Divide and Conquer

Textbook reading

Chapter 2
Overview

**Design principle:**
- Divide and conquer

**Proof technique:**
- Induction, induction, induction

**Analysis technique:**
- Recurrence relations

**Problems:**
- Merge sort
- Selection
- Counting inversions
- Integer multiplication
**Merge Sort**

**MERGE-SORT**\( (A, p, r) \)

1. if \( p < r \)
2. then \( q \leftarrow \lfloor \frac{p+r}{2} \rfloor \)
3. **MERGE-SORT**\( (A, p, q) \)
4. **MERGE-SORT**\( (A, q + 1, r) \)
5. **MERGE**\( (A, p, q, r) \)
**Merge Sort**

**Algorithm**

**MERGE-SORT**($A$, $p$, $r$)

1. if $p < r$
2. then $q \leftarrow \lfloor \frac{p+r}{2} \rfloor$
3. **MERGE-SORT**($A$, $p$, $q$)
4. **MERGE-SORT**($A$, $q + 1$, $r$)
5. **MERGE**($A$, $p$, $q$, $r$)

**O(n log n)**

**Algorithm**

**MERGE**($A$, $p$, $q$, $r$)

1. $n_1 \leftarrow q - p + 1$
2. $n_2 \leftarrow r - q$
3. for $i \leftarrow 1$ to $n_1$
4. do $L[i] \leftarrow A[p + i - 1]$
5. for $i \leftarrow 1$ to $n_2$
6. do $R[i] \leftarrow A[q + i]$
7. $L[n_1 + 1] \leftarrow \infty$
8. $R[n_2 + 1] \leftarrow \infty$
9. $i \leftarrow 1$
10. $j \leftarrow 1$
11. for $k \leftarrow p$ to $r$
12. do if $L[i] \leq R[j]$
13. then $A[k] \leftarrow L[i]$
14. $i \leftarrow i + 1$
15. else $A[k] \leftarrow R[j]$
16. $j \leftarrow j + 1$
Merge Sort: The Micro-View

Divide
Divide
Divide
Do nothing
Base case

Merge
Merge
Merge

Divide
Divide
Divide
Divide

Merge Sort Tree

87 4 17 11 9 13 7 5
87 4 17 11 9 13 7 5
87 4 17 11 9 13 7 5
87 4 17 11 9 13 7 5
4 87 11 17 9 13 5 7
4 11 17 87 5 7 9 13
4 5 7 9 11 13 17 87
Merge Sort: The Macro-View

Divide

Recurse

Merge

inductive hypothesis

inductive step
The Divide-and-Conquer Paradigm

Divide the input instance into one or more smaller instances.

Recursively solve these smaller input instances.

Combine the solutions produced by the recursive calls into a solution to the original instance.
The Divide-and-Conquer Paradigm

**Divide** the input instance into one or more smaller instances.

**Recursively** solve these smaller input instances.

**Combine** the solutions produced by the recursive calls into a solution to the original instance.

In most algorithms, either the divide or the combine step is trivial:

- The divide step in Merge Sort is trivial \(\text{Combining - merge - } O(n)\)
- The combine step in Quick Sort is trivial
Correctness of Merge Sort

Lemma: Merge Sort correctly sorts any input array.
Lemma: Merge Sort correctly sorts any input array.

Proof by induction:

Base case: \((n = 1)\)
Lemma: Merge Sort correctly sorts any input array.

Proof by induction:

Base case: \((n = 1)\)

- A one-element array is already sorted.
Correctness of Merge Sort

Lemma: Merge Sort correctly sorts any input array.

Proof by induction:

Base case: \((n = 1)\)

- A one-element array is already sorted.

Inductive step: \((n > 1)\)
Correctness of Merge Sort

**Lemma:** Merge Sort correctly sorts any input array.

**Proof by induction:**

Base case: \( n = 1 \)
- A one-element array is already sorted.

Inductive step: \( n > 1 \)
- The left and right half each have size less than \( n \).
- By the inductive hypothesis, the recursive calls sort them correctly.
- Merge correctly merges two sorted sequences.
Correctness of D&C Algorithms

*Divide-and-conquer algorithms are the algorithmic incarnation of induction:*

**Base case:** Whenever we don’t recurse, but produce the answer directly (often trivially). 

**Inductive step:** Reduce the solution of a given instance to the solution of smaller instances, by recursing.
Correctness of D&C Algorithms

Divide-and-conquer algorithms are the algorithmic incarnation of induction:

**Base case:** Whenever we don’t recurse, but produce the answer directly (often trivially).

**Inductive step:** Reduce the solution of a given instance to the solution of smaller instances, by recursing.

*Induction is the natural proof method for divide-and-conquer algorithms.*
A *recurrence relation* defines the value of a function $f$ in terms of its values for smaller arguments.
Recurrence Relations

A *recurrence relation* defines the value of a function $f$ in terms of its values for smaller arguments.

**Examples:**

- Fibonacci numbers:

$$f(n) = \begin{cases} 
1 & \text{if } n \leq 2 \\
f(n - 1) + f(n - 2) & \text{if } n > 2 
\end{cases}$$

| $n$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 13 | ...
|-----|---|---|---|---|---|---|---|---|----|---
| $f(n)$ | 1 | 1 | 2 | 3 | 5 | 8 | 13 | 21 | 34 | ...
Recurrence Relations

A recurrence relation defines the value of a function $f$ in terms of its values for smaller arguments.

**Examples:**

- **Fibonacci numbers:**

  $$f(n) = \begin{cases} 
  1 & \text{if } n \leq 2 \\
  f(n - 1) + f(n - 2) & \text{if } n > 2 
  \end{cases}$$

- **Binomial coefficients:**

  $$B(n, k) = \begin{cases} 
  1 & \text{if } k = 1 \text{ or } k = n \\
  B(n - 1, k - 1) + B(n - 1, k) & \text{otherwise} 
  \end{cases}$$

10 things, how many ways can you pick 5 out of them?

$$\binom{10}{5} = 120$$
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>(0)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>l</td>
<td>(0)</td>
</tr>
<tr>
<td></td>
<td>l</td>
<td></td>
<td>l</td>
</tr>
<tr>
<td>l</td>
<td>2</td>
<td></td>
<td>l</td>
</tr>
<tr>
<td>l</td>
<td>3</td>
<td>3</td>
<td>l</td>
</tr>
<tr>
<td>l</td>
<td>4</td>
<td>6</td>
<td>4</td>
</tr>
</tbody>
</table>
A Recurrence for Merge Sort

Analysis:

Recurrence:

\[ T(n) = \begin{cases} \end{cases} \]
A Recurrence for Merge Sort

Analysis:
- If $n = 0$ or $n = 1$, we spend constant time to figure out that there is nothing to do and then exit.

Recurrence:

$$T(n) = \begin{cases} 
\Theta(1) & \text{if } n \leq 1 
\end{cases}$$
A Recurrence for Merge Sort

Analysis:
- If $n = 0$ or $n = 1$, we spend constant time to figure out that there is nothing to do and then exit.
- If $n > 1$, we

Recurrence:

$$T(n) = \begin{cases} 
\Theta(1) & \text{if } n \leq 1 \\
 & \text{if } n > 1 
\end{cases}$$
A Recurrence for Merge Sort

**Analysis:**
- If \( n = 0 \) or \( n = 1 \), we spend constant time to figure out that there is nothing to do and then exit.
- If \( n > 1 \), we
  - Spend constant time to compute the middle index,

**Recurrence:**

\[
T(n) = \begin{cases} 
\Theta(1) & \text{if } n \leq 1 \\
 & \text{if } n > 1 
\end{cases}
\]
A Recurrence for Merge Sort

Analysis:

- If \( n = 0 \) or \( n = 1 \), we spend constant time to figure out that there is nothing to do and then exit.

- If \( n > 1 \), we
  - Spend constant time to compute the middle index,
  - Make one recursive call on the left half, whose size is \( \lceil n/2 \rceil \).

Recurrence:

\[
T(n) = \begin{cases} 
\Theta(1) & \text{if } n \leq 1 \\
T(\lceil n/2 \rceil) & \text{if } n > 1 
\end{cases}
\]
A Recurrence for Merge Sort

Analysis:

- If \( n = 0 \) or \( n = 1 \), we spend constant time to figure out that there is nothing to do and then exit.
- If \( n > 1 \), we
  - Spend constant time to compute the middle index,
  - Make one recursive call on the left half, whose size is \( \lceil n/2 \rceil \),
  - Make one recursive call on the right half, whose size is \( \lfloor n/2 \rfloor \),

Recurrence:

\[
T(n) = \begin{cases} 
\Theta(1) & \text{if } n \leq 1 \\
T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(1) & \text{if } n > 1 
\end{cases}
\]
A Recurrence for Merge Sort

Analysis:
- If \( n = 0 \) or \( n = 1 \), we spend constant time to figure out that there is nothing to do and then exit.
- If \( n > 1 \), we
  - Spend constant time to compute the middle index,
  - Make one recursive call on the left half, whose size is \( \lceil n/2 \rceil \),
  - Make one recursive call on the right half, whose size is \( \lfloor n/2 \rfloor \),
  - Spend linear time to merge the two sorted sequences.

