Algorithms and Data Structures for Mathematicians

Lecture 2: Divide and Conquer

Peter Kostolányi
kostolanyi at fmph and so on
Room M-258

5 October 2017

Recapitulation

Some topics addressed in the first lecture:

Recapitulation

Some topics addressed in the first lecture:

- Algorithms and their description in pseudocode

Recapitulation

Some topics addressed in the first lecture:

- Algorithms and their description in pseudocode
- Worst-case time complexity analysis

Recapitulation

Some topics addressed in the first lecture:

- Algorithms and their description in pseudocode
- Worst-case time complexity analysis
- Asymptotic notation

Recapitulation

Some topics addressed in the first lecture:

- Algorithms and their description in pseudocode
- Worst-case time complexity analysis
- Asymptotic notation
- Sorting in $\Theta(n^2)$ worst-case (Insertion sort)

Recapitulation

Some topics addressed in the first lecture:

- Algorithms and their description in pseudocode
- Worst-case time complexity analysis
- Asymptotic notation
- Sorting in $\Theta(n^2)$ worst-case (Insertion sort)

Questions that we have not dealt with so far:

Recapitulation

Some topics addressed in the first lecture:

- Algorithms and their description in pseudocode
- Worst-case time complexity analysis
- Asymptotic notation
- Sorting in $\Theta(n^2)$ worst-case (Insertion sort)

Questions that we have not dealt with so far:

- Are there any techniques for designing efficient algorithms?

Recapitulation

Some topics addressed in the first lecture:

- Algorithms and their description in pseudocode
- Worst-case time complexity analysis
- Asymptotic notation
- Sorting in $\Theta(n^2)$ worst-case (Insertion sort)

Questions that we have not dealt with so far:

- Are there any techniques for designing efficient algorithms?
- Can we do better than $\Theta(n^2)$ for sorting (in worst-case)?

Divide and Conquer

A paradigm for algorithm design:

Divide and Conquer

A paradigm for algorithm design:

Divide the problem into several **smaller** instances of the same problem

Divide and Conquer

A paradigm for algorithm design:

- **Divide** the problem into several smaller instances of the same problem
- **Conquer** the subproblems by solving them recursively or directly if they are small enough

Divide and Conquer

A paradigm for algorithm design:

- **Divide** the problem into several **smaller** instances of the same problem
- **Conquer** the subproblems by solving them recursively or directly if they are small enough
- **Combine** the solutions of the subproblems to obtain a solution of the original problem

Merge Sort

- Let $(S, \preceq)$ be a totally ordered set

Merge Sort

- Let $(S, \preceq)$ be a totally ordered set
- Given an array of n elements of S, we wish to sort them in increasing order

Merge Sort

- Let $(S, \preceq)$ be a totally ordered set
- Given an array of n elements of S, we wish to sort them in increasing order

The Divide and Conquer Approach:

Merge Sort

- Let $(S, \preceq)$ be a totally ordered set
- Given an array of n elements of S, we wish to sort them in increasing order

The Divide and Conquer Approach:

Divide the array into two (approximate) halves

Merge Sort

- Let $(S, \preceq)$ be a totally ordered set
- Given an array of n elements of S, we wish to sort them in increasing order

The Divide and Conquer Approach:

Divide the array into two (approximate) halves

Conquer the subproblems by sorting the respective subarrays

Merge Sort

- Let $(S, \preceq)$ be a totally ordered set
- Given an array of n elements of S, we wish to sort them in increasing order

The Divide and Conquer Approach:

Divide the array into two (approximate) halves

Conquer the subproblems by sorting the respective subarrays

- If a subarray contains only one element, it is already sorted

Merge Sort

- Let $(S, \preceq)$ be a totally ordered set
- Given an array of n elements of S, we wish to sort them in increasing order

The Divide and Conquer Approach:

Divide the array into two (approximate) halves

Conquer the subproblems by sorting the respective subarrays

- If a subarray contains only one element, it is already sorted
- Otherwise divide once again (recursion)

Merge Sort

- Let $(S, \preceq)$ be a totally ordered set
- Given an array of n elements of S, we wish to sort them in increasing order

The Divide and Conquer Approach:

Divide the array into two (approximate) halves

Conquer the subproblems by sorting the respective subarrays

- If a subarray contains only one element, it is already sorted
- Otherwise divide once again (recursion)

Combine the two sorted subarrays by merging them into a single sorted array

Merge Procedure

- Let $(S, \preceq)$ be a totally ordered set

Merge Procedure

- Let $(S, \preceq)$ be a totally ordered set
- Let $\top \succ x$ for all x in S

Merge Procedure

- Let $(S, \preceq)$ be a totally ordered set
- Let $\top \succ x$ for all x in S

$\mathrm{MERGE}(a, \mathsf{f}, \mathsf{m}, \mathsf{l})$:

Merge Procedure

- Let $(S, \preceq)$ be a totally ordered set
- Let $\top \succ x$ for all x in S

MERGE(a, f, m, l):

Input : Integer $n \geq 1$; Array $a = \langle a[1], \ldots, a[n] \rangle$ of elements of S; Integers f, m, l such that $1 \leq \mathrm{f} \leq \mathrm{m} < \mathrm{l} \leq n$ and such that $\langle a[\mathrm{f}], \ldots, a[\mathrm{m}] \rangle$ and $\langle a[\mathrm{m}+1], \ldots, a[\mathrm{l}] \rangle$ are already sorted

Behaviour: Merges $\langle a[\mathrm{f}], \ldots, a[\mathrm{m}] \rangle$ and $\langle a[\mathrm{m}+1], \ldots, a[\mathrm{l}] \rangle$ into a sorted subarray

$i \leftarrow \mathrm{f}$; $j \leftarrow \mathrm{m}+1$; $k \leftarrow 0$;
while $i \leq \mathrm{m}$ *or* $j \leq \mathrm{l}$ **do**
    $k \leftarrow k+1$;
    if $i \leq \mathrm{m}$ **then** $r \leftarrow a[i]$ **else** $r \leftarrow \top$;
    if $j \leq \mathrm{l}$ **then** $s \leftarrow a[j]$ **else** $s \leftarrow \top$;
    if $r \preceq s$ **then** $c[k] \leftarrow r$; $i \leftarrow i+1$ **else** $c[k] \leftarrow s$; $j \leftarrow j+1$;
end
for $i \leftarrow \mathrm{f}$ **to** l **do** $a[i] \leftarrow c[i-\mathrm{f}+1]$;

