pattern.  At the \( i \)th level total work is \( cn \).
Recursion Trees

MergeSort: \( n \) is a power of 2

1. Unroll the recurrence. \( T(n) = 2T(n/2) + cn \)
2. Identify a pattern. At the \( i \)th level total work is \( cn \).
Recursion Trees

MergeSort: \( n \) is a power of 2

1. Unroll the recurrence. \( T(n) = 2T(n/2) + cn \)
2. Identify a pattern. At the \( i \)th level total work is \( cn \).
3. Sum over all levels. The number of levels is \( \log n \).
   So total is \( cn \log n = O(n \log n) \).
Recursion Trees

MergeSort: \( n \) is a power of 2

1. Unroll the recurrence. \( T(n) = 2T(n/2) + cn \)
2. Identify a pattern. At the \( i \)th level total work is \( cn \).
3. Sum over all levels. The number of levels is \( \log n \).
   So total is \( cn \log n = O(n \log n) \).
Recursion Trees

An illustrated example...
Recursion Trees

An illustrated example...

Work in each node
Recursion Trees

An illustrated example...

![Recursion Tree Diagram]

Work in each node
Recursion Trees

An illustrated example...

$$\log n \begin{cases} \frac{cn}{2} + \frac{cn}{2} = cn \\ \frac{cn}{4} + \frac{cn}{4} + \frac{cn}{4} + \frac{cn}{4} = cn \\ \vdots \Rightarrow cn \end{cases}$$
Recursion Trees

An illustrated example...

\[
\log n \begin{cases} 
\frac{cn}{2} + \frac{cn}{4} + \cdots + \frac{cn}{4} \\
\frac{cn}{4} + \frac{cn}{4} + \cdots + \frac{cn}{4} \\
\vdots \\
\frac{cn}{4} + \frac{cn}{4} + \cdots + \frac{cn}{4} \\
\end{cases} = \frac{cn}{2} + \frac{cn}{2} + \cdots + \frac{cn}{2} = cn \\
\] 

\[= cn \log n = O(n \log n)\]
Merge Sort Variant

**Question:** Merge Sort splits into 2 (roughly) equal sized arrays. Can we do better by splitting into more than 2 arrays? Say $k$ arrays of size $n/k$ each?
THE END

...(for now)