Recurrence:

\[
T(n) = \begin{cases} 
\Theta(1) & \text{if } n \leq 1 \\
T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(1) & \text{if } n > 1
\end{cases}
\]
A Recurrence for Merge Sort

Analysis:

- If $n = 0$ or $n = 1$, we spend constant time to figure out that there is nothing to do and then exit.
- If $n > 1$, we
  - Spend constant time to compute the middle index,
  - Make one recursive call on the left half, whose size is $\lceil n/2 \rceil$,
  - Make one recursive call on the right half, whose size is $\lfloor n/2 \rfloor$,
  - Spend linear time to merge the two sorted sequences.

Recurrence:

$$T(n) = \begin{cases} 
\Theta(1) & \text{if } n \leq 1 \\
T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) & \text{if } n > 1 
\end{cases}$$
A Recurrence for Binary Search

Analysis:

Recurrence:

\[ T(n) = \begin{cases} \end{cases} \]
A Recurrence for Binary Search

Analysis:
- If \( n = 0 \) or \( n = 1 \), we spend constant time to test whether we have found the desired element.

Recurrence:

\[
T(n) = \begin{cases} 
\Theta(1) & \text{if } n \leq 1 
\end{cases}
\]
A Recurrence for Binary Search

Analysis:
- If $n = 0$ or $n = 1$, we spend constant time to test whether we have found the desired element.
- If $n > 1$, we

Recurrence:

$$T(n) = \begin{cases} 
\Theta(1) & \text{if } n \leq 1 \\
 & \text{if } n > 1 
\end{cases}$$
A Recurrence for Binary Search

Analysis:
- If $n = 0$ or $n = 1$, we spend constant time to test whether we have found the desired element.
- If $n > 1$, we
  - Spend constant time to find the middle element of the array and compare it to the search key,

Recurrence:

$$T(n) = \begin{cases} 
\Theta(1) & \text{if } n \leq 1 \\
\Theta(1) & \text{if } n > 1 
\end{cases}$$
A Recurrence for Binary Search

Analysis:
- If $n = 0$ or $n = 1$, we spend constant time to test whether we have found the desired element.
- If $n > 1$, we
  - Spend constant time to find the middle element of the array and compare it to the search key,
  - Make one recursive call on one of the two halves.

Recurrence:

$$T(n) = \begin{cases} 
\Theta(1) & \text{if } n \leq 1 \\
T(\lceil n/2 \rceil) + \Theta(1) & \text{if } n > 1 
\end{cases}$$
A Recurrence for Binary Search

Analysis:
- If $n = 0$ or $n = 1$, we spend constant time to test whether we have found the desired element.
- If $n > 1$, we
  - Spend constant time to find the middle element of the array and compare it to the search key,
  - Make one recursive call on one of the two halves.

Recurrence:

$$T(n) = \begin{cases} 
\Theta(1) & \text{if } n \leq 1 \\
T(\lceil n/2 \rceil) + \Theta(1) & \text{if } n > 1 
\end{cases}$$

Observe how we use an inductive description of the running time of an algorithm that operates inductively. This deserves to be called natural.
The recurrences we use to analyze algorithms will have a base case of

\[ T(n) \leq c \quad \forall n \leq n_0. \]

The exact constants only affect the constant factors in the running time.
The recurrences we use to analyze algorithms will have a base case of

\[ T(n) \leq c \quad \forall n \leq n_0. \]

The exact constants only affect the constant factors in the running time.

Floors and ceilings only affect constant factors.
The recurrences we use to analyze algorithms will have a base case of

\[ T(n) \leq c \quad \forall n \leq n_0. \]

The exact constants only affect the constant factors in the running time.

Floors and ceilings only affect constant factors.

So we are lazy and write:

- **Merge sort:** \( T(n) = 2T(n/2) + \Theta(n) \) 
- **Binary search:** \( T(n) = T(n/2) + \Theta(1) \)
3-way merge sort

3-merge sort (1, \( \frac{1}{3}n \)) \( T(\frac{n}{3}) \)

3-merge sort (\( \frac{1}{3}n+1 \), \( \frac{2}{3}n \)) \( T(\frac{n}{3}) \)

3-merge sort (\( \frac{2}{3}n+1 \), \( n \)) \( T(\frac{n}{3}) \)

merge (1, \( \frac{2}{3}n \)) \( T(\frac{n}{3}) \)

merge (1, \( n \)) \( \Theta(n) \)

\[ T(n) = \begin{cases} 
\Theta(1) & \text{if } n \leq 1 \\
3 \cdot T(\frac{n}{3}) + \Theta(n) & \text{if } n \geq 2 
\end{cases} \]
Given two algorithms \( A \) and \( B \) for the same problem with running times

\[
T_A(n) = 2T(n/2) + \Theta(n) \quad \text{and} \quad T_B(n) = 3T(n/2) + \Theta(\log n)
\]

Which one is faster?
Solving Recurrences

Given two algorithms \( A \) and \( B \) for the same problem with running times

\[
T_A(n) = 2 T(n/2) + \Theta(n)
\]
\[
T_B(n) = 3 T(n/2) + \Theta(\log n)
\]

Which one is faster?

A recurrence as such tells us very little about the running time of the algorithm?
Given two algorithms \( A \) and \( B \) for the same problem with running times

\[
T_A(n) = 2 T(n/2) + \Theta(n)
\]
\[
T_B(n) = 3 T(n/2) + \Theta(\log n)
\]

Which one is faster?

A recurrence as such tells us very little about the running time of the algorithm?

We want to “solve” the recurrence, that is, obtain an expression of the form

\( T(n) = \Theta(n^2) \), \( T(n) = \Theta(n \log n) \) or similar.
Solving Recurrences

Given two algorithms $A$ and $B$ for the same problem with running times

$$T_A(n) = 2T(n/2) + \Theta(n)$$
$$T_B(n) = 3T(n/2) + \Theta(\log n)$$

Which one is faster?

A recurrence as such tells us very little about the running time of the algorithm?

We want to “solve” the recurrence, that is, obtain an expression of the form $T(n) = \Theta(n^2)$, $T(n) = \Theta(n \log n)$ or similar.

Formally, we want an expression $T(n) = \Theta(f(n))$, where $f(n)$ does not depend on $T(n)$. 
Methods to Solve Recurrences

**Substitution**

- Guess what the right answer is.  
  (Intuition, experience, black magic)

- Use induction to prove that the guess is right.
Methods to Solve Recurrences

**Substitution**

- Guess what the right answer is. (Intuition, experience, black magic)
- Use induction to prove that the guess is right.

**Recursion trees**

- Visualize how the recurrence unfolds.
- Use the tree to
  - Obtain a guess, which is then verified using substitution, or
  - Obtain an exact answer if analysis is done sufficiently rigorously.
Methods to Solve Recurrences

**Substitution**

- Guess what the right answer is. (Intuition, experience, black magic)
- Use induction to prove that the guess is right.

**Recursion trees**

- Visualize how the recurrence unfolds.
- Use the tree to
  - Obtain a guess, which is then verified using substitution, or
  - Obtain an exact answer if analysis is done sufficiently rigorously.

**Master theorem**

- Cook book recipe for solving common recurrences. \( T(n) = aT(n/b) + f(n) \)
Lemma: The running time of Merge Sort is $\mathcal{O}(n \lg n)$. 
Substitution Example: Merge Sort

**Lemma:** The running time of Merge Sort is $O(n \lg n)$.

**Recurrence:**

$$T(n) = 2T(n/2) + O(n)$$

2 calls, size $\frac{n}{2}$

actual work (merge)
**Lemma:** The running time of Merge Sort is $O(n \lg n)$.

**Recurrence:**

\[
T(n) = 2T(n/2) + O(n), \text{ that is, }
\]

\[
T(n) \leq 2T(n/2) + an, \text{ for some } a > 0 \text{ and } n \geq n_0.
\]
Substitution Example: Merge Sort

Lemma: The running time of Merge Sort is $O(n \lg n)$.

Recurrence:

$$T(n) = 2T(n/2) + O(n), \text{ that is,}$$

$$T(n) \leq 2T(n/2) + an, \text{ for some } a > 0 \text{ and } n \geq n_0.$$ 

Guess:

$$T(n) \leq cn \lg n, \text{ for some } c > 0 \text{ and } n \geq n_1.$$
Substitution Example: Merge Sort

Lemma: The running time of Merge Sort is $O(n \lg n)$.