Merge Procedure

- Let $(S, \preceq)$ be a totally ordered set
- Let $\top \succ x$ for all x in S

MERGE(a, f, m, l):

Input : Integer $n \geq 1$; Array $a = \langle a[1], \ldots, a[n] \rangle$ of elements of S; Integers f, m, l such that $1 \leq \mathrm{f} \leq \mathrm{m} < \mathrm{l} \leq n$ and such that $\langle a[\mathrm{f}], \ldots, a[\mathrm{m}] \rangle$ and $\langle a[\mathrm{m}+1], \ldots, a[\mathrm{l}] \rangle$ are already sorted

Behaviour: Merges $\langle a[\mathrm{f}], \ldots, a[\mathrm{m}] \rangle$ and $\langle a[\mathrm{m}+1], \ldots, a[\mathrm{l}] \rangle$ into a sorted subarray

```
i ← f; j ← m + 1; k ← 0;
while i ≤ m or j ≤ l do
    k ← k + 1;
    if i ≤ m then r ← a[i] else r ← ⊤;
    if j ≤ l then s ← a[j] else s ← ⊤;
    if r ⪯ s then c[k] ← r; i ← i + 1 else c[k] ← s; j ← j + 1;
end
for i ← f to l do a[i] ← c[i − f + 1];
```

- Time complexity: $\Theta(l - f)$

Merge Sort

MERGESORT(a, f, l):

Merge Sort

$\text{MERGESORT}(a, \mathsf{f}, \mathsf{l})$:

Input : Integer $n \geq 1$; Array $a = \langle a[1], \ldots, a[n] \rangle$ of elements of S;
Integers f, l such that $1 \leq \mathsf{f} \leq \mathsf{l} \leq n$

Behaviour: Sorts the subarray $\langle a[\mathsf{f}], \ldots, a[\mathsf{l}] \rangle$ in increasing order

```
if f = l then
    return
else
    m ← ⌊(f + l)/2⌋;
    MERGESORT(a, f, m);
    MERGESORT(a, m + 1, l);
    MERGE(a, f, m, l);
end
```

Merge Sort

MERGESORT(a, f, l):

Input : Integer $n \geq 1$; Array $a = \langle a[1], \ldots, a[n] \rangle$ of elements of S;
Integers f, l such that $1 \leq \mathrm{f} \leq \mathrm{l} \leq n$

Behaviour: Sorts the subarray $\langle a[\mathrm{f}], \ldots, a[\mathrm{l}] \rangle$ in increasing order

if $\mathrm{f} = \mathrm{l}$ **then**
  return
else
  $\mathrm{m} \leftarrow \lfloor (\mathrm{f} + \mathrm{l})/2 \rfloor$;
  MERGESORT(a, f, m);
  MERGESORT(a, m + 1, l);
  MERGE(a, f, m, l);
end

- Time complexity $T(n)$ of MERGESORT(a, 1, n)?

Merge Sort

$\text{MERGESORT}(a, \mathsf{f}, \mathsf{l})$:

Input : Integer $n \geq 1$; Array $a = \langle a[1], \ldots, a[n] \rangle$ of elements of S;
Integers f, l such that $1 \leq \mathsf{f} \leq \mathsf{l} \leq n$

Behaviour: Sorts the subarray $\langle a[\mathsf{f}], \ldots, a[\mathsf{l}] \rangle$ in increasing order

```
if f = l then
    return
else
    m ← ⌊(f + l)/2⌋;
    MERGESORT(a, f, m);
    MERGESORT(a, m + 1, l);
    MERGE(a, f, m, l);
end
```

- Time complexity $T(n)$ of $\text{MERGESORT}(a, 1, n)$?
- Surely $T(1) = \Theta(1)$

Merge Sort

$\text{MERGESORT}(a, \mathsf{f}, \mathsf{l})$:

Input : Integer $n \geq 1$; Array $a = \langle a[1], \ldots, a[n] \rangle$ of elements of S;
Integers f, l such that $1 \leq \mathsf{f} \leq \mathsf{l} \leq n$

Behaviour: Sorts the subarray $\langle a[\mathsf{f}], \ldots, a[\mathsf{l}] \rangle$ in increasing order

```
if f = l then
    return
else
    m ← ⌊(f + l)/2⌋;
    MERGESORT(a, f, m);
    MERGESORT(a, m + 1, l);
    MERGE(a, f, m, l);
end
```

- Time complexity $T(n)$ of $\text{MERGESORT}(a, 1, n)$?
- Surely $T(1) = \Theta(1)$
- We have $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n)$ for $n \geq 2$

Merge Sort

$\text{MERGESORT}(a, \mathsf{f}, \mathsf{l})$:

Input : Integer $n \geq 1$; Array $a = \langle a[1], \ldots, a[n] \rangle$ of elements of S;
Integers f, l such that $1 \leq \mathsf{f} \leq \mathsf{l} \leq n$

Behaviour: Sorts the subarray $\langle a[\mathsf{f}], \ldots, a[\mathsf{l}] \rangle$ in increasing order

```
if f = l then
    return
else
    m ← ⌊(f + l)/2⌋;
    MERGESORT(a, f, m);
    MERGESORT(a, m + 1, l);
    MERGE(a, f, m, l);
end
```

- Time complexity $T(n)$ of $\text{MERGESORT}(a, 1, n)$?
- Surely $T(1) = \Theta(1)$
- We have $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n)$ for $n \geq 2$
- We need to find a solution to the asymptotic recurrence above

Asymptotic Recurrences

We need a technique for solving recurrences of the form

Asymptotic Recurrences

We need a technique for solving recurrences of the form

$$\begin{aligned} T(n) &= \sum_{i=1}^{a} T(\varphi_i(n/b)) + \Theta(n^d) && \text{for } n \geq b, \\ T(s) &= \Theta(1) && \text{for } s = 1, \ldots, b-1, \end{aligned}$$

where:

Asymptotic Recurrences

We need a technique for solving recurrences of the form

$$\begin{aligned} T(n) &= \sum_{i=1}^{a} T(\varphi_i(n/b)) + \Theta(n^d) && \text{for } n \geq b, \\ T(s) &= \Theta(1) && \text{for } s = 1, \ldots, b-1, \end{aligned}$$

where:

- $a \geq 1$ is in $\mathbb{N}$, $b \geq 2$ is in $\mathbb{N}$, and $d > 0$ is in $\mathbb{R}$

Asymptotic Recurrences

We need a technique for solving recurrences of the form

$$
\begin{aligned}
T(n) &= \sum_{i=1}^{a} T(\varphi_i(n/b)) + \Theta(n^d) && \text{for } n \geq b, \\
T(s) &= \Theta(1) && \text{for } s = 1, \ldots, b-1,
\end{aligned}
$$

where:

- $a \geq 1$ is in $\mathbb{N}$, $b \geq 2$ is in $\mathbb{N}$, and $d > 0$ is in $\mathbb{R}$
- $\varphi_i(x) = \lfloor x \rfloor$ or $\varphi_i(x) = \lceil x \rceil$ for $i = 1, \ldots, a$ and all x in $\mathbb{R}$

Asymptotic Recurrences

We need a technique for solving recurrences of the form

$$
\begin{aligned}
T(n) &= \sum_{i=1}^{a} T(\varphi_i(n/b)) + \Theta(n^d) && \text{for } n \geq b, \\
T(s) &= \Theta(1) && \text{for } s = 1, \ldots, b-1,
\end{aligned}
$$

where:

- $a \geq 1$ is in $\mathbb{N}$, $b \geq 2$ is in $\mathbb{N}$, and $d > 0$ is in $\mathbb{R}$
- $\varphi_i(x) = \lfloor x \rfloor$ or $\varphi_i(x) = \lceil x \rceil$ for $i = 1, \ldots, a$ and all x in $\mathbb{R}$

It is possible to prove that the (asymptotic) solution does not depend on the particular rounding functions φ_i

Asymptotic Recurrences

We need a technique for solving recurrences of the form

$$\begin{aligned} T(n) &= \sum_{i=1}^{a} T(\varphi_i(n/b)) + \Theta(n^d) && \text{for } n \geq b, \\ T(s) &= \Theta(1) && \text{for } s = 1, \ldots, b-1, \end{aligned}$$

where:

- $a \geq 1$ is in $\mathbb{N}$, $b \geq 2$ is in $\mathbb{N}$, and $d > 0$ is in $\mathbb{R}$
- $\varphi_i(x) = \lfloor x \rfloor$ or $\varphi_i(x) = \lceil x \rceil$ for $i = 1, \ldots, a$ and all x in $\mathbb{R}$

It is possible to prove that the (asymptotic) solution does not depend on the particular rounding functions φ_i

We shall thus simply write:

$$T(n) = aT(n/b) + \Theta(n^d)$$

Asymptotic Recurrences

We need a technique for solving recurrences of the form

$$
\begin{aligned}
T(n) &= \sum_{i=1}^{a} T(\varphi_i(n/b)) + \Theta(n^d) && \text{for } n \geq b, \\
T(s) &= \Theta(1) && \text{for } s = 1, \ldots, b-1,
\end{aligned}
$$

where:

- $a \geq 1$ is in $\mathbb{N}$, $b \geq 2$ is in $\mathbb{N}$, and $d > 0$ is in $\mathbb{R}$
- $\varphi_i(x) = \lfloor x \rfloor$ or $\varphi_i(x) = \lceil x \rceil$ for $i = 1, \ldots, a$ and all x in $\mathbb{R}$

It is possible to prove that the (asymptotic) solution does not depend on the particular rounding functions φ_i

We shall thus simply write:

$$T(n) = aT(n/b) + \Theta(n^d)$$

- We shall find a solution assuming that $n = b^k$, where k is in $\mathbb{N}$

Asymptotic Recurrences

We need a technique for solving recurrences of the form

$$\begin{aligned} T(n) &= \sum_{i=1}^{a} T(\varphi_i(n/b)) + \Theta(n^d) && \text{for } n \geq b, \\ T(s) &= \Theta(1) && \text{for } s = 1, \ldots, b-1, \end{aligned}$$

where:

- $a \geq 1$ is in $\mathbb{N}$, $b \geq 2$ is in $\mathbb{N}$, and $d > 0$ is in $\mathbb{R}$
- $\varphi_i(x) = \lfloor x \rfloor$ or $\varphi_i(x) = \lceil x \rceil$ for $i = 1, \ldots, a$ and all x in $\mathbb{R}$

It is possible to prove that the (asymptotic) solution does not depend on the particular rounding functions φ_i

We shall thus simply write:

$$T(n) = aT(n/b) + \Theta(n^d)$$

- We shall find a solution assuming that $n = b^k$, where k is in $\mathbb{N}$
- It is possible to extend the analysis to deal with non-exact-powers of b as well

Asymptotic Recurrences

Theorem

Let $T(n)$ be a function satisfying

$$T(n) \leq aT(n/b) + cn^d \qquad \text{for } n = b^k,\ k = 1, 2, 3, \ldots,$$
$$T(1) = \Theta(1),$$

where $a \geq 1$ and $b \geq 2$ are in $\mathbb{N}$, and $c, d > 0$ are in $\mathbb{R}$. Then

$$T(n) = \left\{ \begin{array}{ll} O\left(n^{\log_b a}\right) & \text{if } d < \log_b a \\ O\left(n^d \log n\right) & \text{if } d = \log_b a \\ O\left(n^d\right) & \text{if } d > \log_b a \end{array} \right\} \text{ for } n = b^k,\ k = 1, 2, 3, \ldots$$

Asymptotic Recurrences

Theorem

Let $T(n)$ be a function satisfying

$$T(n) \geq aT(n/b) + cn^d \qquad \text{for } n = b^k,\ k = 1, 2, 3, \ldots,$$
$$T(1) = \Theta(1),$$

where $a \geq 1$ and $b \geq 2$ are in $\mathbb{N}$, and $c, d > 0$ are in $\mathbb{R}$. Then

$$T(n) = \left\{ \begin{array}{ll} \Omega\left(n^{\log_b a}\right) & \text{if } d < \log_b a \\ \Omega\left(n^d \log n\right) & \text{if } d = \log_b a \\ \Omega\left(n^d\right) & \text{if } d > \log_b a \end{array} \right\} \text{ for } n = b^k,\ k = 1, 2, 3, \ldots$$