Recurrence:

$$T(n) = 2T(n/2) + O(n), \text{ that is,}$$

$$T(n) \leq 2T(n/2) + an, \text{ for some } a > 0 \text{ and } n \geq n_0.$$

Guess:

$$T(n) \leq cn \lg n, \text{ for some } c > 0 \text{ and } n \geq n_1.$$

Base case:

- For $2 \leq n < 4$, $T(n) \leq c' \leq c'n \leq c'n \lg n$, for some $c' > 0$.
- Hence, for $c \geq c'$, $T(n) \leq cn \lg n$.

*pick our $c$ to be larger*
Inductive step: \((n \geq 4)\)
Inductive step: \((n \geq 4)\)

\[
T(n) \leq 2T(n/2) + an
\]
**Inductive step:** \((n \geq 4)\)

Putting in \(\Theta(n \lg n)\) where \(n = \binom{n}{2}\)

\[
T(n) \leq 2T\left(\frac{n}{2}\right) + an
\]

\[
\leq 2 \cdot \frac{cn}{2} \cdot \lg \frac{n}{2} + an \quad \text{by our inductive hypothesis}
\]
**Inductive step:** \((n \geq 4)\)

\[
T(n) \leq 2T(n/2) + an \\
\leq 2 \cdot \frac{cn}{2} \log \frac{n}{2} + an \\
= cn(\log n - 1) + an
\]
**Inductive step:** \((n \geq 4)\)

\[
T(n) \leq 2T(n/2) + an
\]

\[
\leq 2 \cdot \frac{cn}{2} \cdot \frac{n}{2} + an
\]

\[
= cn(\log n - 1) + an \leq cn \log n + cn + an
\]

\[
= cn \log n + (a - c)n
\]
**Inductive step:** \((n \geq 4)\)

\[
T(n) \leq 2T(n/2) + an
\]

\[
\leq 2 \cdot \frac{cn}{2} \lg \frac{n}{2} + an
\]

\[
= cn(\lg n - 1) + an
\]

\[
= cn \lg n + (a - c)n
\]

\[
\leq cn \lg n, \text{ for all } c \geq a.
\]

\[
\Rightarrow cn \lg n + (\text{negligible})n
\]

\[
\leq cn \lg n + 1 \quad \text{WRONG}
\]
**Inductive step:** \((n \geq 4)\)

\[
T(n) \leq 2T\left(\frac{n}{2}\right) + an
\]

\[
\leq 2 \cdot \frac{cn}{2} \lg \frac{n}{2} + an
\]

\[
= cn(\lg n - 1) + an
\]

\[
= cn \lg n + (a - c)n
\]

\[
\leq cn \lg n, \text{ for all } c \geq a.
\]

**Notes:**

- Since the base case is valid only for \(n \geq 2\), we can apply the induction hypothesis only to \(n \geq 2\). This is why the inductive step starts at \(n \geq 4\).
**Inductive step:** \((n \geq 4)\)

\[
T(n) \leq 2T(n/2) + an
\]

\[
\leq 2 \cdot \frac{cn}{2} \lg \frac{n}{2} + an
\]

\[
= cn(\lg n - 1) + an
\]

\[
= cn \lg n + (a - c)n
\]

\[
\leq cn \lg n, \text{ for all } c \geq a.
\]

**Notes:**

- Since the base case is valid only for \(n \geq 2\), we can apply the induction hypothesis only to \(n \geq 2\). This is why the inductive step starts at \(n \geq 4\).

- We only proved the upper bound. The lower bound can be proved similarly, but usually needs to be done separately.
Lemma: The running time of Binary Search is $O(\lg n)$. 
Lemma: The running time of Binary Search is $O(\lg n)$.

Recurrence:

$$T(n) = T(n/2) + O(1),$$
that is,

$$T(n) \leq T(n/2) + a,$$ for some $a > 0$ and $n \geq n_0$. 

($a$ actual amount of work in one call)
Lemma: The running time of Binary Search is $O(\lg n)$.

Recurrence:

$$T(n) = T(n/2) + O(1), \text{ that is,}$$
$$T(n) \leq T(n/2) + a, \text{ for some } a > 0 \text{ and } n \geq n_0.$$

Guess:

$$T(n) \leq c \lg n, \text{ for some } c > 0 \text{ and } n \geq n_1.$$
Substitution Example: Binary Search

**Lemma:** The running time of Binary Search is $O(\lg n)$.

**Recurrence:**

\[
T(n) = T(n/2) + O(1), \text{ that is, } \quad T(n) \leq T(n/2) + a, \text{ for some } a > 0 \text{ and } n \geq n_0.
\]

**Guess:**

\[
T(n) \leq c \lg n, \text{ for some } c > 0 \text{ and } n \geq n_1.
\]

**Base case:**

- For $2 \leq n < 4$, $T(n) \leq c' \leq c' \lg n$, for some $c' > 0$.
- Hence, for $c \geq c'$, $T(n) \leq c \lg n$. 
Inductive step: \((n \geq 4)\)

\[ T(n) \leq T(n/2) + a \]
**Inductive step:** \((n \geq 4)\)

\[
T(n) \leq T\left(\frac{n}{2}\right) + a \\
\leq c \lg \frac{n}{2} + a \quad \text{inductive hypothesis} \\
\Gamma\left(\frac{n}{2}\right) \leq c \lg \frac{n}{2}
\]
Inductive step: $n \geq 4$

\[
T(n) \leq T(n/2) + a \\
\leq c \log \frac{n}{2} + a \\
= c(\log n - 1) + a
\]
**Inductive step:** \((n \geq 4)\)

\[
T(n) \leq T(n/2) + a
\leq c \lg \frac{n}{2} + a
= c(\lg n - 1) + a
= c \lg n + (a - c)
\]
Inductive step: \((n \geq 4)\)

\[
T(n) \leq T\left(\frac{n}{2}\right) + a
\]

\[
\leq c \lg \frac{n}{2} + a
\]

\[
= c(\lg n - 1) + a
\]

\[
= c \lg n + (a - c)
\]

\[
\leq c \lg n, \text{ for all } c \geq a.
\]

\(c\) is fixed but arbitrary.
A Recursion Tree for Merge Sort

**Strategy:** Expand the recurrence

\[ T(n) = 2T(n/2) + \Theta(n) \]
**Strategy:** Expand the recurrence

\[ T(n) = 2T(n/2) + \Theta(n) \]
Strategy: Expand the recurrence

\[ T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n) \]
A Recursion Tree for Merge Sort

**Strategy:** Expand the recurrence

\[ T(n) = 2T(n/2) + \Theta(n) \]

![Recursion Tree Diagram](image-url)
Strategy: Expand the recurrence

\[ T(n) = 2T(n/2) + \Theta(n) \]
**Strategy:** Expand the recurrence

\[ T(n) = 2 T(n/2) + \Theta(n) \]

Running time is the sum of the running times for each recursive call.
**Strategy:** Expand the recurrence

\[ T(n) = 2T(n/2) + \Theta(n) \]

![Recursion Tree for Merge Sort](image-url)
**A Recursion Tree for Merge Sort**

**Strategy:** Expand the recurrence

\[ T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n) \]

![Recursion Tree](image)
**Strategy:** Expand the recurrence

\[ T(n) = 2T(n/2) + \Theta(n) \]
A Recursion Tree for Merge Sort

Strategy: Expand the recurrence

\[ T(n) = 2T(n/2) + \Theta(n) \]
**Strategy:** Expand the recurrence

\[
T(n) = 2T(n/2) + \Theta(n)
\]

A Recursion Tree for Merge Sort

---

CSCI 3110 • Divide and Conquer • 19/39
A Recursion Tree for Merge Sort

Strategy: Expand the recurrence

\[ T(n) = 2T(n/2) + \Theta(n) \]
Strategy: Expand the recurrence

\[ T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n) \]

A Recursion Tree for Merge Sort
A Recursion Tree for Merge Sort

**Strategy:** Expand the recurrence

\[
T(n) = 2T(n/2) + \Theta(n)
\]

**Solution:** \(T(n) = \Theta(n \log n)\)
Recurrence: $T(n) = T(n/2) + \Theta(1)$
**A Recursion Tree for Binary Search**

*Recurrence:* \( T(n) = T(n/2) + \Theta(1) \)

\[
\begin{align*}
\text{lg } n
\end{align*}
\]

\[
\begin{align*}
\vdots
\end{align*}
\]

\[
\begin{align*}
= a \text{lg } n + b = \Theta(\text{lg } n)
\end{align*}
\]
A Recursion Tree for Binary Search