Asymptotic Recurrences

Corollary

Let $T(n)$ be a function satisfying

$$\begin{aligned} &T(n) = aT(n/b) + \Theta(n^d) \qquad \textit{for } n = b^k,\ k = 1, 2, 3, \ldots, \\ &T(1) = \Theta(1), \end{aligned}$$

where $a \geq 1$ and $b \geq 2$ are in $\mathbb{N}$, and $d > 0$ is in $\mathbb{R}$. Then

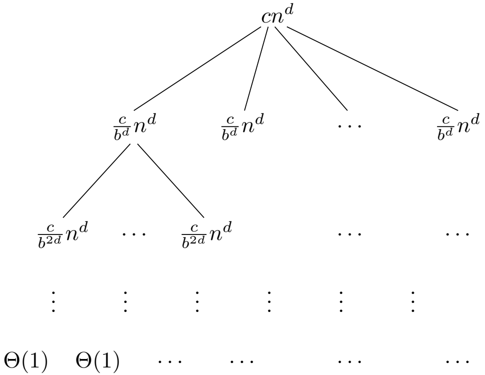
$$T(n) = \left\{ \begin{array}{ll} \Theta\left(n^{\log_b a}\right) & \textit{if } d < \log_b a \\ \Theta\left(n^d \log n\right) & \textit{if } d = \log_b a \\ \Theta\left(n^d\right) & \textit{if } d > \log_b a \end{array} \right\} \textit{ for } n = b^k,\ k = 1, 2, 3, \ldots$$

Asymptotic Recurrences

Proof

Asymptotic Recurrences

Proof



Level	**Nodes**	**Total**
0	1	cn^d
1	a	$\frac{a}{b^d}cn^d$
2	a^2	$\frac{a^2}{b^{2d}}cn^d$
$\vdots$	$\vdots$	
$\log_b n$	$n^{\log_b a}$	$\Theta\left(n^{\log_b a}\right)$

Asymptotic Recurrences

Proof

Level	Nodes							Total
0	1			cn^d				cn^d
1	a		$\frac{c}{b^d}n^d$	$\frac{c}{b^d}n^d$		$\cdots$	$\frac{c}{b^d}n^d$	$\frac{a}{b^d}cn^d$
2	a^2	$\frac{c}{b^{2d}}n^d$	$\cdots$	$\frac{c}{b^{2d}}n^d$		$\cdots$	$\cdots$	$\frac{a^2}{b^{2d}}cn^d$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	
$\log_b n$	$n^{\log_b a}$	$\Theta(1)$	$\Theta(1)$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\Theta\left(n^{\log_b a}\right)$

$$T(n) \leq \left(\sum_{i=0}^{\log_b n - 1} \frac{a^i}{b^{id}} cn^d \right) + \Theta(n^{\log_b a}) = cn^d \left(\sum_{i=0}^{\log_b n - 1} \frac{a^i}{b^{id}} \right) + \Theta(n^{\log_b a})$$

Asymptotic Recurrences

Proof

Level	**Nodes**							**Total**
0	1			cn^d				cn^d
1	a	$\frac{c}{b^d}n^d$		$\frac{c}{b^d}n^d$		$\cdots$	$\frac{c}{b^d}n^d$	$\frac{a}{b^d}cn^d$
2	a^2	$\frac{c}{b^{2d}}n^d$	$\cdots$	$\frac{c}{b^{2d}}n^d$		$\cdots$	$\cdots$	$\frac{a^2}{b^{2d}}cn^d$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	
$\log_b n$	$n^{\log_b a}$	$\Theta(1)$	$\Theta(1)$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\Theta\left(n^{\log_b a}\right)$

$$T(n) \geq \left(\sum_{i=0}^{\log_b n - 1} \frac{a^i}{b^{id}} cn^d\right) + \Theta(n^{\log_b a}) = cn^d \left(\sum_{i=0}^{\log_b n - 1} \frac{a^i}{b^{id}}\right) + \Theta(n^{\log_b a})$$

Asymptotic Recurrences

Let

$$\sum_{i=0}^{\log_b n-1} \frac{a^i}{b^{id}} = \sum_{i=0}^{\log_b n-1} \left(\frac{a}{b^d}\right)^i =: S(n)$$

Asymptotic Recurrences

Let

$$\sum_{i=0}^{\log_b n-1} \frac{a^i}{b^{id}} = \sum_{i=0}^{\log_b n-1} \left(\frac{a}{b^d}\right)^i =: S(n)$$

1. If $d < \log_b a$, then $a/b^d > 1$ and

Asymptotic Recurrences

Let

$$\sum_{i=0}^{\log_b n-1} \frac{a^i}{b^{id}} = \sum_{i=0}^{\log_b n-1} \left(\frac{a}{b^d}\right)^i =: S(n)$$

1. If $d < \log_b a$, then $a/b^d > 1$ and

$$S(n) = \sum_{i=0}^{\log_b n-1} \left(\frac{a}{b^d}\right)^i = \frac{(a/b^d)^{\log_b n} - 1}{a/b^d - 1} =$$

$$= \frac{n^{\log_b (a/b^d)} - 1}{a/b^d - 1} = \Theta\left(n^{\log_b a - d}\right).$$

Asymptotic Recurrences

Let

$$\sum_{i=0}^{\log_b n-1} \frac{a^i}{b^{id}} = \sum_{i=0}^{\log_b n-1} \left(\frac{a}{b^d}\right)^i =: S(n)$$

1. If $d < \log_b a$, then $a/b^d > 1$ and

$$S(n) = \sum_{i=0}^{\log_b n-1} \left(\frac{a}{b^d}\right)^i = \frac{(a/b^d)^{\log_b n} - 1}{a/b^d - 1} =$$

$$= \frac{n^{\log_b (a/b^d)} - 1}{a/b^d - 1} = \Theta\left(n^{\log_b a - d}\right).$$

As a result,

$$T(n) \leq cn^d S(n) + \Theta(n^{\log_b a}) = O(n^{\log_b a}).$$

Asymptotic Recurrences

Let

$$\sum_{i=0}^{\log_b n-1} \frac{a^i}{b^{id}} = \sum_{i=0}^{\log_b n-1} \left(\frac{a}{b^d}\right)^i =: S(n)$$

1. If $d < \log_b a$, then $a/b^d > 1$ and

$$S(n) = \sum_{i=0}^{\log_b n-1} \left(\frac{a}{b^d}\right)^i = \frac{(a/b^d)^{\log_b n} - 1}{a/b^d - 1} =$$

$$= \frac{n^{\log_b (a/b^d)} - 1}{a/b^d - 1} = \Theta\left(n^{\log_b a - d}\right).$$

As a result,

$$T(n) \geq cn^d S(n) + \Theta(n^{\log_b a}) = \Omega(n^{\log_b a}).$$

Asymptotic Recurrences

Let

$$\sum_{i=0}^{\log_b n-1} \frac{a^i}{b^{id}} = \sum_{i=0}^{\log_b n-1} \left(\frac{a}{b^d}\right)^i =: S(n)$$

2. If $d = \log_b a$, then $a/b^d = 1$ and

Asymptotic Recurrences

Let

$$\sum_{i=0}^{\log_b n-1} \frac{a^i}{b^{id}} = \sum_{i=0}^{\log_b n-1} \left(\frac{a}{b^d}\right)^i =: S(n)$$

2. If $d = \log_b a$, then $a/b^d = 1$ and

$$S(n) = \log_b n = \Theta(\log n).$$

Asymptotic Recurrences

Let

$$\sum_{i=0}^{\log_b n-1} \frac{a^i}{b^{id}} = \sum_{i=0}^{\log_b n-1} \left(\frac{a}{b^d}\right)^i =: S(n)$$

2. If $d = \log_b a$, then $a/b^d = 1$ and

$$S(n) = \log_b n = \Theta(\log n).$$

As a result,

$$T(n) \le cn^d S(n) + \Theta(n^{\log_b a}) = O(n^d \log n).$$

Asymptotic Recurrences

Let

$$\sum_{i=0}^{\log_b n-1} \frac{a^i}{b^{id}} = \sum_{i=0}^{\log_b n-1} \left(\frac{a}{b^d}\right)^i =: S(n)$$

2. If $d = \log_b a$, then $a/b^d = 1$ and

$$S(n) = \log_b n = \Theta(\log n).$$

As a result,

$$T(n) \geq cn^d S(n) + \Theta(n^{\log_b a}) = \Omega(n^d \log n).$$

Asymptotic Recurrences

Let

$$\sum_{i=0}^{\log_b n-1} \frac{a^i}{b^{id}} = \sum_{i=0}^{\log_b n-1} \left(\frac{a}{b^d}\right)^i =: S(n)$$

3. If $d > \log_b a$, then $a/b^d < 1$ and

Asymptotic Recurrences

Let

$$\sum_{i=0}^{\log_b n - 1} \frac{a^i}{b^{id}} = \sum_{i=0}^{\log_b n - 1} \left(\frac{a}{b^d}\right)^i =: S(n)$$

3. If $d > \log_b a$, then $a/b^d < 1$ and

$$S(n) = \sum_{i=0}^{\log_b n - 1} \left(\frac{a}{b^d}\right)^i \leq \sum_{i=0}^{\infty} \left(\frac{a}{b^d}\right)^i = O(1).$$

Asymptotic Recurrences

Let

$$\sum_{i=0}^{\log_b n-1} \frac{a^i}{b^{id}} = \sum_{i=0}^{\log_b n-1} \left(\frac{a}{b^d}\right)^i =: S(n)$$

3. If $d > \log_b a$, then $a/b^d < 1$ and

$$S(n) = \sum_{i=0}^{\log_b n-1} \left(\frac{a}{b^d}\right)^i \leq \sum_{i=0}^{\infty} \left(\frac{a}{b^d}\right)^i = O(1).$$

   Moreover,

$$S(n) \geq 1 = \Omega(1).$$

Asymptotic Recurrences

Let

$$\sum_{i=0}^{\log_b n-1} \frac{a^i}{b^{id}} = \sum_{i=0}^{\log_b n-1} \left(\frac{a}{b^d}\right)^i =: S(n)$$

3. If $d > \log_b a$, then $a/b^d < 1$ and

$$S(n) = \sum_{i=0}^{\log_b n-1} \left(\frac{a}{b^d}\right)^i \leq \sum_{i=0}^{\infty} \left(\frac{a}{b^d}\right)^i = O(1).$$

Moreover,

$$S(n) \geq 1 = \Omega(1).$$

Hence,

$$S(n) = \Theta(1).$$

Asymptotic Recurrences

Let

$$\sum_{i=0}^{\log_b n-1} \frac{a^i}{b^{id}} = \sum_{i=0}^{\log_b n-1} \left(\frac{a}{b^d}\right)^i =: S(n)$$

3. If $d > \log_b a$, then $a/b^d < 1$ and

$$S(n) = \sum_{i=0}^{\log_b n-1} \left(\frac{a}{b^d}\right)^i \leq \sum_{i=0}^{\infty} \left(\frac{a}{b^d}\right)^i = O(1).$$

   Moreover,

$$S(n) \geq 1 = \Omega(1).$$

   Hence,

$$S(n) = \Theta(1).$$

   As a result,

$$T(n) \leq cn^d S(n) + \Theta(n^{\log_b a}) = O(n^d).$$

Asymptotic Recurrences

Let

$$\sum_{i=0}^{\log_b n-1} \frac{a^i}{b^{id}} = \sum_{i=0}^{\log_b n-1} \left(\frac{a}{b^d}\right)^i =: S(n)$$

3. If $d > \log_b a$, then $a/b^d < 1$ and

$$S(n) = \sum_{i=0}^{\log_b n-1} \left(\frac{a}{b^d}\right)^i \leq \sum_{i=0}^{\infty} \left(\frac{a}{b^d}\right)^i = O(1).$$

   Moreover,

$$S(n) \geq 1 = \Omega(1).$$

   Hence,

$$S(n) = \Theta(1).$$

   As a result,

$$T(n) \geq cn^d S(n) + \Theta(n^{\log_b a}) = \Omega(n^d).$$

Asymptotic Recurrences

Corollary

Let $T(n)$ be a function satisfying

$$T(n) = aT(n/b) + \Theta(n^d) \qquad \textit{for } n = b^k,\ k = 1, 2, 3, \ldots,$$
$$T(1) = \Theta(1),$$

where $a \geq 1$ and $b \geq 2$ are in $\mathbb{N}$, and $d > 0$ is in $\mathbb{R}$. Then

$$T(n) = \left\{ \begin{array}{ll} \Theta\left(n^{\log_b a}\right) & \textit{if } d < \log_b a \\ \Theta\left(n^d \log n\right) & \textit{if } d = \log_b a \\ \Theta\left(n^d\right) & \textit{if } d > \log_b a \end{array} \right\} \textit{ for } n = b^k,\ k = 1, 2, 3, \ldots$$