**Recurrence:** \( T(n) = T(n/2) + \Theta(1) \)

**Solution:** \( T(n) = \Theta(\lg n) \)
A Less Obvious Recursion Tree

**Recurrence:** \( T(n) = T(n/3) + T(2n/3) + \Theta(n) \)

\[
\begin{align*}
\log_{3/2} n & \quad \log_3 n \\
\frac{8n}{27} & \quad \frac{n}{27} \\
\frac{4n}{9} & \quad \frac{n}{9} \\
\frac{2n}{3} & \quad an & \frac{n}{3} \\
n & \quad \frac{n}{3} \\
\end{align*}
\]

\[
an \log_{3/2} n + bn = \Theta(n \log n)
\]

\[
\log_{3/2} n = \Theta(\log_3 n)
\]
A Less Obvious Recursion Tree

**Recurrence:** \( T(n) = T(n/3) + T(2n/3) + \Theta(n) \)
**Recurrence:** \( T(n) = T(n/3) + T(2n/3) + \Theta(n) \)

**Solution:** \( T(n) = \Theta(n \log n) \)
Sometimes Only Substitution Will Do

**Recurrence:** \( T(n) = 2T(n/3) + T(n/2) + \Theta(n) \)
Sometimes Only Substitution Will Do

Recurrence: \( T(n) = 2T(n/3) + T(n/2) + \Theta(n) \)

\[
\begin{align*}
\frac{n}{3} & \quad \frac{n}{3} & \quad \frac{n}{2} \\
\frac{n}{3} & \quad \frac{n}{3} & \quad \frac{2n}{6} + \frac{2n}{6} + \frac{3n}{6} = \frac{7n}{6} \\
\log_2 n & \quad \log_3 n
\end{align*}
\]

\[i\text{-th level: } \Theta(n \cdot (7/6)^i)\]
Sometimes Only Substitution Will Do

**Recurrence:** \( T(n) = 2T(n/3) + T(n/2) + \Theta(n) \)

\[ \log_2 n \quad \{ \quad \text{i-th level: } \Theta(n \cdot (7/6)^i) \quad \} \]

\[ \log_3 n \]

**Lower bound:** \( T(n) = \Omega(n^{1+\log_3(7/6)}) \approx \Omega(n^{1.14}) \)
Sometimes Only Substitution Will Do

Recurrence: \[ T(n) = 2T(n/3) + T(n/2) + \Theta(n) \]

Lower bound: \[ T(n) = \Omega(n^{1+\log_3(7/6)}) \approx \Omega(n^{1.14}) \]

Upper bound: \[ T(n) = O(n^{1+\log_2(7/6)}) \approx O(n^{1.22}) \]
\[ T(n) = a T\left(\frac{n}{b}\right) + f(n) \]

\[ = 5 T\left(\frac{n}{3}\right) + \Theta(n^2) \]

\[
\begin{align*}
\frac{1}{l} & \quad a n^2 & \quad \frac{5}{q} & \quad \frac{5^6}{q^6} & \quad \frac{n}{3} & \quad \frac{n}{3} & \quad \frac{n}{3} & \quad \frac{n}{3} \\
\frac{5}{q} & \quad a n^2 & \quad \frac{5^1}{q^1} & \quad \frac{n}{3} & \quad \frac{n}{3} & \quad \frac{n}{3} & \quad \frac{n}{3} \\
\frac{125}{q} & \quad a n^2 & \quad \frac{5^3}{q^3} & \quad \frac{n}{9} & \quad \frac{n}{9} \\
q & \quad \text{level } i & \quad \frac{5^i}{q^i} & \quad an^2
\end{align*}
\]

\[ q \left(\frac{n}{3}\right)^2 = \frac{an^2}{q^i} \]

\[ q \left(\frac{n}{q}\right)^2 = \frac{an^2}{81} \]

\[ \text{height} = \log_3 n \]

\[ \frac{5 a n^2}{q} \]

\[ \frac{125 a n^2}{81} \]
Total running: \( \sum_{i=0}^{\log_3 n} \left( \frac{5}{9^i} \right) a_n^2 = \Theta(n^2) \)
\[ T(n) = 2T(n-1) + c = \Theta(2^n) \]

\[
\begin{array}{c}
\text{n} \\
\text{n-1} \quad \text{1} \\
\text{n-1} \quad \text{n-1} \quad \text{2} \\
\text{n-2} \quad \text{n-2} \quad \text{n-2} \quad \text{4} \\
\text{o..o} \quad \text{#leaves} \quad 2^n \\
\end{array}
\]
break into smaller problems
  divide

  recursively solve

  Combine

  conquer
Analysis: Recurrence Relations

recursive running time

\( T(n) = \) running time on a problem of size \( n \)

\text{merge-sort} (\text{start}, \text{end}, A)

\( O(1) \) \hspace{1em} \text{let } m \text{ be the middle} \hspace{1em} \text{base-case return} \hspace{1em} \text{if } \text{start} = \text{end} \n
\( T(n/2) \) \hspace{1em} \text{merge-sort} (\text{start}, \text{middle}) \n
\( T(n/2) \) \hspace{1em} \text{merge-sort} (\text{middle}+1, \text{end}) \n
\( O(n) \) \hspace{1em} \text{merge} (\text{start}, \text{middle}, \text{end}) \n
\text{Total} = O(1) + T(n/2) + \overline{T}(n/2) + O(n) = 2T(n/2) + O(n)
Substitution - need a guess

Recurrence Trees -

Master Theorem
3 examples

1. $T(n) = 1T(n/3) + O(n)$

2. $T(n) > 2T(n/3) + O(n)$

3. $T(n) = 3T(n/3) + O(n)$
\[ T(n) = 2T\left(\frac{n}{3}\right) + O(n) \]

\[ \text{Leaves} = n \log_2 3 = n \]

\[ \frac{a_n}{e} + \frac{a_n}{c} = a_n \]

\[ T_{\text{Total}} = \sum_{i=0}^{\log n} a_n + bn = O(n \log n) \]
\[ T(n) = \Theta \left( \sum_{i=1}^{\log_2 n} a \cdot \frac{n}{i} \right) \]

\[ \sum_{i=1}^{\log_2 n} a \cdot \frac{n}{i} + b \]

\[ a \cdot \frac{n}{2} \]

\[ a \cdot \frac{n}{4} \]

\[ \cdots \]

\[ a \cdot \frac{n}{8} \]

\[ n + \frac{n}{2} + \frac{n}{4} + \cdots \sim 2n \]

\[ = O(2n) = O(n) \]

\[ b \cdot 1 \]
\[ T(n) = 3T\left(\frac{n}{3}\right) + O(n) \]

\[
\sum_{i=1}^{\log_2 n} \left(\frac{3}{2}\right)^i a_n + \frac{\log n}{n} a_n = \frac{1}{n} \log_2 3
\]

\[
\text{#leaves} = \frac{\log n}{\log 2}
\]

\[
\frac{3^{\log_2 n} a_n}{2^{\log_2 n}} = 1.5 \log_2 a_n
\]

Compare this with \( n \log_2 3 \).
**Master Theorem**

**Theorem: (Master Theorem)**

Let \( a \geq 1 \) and \( b > 1 \), let \( f(n) \) be a function over the positive integers, and let \( T(n) \) be given by the following recurrence:

\[
T(n) = a \, T\left(\frac{n}{b}\right) + f(n)
\]

(i) If \( f(n) = \mathcal{O}(n^{\log_b a - \epsilon}) \), for some \( \epsilon > 0 \), then \( T(n) = \Theta(n^{\log_b a}) \).  

(ii) If \( f(n) = \Theta(n^{\log_b a}) \), then \( T(n) = \Theta(n^{\log_b a} \lg n) \).  

(iii) If \( f(n) = \Omega(n^{\log_b a + \epsilon}) \), for some \( \epsilon > 0 \), and \( a \, f\left(\frac{n}{b}\right) \leq c \, f(n) \), for some \( c < 1 \) and all \( n \geq n_0 \), then \( T(n) = \Theta(f(n)) \).
\[ T(n) = 2T\left( \frac{n}{3} \right) + O(n) \]

\[ a = 2 \quad b = 2 \quad f(n) = n \]

Start: compare \( f(n) \) with \( n^{\log_b a} \)

\[ = n \quad = n^{\log_2 2} = n \]

\[ = \text{case 2!} \]

\[ \therefore T(n) = \Theta(n^{\log_2 2}) \]
\[ T(n) = 1 T\left( \frac{n}{2} \right) + f(n) \]

\[ a = 1 \quad b = 2 \quad f(n) = n \]

\[ f(n) = n \quad n^{\log_{10} 2} = n^{\log_{2} n} = n^n = 1 \]

\[ n = \log(1) \]

Case 3: extra condition

we need to prove that \( a f\left( \frac{n}{b} \right) \leq c f(n) \)

\[ T(n) = \Theta(f(n)) = \Theta(n) \quad \text{if} \quad 1 \left( \frac{n}{b} \right) \leq Cn \]

\[ \frac{n}{b} \leq C \cdot n \]
\[ T(n) = 3T\left(\frac{n}{2}\right) + O(n) \]

where:
\[ a = 3 \quad b = 2 \quad f(n) = n \]

\[ f(n) = n \quad n \log_b a = n \log_2 3 = n \]

\[ n = \Theta(n \log_2 3) \]

**Case 1**

\[ \therefore T(n) = \Theta(n \log_2 3) \]
\[ T(n) = 4T\left( \frac{n}{2} \right) + \Theta(n^2) \]

\[ f(n) = \Theta(n^2) \]

\[ n \log_b a = n \log_2 4 = n^2 \]

\[ T(n) = \Theta(n^2 \log n) \]

---

*Master Theorem does not work for*

\[ T(n) = T\left( \frac{n}{3} \right) + T\left( \frac{2n}{3} \right) + \Theta(n) \]
\[ a^{\log_b n} = n^{\log_b a} \]
Selection

Given:
- An array storing \( n \) numbers, \( x_1 \leq x_2 \leq \cdots \leq x_n \), in any order
- A parameter \( 1 \leq k \leq n \)

To compute: \( x_k \)

Element \( x_k \) is also referred to as the \( k \)-th order statistic of the set \( \{x_1, x_2, \ldots, x_n\} \).

Example:

\[
\text{Given: } \quad \begin{array}{c}
16 & 3 & 5 & 21 & 8 & 10 & 7 & 17 \\
\not{\text{sorted}}
\end{array}
\]

The 4-th order statistic is 8 because

\[
\begin{array}{c}
3 & 5 & 7 & 8 & 10 & 16 & 17 & 21 \\
\end{array}
\]
Two Simple Solutions

Repeated Minimum:

- Find and remove the minimum, repeat $k$ times.

\[
16 \ 3 \ 5 \ 21 \ 8 \ 16 \ 7 \ 17
\]

4th order statistic

\[
\begin{align*}
\text{min} &= 16 \ 3 \ \theta(n) \\
\text{min} &= 16 \ 5 \\
\text{min} &= 16 \ 8 \\
\text{min} &= 8
\end{align*}
\]

$O(hn)$ if $h = \Theta(h\log n)$

$O(n\log n)$
Two Simple Solutions

Repeated Minimum:

- Find and remove the minimum, repeat $k$ times.

Running time:
Two Simple Solutions

Repeated Minimum:

- Find and remove the minimum, repeat $k$ times.

Running time: $\Theta(kn)$
Two Simple Solutions

Repeated Minimum:

- Find and remove the minimum, repeat $k$ times.

Running time: $\Theta(kn)$

Sort and select:

- Sort the sequence
- Report the $k$-th item in the sorted sequence
Two Simple Solutions

Repeated Minimum:
- Find and remove the minimum, repeat \( k \) times.

Running time: \( \Theta(nk) \)

Sort and select:
- Sort the sequence
- Report the \( k \)-th item in the sorted sequence

Running time:
Two Simple Solutions

Repeated Minimum:
- Find and remove the minimum, repeat $k$ times.

Running time: $\Theta(kn)$

Sort and select:
- Sort the sequence
- Report the $k$-th item in the sorted sequence

Running time: $\Theta(n \lg n)$
The Hunt for Intuition: Quicksort

**Quicksort**

1. if $|A| \leq 1$
2. then return $A$
3. else $p \leftarrow$ median of $A$
4. Partition $A$ into three pieces:
   - $L = \{ x \in A \mid x < p \}$
   - $\{p\}$
   - $R = \{ x \in A \setminus \{p\} \mid x \geq p \}$
5. $L' \leftarrow \text{Quicksort}(L)$
6. $R' \leftarrow \text{Quicksort}(R)$
7. return $L' \circ \{p\} \circ R'$

$O(\log n)$

**Complicated divide**

**Simple combine**

Sorted
The Hunt for Intuition: Quicksort

**Quicksort**

1. if $|A| \leq 1$
2. then return $A$
3. else $p \leftarrow$ median of $A$ (hope for $\Theta(n)$)
4. Partition $A$ into three pieces:
   - $L = \{x \in A \mid x < p\}$
   - $\{p\}$
   - $R = \{x \in A \setminus \{p\} \mid x \geq p\}$
5. $L' \leftarrow \text{Quicksort}(L)$
6. $R' \leftarrow \text{Quicksort}(R)$
7. return $L' \circ \{p\} \circ R' \circ \Theta(1)$

$T(n) = \Theta(n) + \Theta(\text{median})$

$T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right)$
\[ T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + \Theta(n) \]

\begin{align*}
\Theta(n) & \quad \Theta(n) \\
\Theta(n^2) & \quad \Theta(n^2) \\
\log_3 n & \quad \log_3 n \\
\text{height} = n & \quad \text{height} = n \\
\frac{1}{3}n & \quad \frac{2}{3}n & \text{total } n \\
\frac{1}{3}n & \quad \frac{1}{3}n & \frac{1}{3}n & \frac{1}{3}n & \text{total } n \\
\frac{1}{3}n & \quad \frac{2}{3}n & \frac{1}{3}n & \frac{1}{3}n & \frac{1}{3}n & \text{total } n \\
\end{align*}
\[
5 \div 3 = 1.66666666667
\]

\[
\begin{array}{c}
\emptyset \quad 5324 \\
\hline
\emptyset \quad 534 \\
\emptyset \quad 34
\end{array}
\]

\[
\frac{nr}{n-1 + \alpha + \ldots + O(n)}
\]
\[ 3 \ 7 \ 6 \ 4 \ 5 \]
\[ 3 \ 4 \ 1 \ 5 \ 7 \ 6 \]
\[ L = 134 \quad P \quad R = 67 \]
\[ 1 \ 3 \ 4 \quad 6 \ 7 \ 0 \]
\[ L \quad P \quad R \quad L \quad P \quad R \]
\[ \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \]
\[ \text{base case} \quad \text{base case} \quad \text{base case} \quad \text{base case} \]
**The Hunt for Intuition: Quicksort**

**QUICKSORT**(\(A\))

1. if \(|A| \leq 1\) then return \(A\)
2. else \(p \leftarrow \text{median of } A\)
3. Partition \(A\) into three pieces:
   - \(L = \{x \in A \mid x < p\}\)
   - \(\{p\}\)
   - \(R = \{x \in A \setminus \{p\} \mid x \geq p\}\)
4. \(L' \leftarrow \text{QUICKSORT}(L)\)
5. \(R' \leftarrow \text{QUICKSORT}(R)\)
6. return \(L' \circ \{p\} \circ R'\)

**Assumption:** We know how to find the median in \(\Theta(n)\) time.

**Running time:** \(T(n) = \)
Quicksort

1. if $|A| \leq 1$
2. then return $A$
3. else $p \leftarrow$ median of $A$
4. Partition $A$ into three pieces:
   - $L = \{x \in A \mid x < p\}$
   - $\{p\}$
   - $R = \{x \in A \setminus \{p\} \mid x \geq p\}$
5. $L' \leftarrow$ Quicksort($L$)
6. $R' \leftarrow$ Quicksort($R$)
7. return $L' \circ \{p\} \circ R'$

Assumption: We know how to find the median in $\Theta(n)$ time.

Running time: $T(n) = 2T(n/2) + \Theta(n)$

$\leq \Theta(n \log n)$
**The Hunt for Intuition: Quicksort**

**QUICKSORT**\( (A) \)

1. if \(|A| \leq 1\)  
2. then return \(A\)  
3. else \(p \leftarrow \text{median of } A\)  
4. Partition \(A\) into three pieces:
   - \(L = \{ x \in A \mid x < p \}\)  
   - \(\{p\}\)  
   - \(R = \{ x \in A \setminus \{p\} \mid x \geq p \}\)  
5. \(L' \leftarrow \text{QUICKSORT}(L)\)  
6. \(R' \leftarrow \text{QUICKSORT}(R)\)  
7. return \(L' \circ \{p\} \circ R'\)

**Assumption:** We know how to find the median in \(\Theta(n)\) time.

**Running time:** \(T(n) = 2T(n/2) + \Theta(n) = \Theta(n \lg n)\)
What is the median?

$n$ elements — the median is the $\left\lfloor \frac{n}{2} \right\rfloor$ or $(\lceil \frac{n}{2} \rceil)$ statistic

$O(\log n)$ \quad n \cdot \frac{n}{2} = n^3$
We can't find a pivot by sorting!