Asymptotic Recurrences

Corollary

Let $T(n)$ be a function satisfying

$$T(n) = aT(n/b) + \Theta(n^d) \qquad \textit{for } n = b^k,\ k = 1, 2, 3, \ldots,$$
$$T(1) = \Theta(1),$$

where $a \geq 1$ and $b \geq 2$ are in $\mathbb{N}$, and $d > 0$ is in $\mathbb{R}$. Then

$$T(n) = \left\{ \begin{array}{ll} \Theta\left(n^{\log_b a}\right) & \textit{if } d < \log_b a \\ \Theta\left(n^d \log n\right) & \textit{if } d = \log_b a \\ \Theta\left(n^d\right) & \textit{if } d > \log_b a \end{array} \right\} \textit{ for } n = b^k,\ k = 1, 2, 3, \ldots$$

- Holds for non-exact-powers of b as well

Asymptotic Recurrences

Corollary

Let $T(n)$ be a function satisfying

$$\begin{aligned} &T(n) = aT(n/b) + \Theta(n^d) \qquad \textit{for } n = b^k,\ k = 1,2,3,\ldots, \\ &T(1) = \Theta(1), \end{aligned}$$

where $a \geq 1$ and $b \geq 2$ are in $\mathbb{N}$, and $d > 0$ is in $\mathbb{R}$. Then

$$T(n) = \left\{ \begin{array}{ll} \Theta\left(n^{\log_b a}\right) & \textit{if } d < \log_b a \\ \Theta\left(n^d \log n\right) & \textit{if } d = \log_b a \\ \Theta\left(n^d\right) & \textit{if } d > \log_b a \end{array} \right\} \textit{ for } n = b^k,\ k = 1,2,3,\ldots$$

- Holds for non-exact-powers of b as well
- More general statement: Master theorem (see Cormen et al.)

Analysis of Merge Sort

$\text{MERGESORT}(a, \mathsf{f}, \mathsf{l})$:

Analysis of Merge Sort

$\text{MERGESORT}(a, \mathsf{f}, \mathsf{l})$:

Input : Integer $n \geq 1$; Array $a = \langle a[1], \ldots, a[n] \rangle$ of elements of S;
Integers f, l such that $1 \leq \mathsf{f} \leq \mathsf{l} \leq n$

Behaviour: Sorts the subarray $\langle a[\mathsf{f}], \ldots, a[\mathsf{l}] \rangle$ in increasing order

```
if f = l then
    return
else
    m ← ⌊(f + l)/2⌋;
    MERGESORT(a, f, m);
    MERGESORT(a, m + 1, l);
    MERGE(a, f, m, l);
end
```

Analysis of Merge Sort

$\text{MERGESORT}(a, \mathsf{f}, \mathsf{l})$:

Input : Integer $n \geq 1$; Array $a = \langle a[1], \ldots, a[n] \rangle$ of elements of S;
Integers f, l such that $1 \leq \mathsf{f} \leq \mathsf{l} \leq n$

Behaviour: Sorts the subarray $\langle a[\mathsf{f}], \ldots, a[\mathsf{l}] \rangle$ in increasing order

```
if f = l then
    return
else
    m ← ⌊(f + l)/2⌋;
    MERGESORT(a, f, m);
    MERGESORT(a, m + 1, l);
    MERGE(a, f, m, l);
end
```

- Time complexity $T(n)$ of $\text{MERGESORT}(a, 1, n)$:

Analysis of Merge Sort

MERGESORT(a, f, l):

Input : Integer $n \geq 1$; Array $a = \langle a[1], \ldots, a[n] \rangle$ of elements of S;
Integers f, l such that $1 \leq \mathrm{f} \leq \mathrm{l} \leq n$

Behaviour: Sorts the subarray $\langle a[\mathrm{f}], \ldots, a[\mathrm{l}] \rangle$ in increasing order

```
if f = l then
|   return
else
|   m ← ⌊(f + l)/2⌋;
|   MERGESORT(a, f, m);
|   MERGESORT(a, m + 1, l);
|   MERGE(a, f, m, l);
end
```

- Time complexity $T(n)$ of MERGESORT(a, 1, n):

$$T(n) = 2T(n/2) + \Theta(n)$$
$$T(1) = \Theta(1)$$

Analysis of Merge Sort

MERGESORT(a, f, l):

Input : Integer $n \geq 1$; Array $a = \langle a[1], \ldots, a[n] \rangle$ of elements of S;
Integers f, l such that $1 \leq \mathrm{f} \leq \mathrm{l} \leq n$

Behaviour: Sorts the subarray $\langle a[\mathrm{f}], \ldots, a[\mathrm{l}] \rangle$ in increasing order

if $\mathrm{f} = \mathrm{l}$ **then**
  return
else
  $\mathrm{m} \leftarrow \lfloor (\mathrm{f} + \mathrm{l})/2 \rfloor$;
  MERGESORT(a, f, m);
  MERGESORT(a, m + 1, l);
  MERGE(a, f, m, l);
end

- Time complexity $T(n)$ of MERGESORT(a, 1, n):

$$T(n) = 2T(n/2) + \Theta(n)$$
$$T(1) = \Theta(1)$$

- Solution: $T(n) = \Theta(n \log n)$

Analysis of Merge Sort

$\mathrm{MERGESORT}(a, \mathsf{f}, \mathsf{l})$:

Input : Integer $n \geq 1$; Array $a = \langle a[1], \ldots, a[n] \rangle$ of elements of S;
Integers f, l such that $1 \leq \mathsf{f} \leq \mathsf{l} \leq n$

Behaviour: Sorts the subarray $\langle a[\mathsf{f}], \ldots, a[\mathsf{l}] \rangle$ in increasing order

```
if f = l then
|   return
else
|   m ← ⌊(f + l)/2⌋;
|   MERGESORT(a, f, m);
|   MERGESORT(a, m + 1, l);
|   MERGE(a, f, m, l);
end
```

- Time complexity $T(n)$ of $\mathrm{MERGESORT}(a, 1, n)$:

$$T(n) = 2T(n/2) + \Theta(n)$$
$$T(1) = \Theta(1)$$

- Solution: $T(n) = \Theta(n \log n)$
- Better than Insertion sort...