QuickSort (A, start, end)

Merge-Sort (A, start, end)

p \in \text{median} (A)

split into L, p, R

QuickSort (L)

|| |

return L + p + R
Start by partitioning $A$ around the median:
Start by partitioning $A$ around the median:

If $k = |L| + 1$, then $p = x_k$. 

\[
L \quad \boxed{p} \quad R
\]
Start by partitioning $A$ around the median:

If $k = |L| + 1$, then $p = x_k$.

Return $p$
Start by partitioning $A$ around the median:

- If $k = |L| + 1$, then $p = x_k$. 
  
  Return $p$

- If $k < |L| + 1$, then $x_k \in L$ and $y > x_k$, for all $y \in R \cup \{p\}$. 
Start by partitioning $A$ around the median:

\[ L \quad p \quad R \]

- If $k = |L| + 1$, then $p = x_k$.

  Return $p$

- If $k < |L| + 1$, then $x_k \in L$ and $y > x_k$, for all $y \in R \cup \{p\}$.

  Recursively find the $k$-th order statistic in $L$ and return it.
Start by partitioning $A$ around the median:

$$L\quad p\quad R$$

- If $k = |L| + 1$, then $p = x_k$.
  
  Return $p$

- If $k < |L| + 1$, then $x_k \in L$ and $y > x_k$, for all $y \in R \cup \{p\}$.
  
  Recursively find the $k$-th order statistic in $L$ and return it.

- If $k > |L| + 1$, then $x_k \in R$ and $y \leq x_k$, for all $y \in L \cup \{p\}$. 
Start by partitioning $A$ around the median:

<table>
<thead>
<tr>
<th></th>
<th>L</th>
<th>$p$</th>
<th>R</th>
</tr>
</thead>
</table>

- If $k = |L| + 1$, then $p = x_k$.  
  
  Return $p$

- If $k < |L| + 1$, then $x_k \in L$ and $y > x_k$, for all $y \in R \cup \{p\}$.  
  
  Recursively find the $k$-th order statistic in $L$ and return it.

- If $k > |L| + 1$, then $x_k \in R$ and $y \leq x_k$, for all $y \in L \cup \{p\}$.  
  
  Recursively find the $(k - |L| - 1)$-st order statistic in $R$ and return it.
Start by partitioning $A$ around the median:

\[
\begin{array}{ccc}
L & p & R \\
\end{array}
\]

- If $k = |L| + 1$, then $p = x_k$.
  
  Return $p$

- If $k < |L| + 1$, then $x_k \in L$ and $y > x_k$, for all $y \in R \cup \{p\}$.
  
  Recursively find the $k$-th order statistic in $L$ and return it.

- If $k > |L| + 1$, then $x_k \in R$ and $y \leq x_k$, for all $y \in L \cup \{p\}$.
  
  Recursively find the $(k - |L| - 1)$-st order statistic in $R$ and return it.

**Running time:** $T(n) \leq$
Start by partitioning $A$ around the median:

\[ L \quad p \quad R \]

- If $k = |L| + 1$, then $p = x_k$. 
  
  Return $p$

- If $k < |L| + 1$, then $x_k \in L$ and $y > x_k$, for all $y \in R \cup \{p\}$. 
  Recursively find the $k$-th order statistic in $L$ and return it.

- If $k > |L| + 1$, then $x_k \in R$ and $y \leq x_k$, for all $y \in L \cup \{p\}$. 
  Recursively find the $(k - |L| - 1)$-st order statistic in $R$ and return it.

**Running time:** $T(n) \leq T(n/2) + O(n)$
Start by partitioning $A$ around the median:

\[ L \quad p \quad R \]

- If $k = |L| + 1$, then $p = x_k$.
  
  Return $p$

- If $k < |L| + 1$, then $x_k \in L$ and $y > x_k$, for all $y \in R \cup \{p\}$.
  
  Recursively find the $k$-th order statistic in $L$ and return it.

- If $k > |L| + 1$, then $x_k \in R$ and $y \leq x_k$, for all $y \in L \cup \{p\}$.
  
  Recursively find the $(k - |L| - 1)$-st order statistic in $R$ and return it.

**Running time:** $T(n) \leq T(n/2) + O(n) = O(n)$
3 7 1 8 6 2 9
3 1 2 5 7 6 9
L P N
4th - order

|L|+1| - order statistic

find 2nd order statistic of L

find 6th order statistic of N

7 6 9
6 7 9
L P N
**Problem:** Finding the median of $A$ is selection.
Problem: Finding the median of $A$ is selection.

Have we walked in a circle?
Relaxing the Partition

**Problem:** Finding the median of $A$ is selection.

Have we walked in a circle?

**Observation:** An “approximate” median does the job:

If $|L| \leq cn$ and $|R| \leq cn$, for some $c < 1$, then

$$T(n) \leq T(cn) + \mathcal{O}(n) = \mathcal{O}(n).$$
Finding an Approximate Median

- Partition input into groups of 5 elements.
- Sort each group and add its 3rd element to an array $A'$.
- Find the median of $A'$ (by calling the selection algorithm recursively!) and return as approximate median.
\[
\begin{array}{cccc}
7 & 3 & 2 & 5 \\
6 & 8 & 9 & 6 7 \\
6 6 7 8 9 & 1 1 2 3 4 \\
\hline
3 & 7 & 2 \\
\hline
1 \\
\hline
\text{selection} \\
3 \\
\hline
2 & 1 1 2 3 & 3 & 7 & 5 & 6 8 9 6 7 4 \\
\hline
6 & & 8
\end{array}
\]
\[ n = 15 \]

\[
7 \quad 3 \quad 2 \quad 5 \quad 1 \quad 6 \quad 8 \quad 9 \quad 6 \quad 7 \quad 1 \quad 1 \quad 2 \quad 3 \quad 4
\]

\[
1 \quad 2 \quad 3 \quad 5 \quad 7 \quad 6 \quad 6 \quad 7 \quad 8 \quad 9 \quad 1 \quad 1 \quad 2 \quad 3 \quad 4
\]

- 2

- 2

- 2

- 2

789 must be larger than 3

57 must be larger than 3

Selection

\[
3 \quad 7 \quad 2
\]

\[
\frac{n}{10} \text{  medians } \geq \text{ the Med}_M
\]

\[
\frac{3n}{10} \text{ elements } \geq \text{ the Med}_M
\]

\[
2 \quad 1 \quad 1 \quad 2 \quad 3 \quad 3 \quad 7 \quad 5 \quad 6 \quad 8 \quad 9 \quad 6 \quad 7 \quad 4
\]

\[
\frac{\text{median of medians}}{6} \quad 8
\]
original array split into groups of 5 and then each group is sorted

\[ a_1, a_2, m_1, a_3, a_4, a_5, q_6, m_2, q_7, q_8, a_9, q_{10}, m_3, q_{11}, q_{12} \]

median array: \[ m_1, m_2, m_3 \]

\( n = 15 \)

\( \frac{n}{5} = 3 \) medians

\( \geq \frac{n}{10} \) medians larger than the MoFM

\[ \sqrt{\text{selection to find median of medians}} = m_2 \]

\[ m_2, a_7, a_8 \geq m_2 \]

\[ m_3, q_{11}, q_{12} \geq m_2 \]

3 from each of the \( \frac{n}{10} \) groups are \( \geq \) MoFM

\[ \frac{3n}{10} \geq 3 \times \text{MoFM} \]
The Procedure Finds an Approximate Median

Lemma: There are at least \( \frac{3n}{10} - 6 \) elements on either side of the computed approximate median \( p \).

\[ \begin{align*}
11 & \geq \frac{3n}{10} - 6 \\
\text{Most uneven split} & \Rightarrow \\
\text{Selection recurrence} & \\
T(n) & = T\left(\frac{2n}{10} + 6\right) + n + T\left(\frac{n}{5}\right)
\end{align*} \]
The Procedure Finds an Approximate Median

Lemma: There are at least $\frac{3n}{10} - 6$ elements on either side of the computed approximate median $p$.

Proof: (for elements greater than $p$)

- At least $\left\lceil \frac{\lceil n/5 \rceil}{2} \right\rceil - 1 \geq \frac{n}{10} - 1$ groups to the right of $p$
- At most one is not full
- Every full group contains at least 3 elements $> p$

Total: $3 \left( \frac{n}{10} - 2 \right) = \frac{3n}{10} - 6$
\[
\frac{1}{10} \text{ medians}
\]
\[
- \frac{m_3 - m_4}{\text{median of medians}}
\]
\[
\text{could be smaller}
\]
\[
\frac{3}{10} \text{ elements must be } \geq \text{ the median of medians}
\]
The Final Running Time

Summary of selection algorithm:
■ Find approximate median: linear work + recurse on $\lceil n/5 \rceil$ elements $T(n/5)$
■ Partition: linear work $\Theta(n)$
■ Recurse on piece of size at most $7n/10 + 6$ $T(\frac{7}{10}n + 6)$
The Final Running Time

Summary of selection algorithm:
- Find approximate median: linear work + recurse on $\lceil n/5 \rceil$ elements
- Partition: linear work
- Recurse on piece of size at most $7n/10 + 6$

Recurrence:

$$T(n) = \begin{cases} 
O(1) & \text{if } n \leq 140 \\
O(n) + T(\lceil n/5 \rceil) + T(\frac{7n}{10} + 6) & \text{if } n > 140
\end{cases}$$
The Final Running Time

Summary of selection algorithm:

- Find approximate median: linear work + recurse on \( \lceil n/5 \rceil \) elements
- Partition: linear work
- Recurse on piece of size at most \( 7n/10 + 6 \)

Recurrence:

\[
T(n) = \begin{cases} 
\mathcal{O}(1) & \text{if } n \leq 140 \\
\mathcal{O}(n) + T(\lceil n/5 \rceil) + T(\frac{7n}{10} + 6) & \text{if } n > 140 
\end{cases}
\]

\[
= \mathcal{O}(n)
\]

Substitution

\[
T(n) \leq \mathcal{O}(n) + 7\left(\frac{n}{5}\right) + 7\left(\frac{7n}{10} + 6\right)
\]

\[
\leq \mathcal{O}(n) + c\frac{n}{5} + c\left(\frac{7n}{10} + 6\right)
\]

\[
\leq \mathcal{O}(n) + c\left(\frac{9n}{10} + 6\right)
\]

\[
\leq \mathcal{O}(n)
\]
Summary of selection algorithm:
- Find approximate median: linear work + recurse on \([n/5]\) elements
- Partition: linear work
- Recurse on piece of size at most \(7n/10 + 6\)

Recurrence:

\[
T(n) = \begin{cases} 
O(1) & \text{if } n \leq 140 \\
O(n) + T(\lceil n/5 \rceil) + T(7n/10 + 6) & \text{if } n > 140 
\end{cases}
\]

= \(O(n)\)

Theorem: The \(k\)-th order statistic of a set of \(n\) elements can be found in \(O(n)\) time.
Counting Inversions

Given a sequence \( S = (x_1, x_2, \ldots, x_n) \) of \( n \) numbers, an inversion is a pair \((x_i, x_j)\) such that
- \( i < j \) and
- \( x_i > x_j \).

**Example:**

\[
\begin{array}{cccccccc}
5 & 3 & 7 & 8 & 21 & 10 & 17 & 16 \\
\end{array}
\]

Inversions: \((5, 3), (21, 10), (21, 17), (21, 16), (17, 16)\)
Counting Inversions

Given a sequence $S = (x_1, x_2, \ldots, x_n)$ of $n$ numbers, an inversion is a pair $(x_i, x_j)$ such that
- $i < j$ and
- $x_i > x_j$.

Example:

\begin{center}
\begin{tabular}{cccccccc}
5 & 3 & 7 & 8 & 21 & 10 & 17 & 16 \\
\end{tabular}
\end{center}

Inversions: $(5, 3), (21, 10), (21, 17), (21, 16), (17, 16)$

Problem: Count all inversions in $S$. 

$$\sum_{k=1}^{n} \sum_{j=k+1}^{n} = \frac{n-1}{2} \cdot \frac{n+1}{2} \cdot \frac{n}{2} \cdot \frac{n+1}{2} = \Theta(n^2)$$
Classifying Inversions

As in Merge Sort, partition array into left half, $L$, and right half, $R$:

- An inversion $(x_i, x_j)$ is **short** if $\{x_i, x_j\} \subseteq L$ or $\{x_i, x_j\} \subseteq R$
- An inversion $(x_i, x_j)$ is **long** if $x_i \in L$ and $x_j \in R
Classifying Inversions

As in Merge Sort, partition array into left half, $L$, and right half, $R$:

- An inversion $(x_i, x_j)$ is **short** if $\{x_i, x_j\} \subseteq L$ or $\{x_i, x_j\} \subseteq R$

- An inversion $(x_i, x_j)$ is **long** if $x_i \in L$ and $x_j \in R$

Since we are talking about divide and conquer:
- Find short recursions recursively.
Counting Long Inversions

Observation: Sorting $L$ and $R$ does not affect the number of long inversions.
Counting Long Inversions

**Observation:** Sorting $L$ and $R$ does not affect the number of long inversions.

**Procedure:**
- Sort $L$ and $R$.
- Count long inversions by merging $L$ and $R$: 

![Diagram showing the merging of $L$ and $R$ with a focus on counting long inversions.](image)
**Observation:** Sorting \( L \) and \( R \) does not affect the number of long inversions.

**Procedure:**
- Sort \( L \) and \( R \).
- Count long inversions by merging \( L \) and \( R \):
  - When \( y < x \), then \( y \) forms an inversion with exactly the elements remaining in \( L \).
Observation: Sorting $L$ and $R$ does not affect the number of long inversions.

**Procedure:**
- Sort $L$ and $R$.
- Count long inversions by merging $L$ and $R$:
  - When $y < x$, then $y$ forms an inversion with exactly the elements remaining in $L$.
  - Increase inversion count by $|L|$.
Analysis

\[ T(n) = \]
\[ T(n) = 2T(n/2) + \Theta(n \log n) \]

\[ n \log n = \Theta(n) \]

\[ \text{case 3} \quad n \log n = \Theta(n \log_{5.3} n) \text{ does not work with the master theorem} \]
Analysis

\[ T(n) = 2T(n/2) + \Theta(n \lg n) = \Theta(n \lg^2 n). = \Theta(n^2) \]

\[
\begin{align*}
\text{count}^- \text{-inversions} (A, \text{start}, \text{end}) \\
\text{count}^- \text{-inversions} (A, \text{start}, \text{middle}) \\
\text{count}^- \text{-inversions} (A, \text{middle}+1, \text{end}) \\
\text{merge}^- \text{sort}^+ (A, \text{start}, \text{middle}) \\
\text{merge}^- \text{sort}^+ 
\end{align*}
\]
Analysis

\[ T(n) = 2T(n/2) + \Theta(n \lg n) = \Theta(n \lg^2 n). \]

**Observation:**
- Counting long inversions produces \( L \cup R \) in sorted order, as a by-product.
\[ T(n) = 2T(n/2) + \Theta(n \lg n) = \Theta(n \lg^2 n). \]

**Observation:**

- Counting long inversions produces \( L \cup R \) in sorted order, as a by-product.
- In particular, the recursive calls on \( L \) and \( R \) return \( L \) and \( R \) in sorted order.
Analysis

\[ T(n) = 2T(n/2) + \Theta(n \lg n) = \Theta(n \lg^2 n). \]

**Observation:**
- Counting long inversions produces \( L \cup R \) in sorted order, as a by-product.
- In particular, the recursive calls on \( L \) and \( R \) return \( L \) and \( R \) in sorted order.

∴ We can save the sorting step.
Analysis

\[ T(n) = 2 T(n/2) + \Theta(n \lg n) = \Theta(n \lg^2 n). \]

Observation:
- Counting long inversions produces \( L \cup R \) in sorted order, as a by-product.
- In particular, the recursive calls on \( L \) and \( R \) return \( L \) and \( R \) in sorted order.

\[ \therefore \text{We can save the sorting step.} \]

\[ \therefore T(n) = \]
Analysis

\[ T(n) = 2T(n/2) + \Theta(n \log n) = \Theta(n \log^2 n). \]

**Observation:**

- Counting long inversions produces \( L \cup R \) in sorted order, as a by-product.
- In particular, the recursive calls on \( L \) and \( R \) return \( L \) and \( R \) in sorted order.

\[ \therefore T(n) = 2T(n/2) + \Theta(n) \]
Analysis

\[ T(n) = 2T(n/2) + \Theta(n \lg n) = \Theta(n \lg^2 n). \]

**Observation:**
- Counting long inversions produces \( L \cup R \) in sorted order, as a by-product.
- In particular, the recursive calls on \( L \) and \( R \) return \( L \) and \( R \) in sorted order.
∴ We can save the sorting step.

∴ \[ T(n) = 2T(n/2) + \Theta(n) = \Theta(n \lg n). \equiv O(n^2) \]
1 + 2 = 3
+ 1 = 4
+ 1 + 1 = 6

1 + 2 = 3
+ 1 = 4
+ 3 = 7
+ 2 = 9

1 + 5 = 6
+ 4 = 10
+ 4 = 14

dotil: 27
Multiplying Large Integers

**Problem:** Given two $n$-digit numbers $x = x_{n-1}x_{n-2}\ldots x_0$ and $y = y_{n-1}y_{n-2}\ldots y_0$, we want to compute $z = x \cdot y$ using only digit-wise operations.