Multiplication of Square Matrices

$$\begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \dots & a_{n,n} \end{pmatrix} \cdot \begin{pmatrix} b_{1,1} & b_{1,2} & \dots & b_{1,n} \\ b_{2,1} & b_{2,2} & \dots & b_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n,1} & b_{n,2} & \dots & b_{n,n} \end{pmatrix} =$$

$$= \begin{pmatrix} c_{1,1} & c_{1,2} & \dots & c_{1,n} \\ c_{2,1} & c_{2,2} & \dots & c_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n,1} & c_{n,2} & \dots & c_{n,n} \end{pmatrix}$$

Multiplication of Square Matrices

$$\begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \dots & a_{n,n} \end{pmatrix} \cdot \begin{pmatrix} b_{1,1} & b_{1,2} & \dots & b_{1,n} \\ b_{2,1} & b_{2,2} & \dots & b_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n,1} & b_{n,2} & \dots & b_{n,n} \end{pmatrix} =$$

$$= \begin{pmatrix} c_{1,1} & c_{1,2} & \dots & c_{1,n} \\ c_{2,1} & c_{2,2} & \dots & c_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n,1} & c_{n,2} & \dots & c_{n,n} \end{pmatrix}$$

$$c_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j}, \qquad i, j = 1, \dots, n$$

Naive Matrix Multiplication

Input : Integer $n \geq 1$; Matrices $A = (a[i,j] \mid 1 \leq i,j \leq n)$,
$B = (b[i,j] \mid 1 \leq i,j \leq n)$ over some semiring
Output: The matrix $A \cdot B = (c[i,j] \mid 1 \leq i,j \leq n)$

```
for i ← 1 to n do
    for j ← 1 to n do
        c[i,j] ← 0;
        for k ← 1 to n do
            c[i,j] ← c[i,j] + a[i,k] * b[k,j];
        end
    end
end
```

Naive Matrix Multiplication

Input : Integer $n \geq 1$; Matrices $A = (a[i,j] \mid 1 \leq i,j \leq n)$,
$B = (b[i,j] \mid 1 \leq i,j \leq n)$ over some semiring
Output: The matrix $A \cdot B = (c[i,j] \mid 1 \leq i,j \leq n)$

```
for i ← 1 to n do
    for j ← 1 to n do
        c[i,j] ← 0;
        for k ← 1 to n do
            c[i,j] ← c[i,j] + a[i,k] * b[k,j];
        end
    end
end
```

- $\Theta(n^3)$ arithmetic operations

Naive Matrix Multiplication

Input : Integer $n \geq 1$; Matrices $A = (a[i,j] \mid 1 \leq i,j \leq n)$,
$B = (b[i,j] \mid 1 \leq i,j \leq n)$ over some semiring
Output: The matrix $A \cdot B = (c[i,j] \mid 1 \leq i,j \leq n)$

```
for i ← 1 to n do
    for j ← 1 to n do
        c[i,j] ← 0;
        for k ← 1 to n do
            c[i,j] ← c[i,j] + a[i,k] * b[k,j];
        end
    end
end
```

- $\Theta(n^3)$ arithmetic operations
- Can we do better?

Matrix Multiplication: Divide and Conquer?

$$\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \cdot \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} = \begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix}$$

Matrix Multiplication: Divide and Conquer?

$$\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \cdot \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} = \begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix}$$

$$C_{1,1} = A_{1,1} \cdot B_{1,1} + A_{1,2} \cdot B_{2,1}$$
$$C_{1,2} = A_{1,1} \cdot B_{1,2} + A_{1,2} \cdot B_{2,2}$$
$$C_{2,1} = A_{2,1} \cdot B_{1,1} + A_{2,2} \cdot B_{2,1}$$
$$C_{2,2} = A_{2,1} \cdot B_{1,2} + A_{2,2} \cdot B_{2,2}$$

Matrix Multiplication: Divide and Conquer?

$$\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \cdot \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} = \begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix}$$

$$\begin{aligned} C_{1,1} &= A_{1,1} \cdot B_{1,1} + A_{1,2} \cdot B_{2,1} \\ C_{1,2} &= A_{1,1} \cdot B_{1,2} + A_{1,2} \cdot B_{2,2} \\ C_{2,1} &= A_{2,1} \cdot B_{1,1} + A_{2,2} \cdot B_{2,1} \\ C_{2,2} &= A_{2,1} \cdot B_{1,2} + A_{2,2} \cdot B_{2,2} \end{aligned}$$

- Suppose that the blocks are square matrices of sizes $\lfloor n/2 \rfloor \times \lfloor n/2 \rfloor$ or $\lceil n/2 \rceil \times \lceil n/2 \rceil$

Matrix Multiplication: Divide and Conquer?

$$\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \cdot \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} = \begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix}$$

$$\begin{aligned} C_{1,1} &= A_{1,1} \cdot B_{1,1} + A_{1,2} \cdot B_{2,1} \\ C_{1,2} &= A_{1,1} \cdot B_{1,2} + A_{1,2} \cdot B_{2,2} \\ C_{2,1} &= A_{2,1} \cdot B_{1,1} + A_{2,2} \cdot B_{2,1} \\ C_{2,2} &= A_{2,1} \cdot B_{1,2} + A_{2,2} \cdot B_{2,2} \end{aligned}$$

- Suppose that the blocks are square matrices of sizes $\lfloor n/2 \rfloor \times \lfloor n/2 \rfloor$ or $\lceil n/2 \rceil \times \lceil n/2 \rceil$
- (If not, we can simply forget about at most one row or column, which we may compute separately in $\Theta(n^2)$ operations)

Matrix Multiplication: Divide and Conquer?

$$\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \cdot \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} = \begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix}$$

$$\begin{aligned} C_{1,1} &= A_{1,1} \cdot B_{1,1} + A_{1,2} \cdot B_{2,1} \\ C_{1,2} &= A_{1,1} \cdot B_{1,2} + A_{1,2} \cdot B_{2,2} \\ C_{2,1} &= A_{2,1} \cdot B_{1,1} + A_{2,2} \cdot B_{2,1} \\ C_{2,2} &= A_{2,1} \cdot B_{1,2} + A_{2,2} \cdot B_{2,2} \end{aligned}$$