- Multiply two 64-bit numbers? $O(1)$
- Multiply two 128-bit numbers?

```
  16  2
\times 11 3

\underline{10}
\underline{106}
\underline{1106}
```
**Problem:** Given two $n$-digit numbers $x = x_{n-1}x_{n-1} \ldots x_0$ and $y = y_{n-1}y_{n-2} \ldots y_0$, we want to compute $z = x \cdot y$ using only digit-wise operations.

**The traditional method:**

\[
\begin{array}{c}
54163 \times 63021 \\
324978 \quad O(n) \\
162489 \quad O(n) \\
0 \quad O(n) \\
108326 \quad O(n) \\
54163 \quad O(n) \\
\hline
3413406423 \quad O(n^3)
\end{array}
\]
Multiplying Large Integers

Problem: Given two $n$-digit numbers $x = x_{n-1}x_{n-2}\cdots x_0$ and $y = y_{n-1}y_{n-2}\cdots y_0$, we want to compute $z = x \cdot y$ using only digit-wise operations.

The traditional method:

\[
\begin{array}{c}
54163 \times 63021 \\
\hline
324978 \\
162489 \\
\hline
0 \\
108326 \\
\hline
54163 \\
\hline
3413406423
\end{array}
\]

Cost:
Multiplying Large Integers

Problem: Given two \( n \)-digit numbers \( x = x_{n-1}x_{n-1} \ldots x_0 \) and
\( y = y_{n-1}y_{n-2} \ldots y_0 \), we want to compute \( z = x \cdot y \) using only digit-wise
operations.

The traditional method:

\[
\begin{array}{cccc}
54163 & \times & 63021 \\
\hline
324978 \\
162489 \\
0 \\
108326 \\
54163 \\
\hline
3413406423
\end{array}
\]

Cost: \( \Theta(n^2) \)
Divide-and-Conquer Multiplication

Assumption: $n = 2^k$

A recursive method:

\[
x' \quad x'' \\
\hline
y' \quad y''
\]

\[
x'' \cdot y'' \\
\hline
x'' \cdot y' + x' \cdot y'' \\
\hline
x' \cdot y'
\]

\[
\hline
x \cdot y
\]

\[
T(n) = 4 \cdot T\left(\frac{n}{3}\right) + \Theta(n)
\]
**Assumption:** \( n = 2^k \)

**A recursive method:**

\[
\begin{align*}
& x' \quad x'' \\
& \hline \\
& y' \quad y'' \\
& \hline \\
& x'' \cdot y'' \\
& \hline \\
& x'' \cdot y' + x' \cdot y'' \\
& \hline \\
& x' \cdot y' \\
& \hline \\
& x \cdot y
\end{align*}
\]

**Recurrence:** \( T(n) = \)
Divide-and-Conquer Multiplication

Assumption: \( n = 2^k \)

A recursive method:

\[
\begin{align*}
  x' & \quad x'' \\
  y' & \quad y'' \\
  x'' \cdot y'' \\
  x'' \cdot y' + x' \cdot y'' \\
  x' \cdot y' \\
  x \cdot y
\end{align*}
\]

Recurrence: \( T(n) = 4T(n/2) + \Theta(n) \)
Divide-and-Conquer Multiplication

Assumption: \( n = 2^k \)

A recursive method:

\[
\begin{align*}
(x' \cdot y') &+ (x'' \cdot y'') \\
(x'' \cdot y') &+ (x' \cdot y'') \\
(x'' \cdot y'') &
\end{align*}
\]

Recurrence: \( T(n) = 4T(n/2) + \Theta(n) = \Theta(n^2) \)

Bummer!
One Less Recursive Call

Compute recursively:

- \( A = x' \cdot y' \)
- \( B = x'' \cdot y'' \)
- \( C = (x' + x'') \cdot (y' + y'') \)
Compute recursively:

- $A = x' \cdot y'$
- $B = x'' \cdot y''$
- $C = (x' + x'') \cdot (y' + y'')$

Combine results:
Compute recursively:

- $A = x' \cdot y'$
- $B = x'' \cdot y''$
- $C = (x' + x'') \cdot (y' + y'')$

Combine results:

- $x' \cdot y' = A$
- $x'' \cdot y'' = B$
One Less Recursive Call

Compute recursively:

- \( A = x' \cdot y' \)
- \( B = x'' \cdot y'' \)
- \( C = (x' + x'') \cdot (y' + y'') \)

Combine results:

- \( x' \cdot y' = A \)
- \( x'' \cdot y'' = B \)
- \( x' \cdot y'' + x'' \cdot y' = C - A - B \)
- \( (x' + x') (y' + y'') \sim x' y' - x'' y'' \)
- \( x y' + x y'' + x y' + x y'' \sim x y' - x y'' \)
Compute recursively:

- \( A = x' \cdot y' \)
- \( B = x'' \cdot y'' \)
- \( C = (x' + x'') \cdot (y' + y'') \)

Combine results:

- \( x' \cdot y' = A \)
- \( x'' \cdot y'' = B \)
- \( x' \cdot y'' + x'' \cdot y' = C - A - B \)

Recurrence: \( T(n) = \)
One Less Recursive Call

Compute recursively:

- $A = x' \cdot y'$
- $B = x'' \cdot y''$
- $C = (x' + x'') \cdot (y' + y'')$

Combine results:

- $x' \cdot y' = A$
- $x'' \cdot y'' = B$
- $x' \cdot y'' + x'' \cdot y' = C - A - B$

Recurrence: $T(n) = 3T(n/2) + \Theta(n)$

$$n^{\log_{b}a} = n^{\log_{2}3}$$

$$f(n) = n^{\log_{b}a - 3} = n^{\log_{b}3 - (1 - \log_{b}3)} = n^{\log_{2}3 - (1 - \log_{2}3)} = 1$$
Compute recursively:

- $A = x' \cdot y'$
- $B = x'' \cdot y''$
- $C = (x' + x'') \cdot (y' + y'')$

Combine results:

- $x' \cdot y' = A$
- $x'' \cdot y'' = B$
- $x' \cdot y'' + x'' \cdot y' = C - A - B$

Recurrence: $T(n) = 3T(n/2) + \Theta(n) = \Theta(n^{1+\lg(3/2)}) = n^{\lg_2 3} = \Theta(n^3)$
One Less Recursive Call

Compute recursively:

- \(A = x' \cdot y'\)
- \(B = x'' \cdot y''\)
- \(C = (x' + x'') \cdot (y' + y'')\)

Combine results:

- \(x' \cdot y' = A\)
- \(x'' \cdot y'' = B\)
- \(x' \cdot y'' + x'' \cdot y' = C - A - B\)

Recurrence: \(T(n) = 3T(n/2) + \Theta(n) = \Theta(n^{1+\log(3/2)}) \approx \Theta(n^{1.58})\)
One Less Recursive Call

Compute recursively:

- $A = x' \cdot y'$
- $B = x'' \cdot y''$
- $C = (x' + x'') \cdot (y' + y'')$

Combine results:

- $x' \cdot y' = A$
- $x'' \cdot y'' = B$
- $x' \cdot y'' + x'' \cdot y' = C - A - B$

Recurrence: $T(n) = 3T(n/2) + \Theta(n) = \Theta(n^{1+\lg(3/2)}) \approx \Theta(n^{1.58})$

Note: This works only because addition has an inverse operation; that is, it does not work over a semi-ring.
Summary

**Divide and conquer:**

- *Divide* the problem instance into two or more smaller instances of the same problem.
- Recursively solve (*conquer*) these smaller problem instances.
- *Combine* the solutions obtained for the smaller instances to construct a solution for the original problem instance.

*Divide-and-conquer algorithms are by definition recursive.*

- Natural expression of running time using recurrence relations
- Natural correctness proofs using induction

**Solving recurrence relations:**

- Substitution
- Recursion trees
- Master theorem
trying to find out something is $\Theta(n \log n)$

first prove $O(n^2)$

second prove $\Omega(n)$
\[ n \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ \vdots & \vdots & \vdots & \vdots \\ y_1 & y_2 & y_3 & y_4 \end{bmatrix} n \]

\[
\begin{bmatrix}
\approx x_1 y_1 + x_2 y_2 + x_3 y_3 + x_4 y_4 \\
\end{bmatrix}
\]

\[
\Theta (n^3)
\]

naive divide and conquer

\[
T(n) = 8 T \left( \frac{n}{8} \right) + \Theta(n^2) = \Theta(n^3)
\]