- Suppose that the blocks are square matrices of sizes $\lfloor n/2 \rfloor \times \lfloor n/2 \rfloor$ or $\lceil n/2 \rceil \times \lceil n/2 \rceil$
- (If not, we can simply forget about at most one row or column, which we may compute separately in $\Theta(n^2)$ operations)
- $T(n) = 8T(n/2) + \Theta(n^2),\ T(1) = \Theta(1)$

Matrix Multiplication: Divide and Conquer?

$$\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \cdot \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} = \begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix}$$

$$\begin{aligned} C_{1,1} &= A_{1,1} \cdot B_{1,1} + A_{1,2} \cdot B_{2,1} \\ C_{1,2} &= A_{1,1} \cdot B_{1,2} + A_{1,2} \cdot B_{2,2} \\ C_{2,1} &= A_{2,1} \cdot B_{1,1} + A_{2,2} \cdot B_{2,1} \\ C_{2,2} &= A_{2,1} \cdot B_{1,2} + A_{2,2} \cdot B_{2,2} \end{aligned}$$

- Suppose that the blocks are square matrices of sizes $\lfloor n/2 \rfloor \times \lfloor n/2 \rfloor$ or $\lceil n/2 \rceil \times \lceil n/2 \rceil$
- (If not, we can simply forget about at most one row or column, which we may compute separately in $\Theta(n^2)$ operations)
- $T(n) = 8T(n/2) + \Theta(n^2),\ T(1) = \Theta(1)$
- Solution: $T(n) = \Theta(n^3)$

Matrix Multiplication: Divide and Conquer?

$$\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \cdot \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} = \begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix}$$

$$\begin{aligned} C_{1,1} &= A_{1,1} \cdot B_{1,1} + A_{1,2} \cdot B_{2,1} \\ C_{1,2} &= A_{1,1} \cdot B_{1,2} + A_{1,2} \cdot B_{2,2} \\ C_{2,1} &= A_{2,1} \cdot B_{1,1} + A_{2,2} \cdot B_{2,1} \\ C_{2,2} &= A_{2,1} \cdot B_{1,2} + A_{2,2} \cdot B_{2,2} \end{aligned}$$

- Suppose that the blocks are square matrices of sizes $\lfloor n/2 \rfloor \times \lfloor n/2 \rfloor$ or $\lceil n/2 \rceil \times \lceil n/2 \rceil$
- (If not, we can simply forget about at most one row or column, which we may compute separately in $\Theta(n^2)$ operations)
- $T(n) = 8T(n/2) + \Theta(n^2),\ T(1) = \Theta(1)$
- Solution: $T(n) = \Theta(n^3)$
- This is no better than the naive approach

Strassen's Algorithm

- Works for matrices over rings

Strassen's Algorithm

- Works for matrices over rings

$$\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \cdot \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} = \begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix}$$

Strassen's Algorithm

- Works for matrices over rings

$$\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \cdot \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} = \begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix}$$

$$\begin{aligned} C_{1,1} &= M_1 + M_2 - M_4 + M_6 \\ C_{1,2} &= M_4 + M_5 \\ C_{2,1} &= M_6 + M_7 \\ C_{2,2} &= M_2 - M_3 + M_5 - M_7 \end{aligned}$$

Strassen's Algorithm

- Works for matrices over rings

$$\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \cdot \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} = \begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix}$$

$$\begin{aligned} C_{1,1} &= M_1 + M_2 - M_4 + M_6 \\ C_{1,2} &= M_4 + M_5 \\ C_{2,1} &= M_6 + M_7 \\ C_{2,2} &= M_2 - M_3 + M_5 - M_7 \end{aligned}$$

where

$$\begin{aligned} M_1 &= (A_{1,2} - A_{2,2}) \cdot (B_{2,1} + B_{2,2}) \\ M_2 &= (A_{1,1} + A_{2,2}) \cdot (B_{1,1} + B_{2,2}) \\ M_3 &= (A_{1,1} - A_{2,1}) \cdot (B_{1,1} + B_{1,2}) \\ M_4 &= (A_{1,1} + A_{1,2}) \cdot B_{2,2} \\ M_5 &= A_{1,1} \cdot (B_{1,2} - B_{2,2}) \\ M_6 &= A_{2,2} \cdot (B_{2,1} - B_{1,1}) \\ M_7 &= (A_{2,1} + A_{2,2}) \cdot B_{1,1} \end{aligned}$$

Strassen's Algorithm

- Time complexity (in arithmetic operations):

Strassen's Algorithm

- Time complexity (in arithmetic operations):

$$T(n) = 7T(n/2) + \Theta(n^2)$$
$$T(1) = \Theta(1)$$

Strassen's Algorithm

- Time complexity (in arithmetic operations):

$$T(n) = 7T(n/2) + \Theta(n^2)$$
$$T(1) = \Theta(1)$$

- Solution: $T(n) = \Theta(n^{\log_2 7})$, where $\log_2 7$ is approximately 2.807

Strassen's Algorithm

- Time complexity (in arithmetic operations):

$$T(n) = 7T(n/2) + \Theta(n^2)$$
$$T(1) = \Theta(1)$$

- Solution: $T(n) = \Theta(n^{\log_2 7})$, where $\log_2 7$ is approximately 2.807
- The constant factor is pretty large (not ideal for small matrices)

Strassen's Algorithm

- Time complexity (in arithmetic operations):

$$T(n) = 7T(n/2) + \Theta(n^2)$$
$$T(1) = \Theta(1)$$

- Solution: $T(n) = \Theta(n^{\log_2 7})$, where $\log_2 7$ is approximately 2.807
- The constant factor is pretty large (not ideal for small matrices)
- There are some asymptotically faster algorithms, but with extremely large constants (limited practical value)

Strassen's Algorithm

- Time complexity (in arithmetic operations):

$$T(n) = 7\,T(n/2) + \Theta(n^2)$$
$$T(1) = \Theta(1)$$

- Solution: $T(n) = \Theta(n^{\log_2 7})$, where $\log_2 7$ is approximately 2.807
- The constant factor is pretty large (not ideal for small matrices)
- There are some asymptotically faster algorithms, but with extremely large constants (limited practical value)
- Current record of Le Gall (2014): $\Theta(n^{2.373\ldots})$