Forelesning 1

Bonusmateriale

Ting som ikke ble med i forelesningen, men som kanskje kan være av interesse

Induksjonseksempel

$$n! > 2^n$$

Et lite induksjonseksempel

Vil vise $n! > 2^n$ $\mathrm{P}(n)$

Vi vil vise dette for alle heltall $n \geqslant 4$

Vil vise $n! > 2^n$ $\mathrm{P}(n)$

Ekvivalent: Vi vil vise dette for et vilkårlig heltall $n \geqslant 4$

Vil vise	$n! > 2^n$	$\mathrm{P}(n)$
Grunntilfelle	$4! > 2^4$	$\mathrm{P}(4)$

$$4! = 24 > 15 = 2^4$$

Vil vise	$n! > 2^n$	$\mathrm{P}(n)$
Grunntilfelle	$4! > 2^4$	$\mathrm{P}(4)$

Vi starter her

Vil vise	$n! > 2^n$	$\mathrm{P}(n)$
Grunntilfelle	$4! > 2^4$	$\mathrm{P}(4)$

Vi vil så vise $\mathrm{P}(4) \implies \mathrm{P}(5) \implies \mathrm{P}(6) \cdots$

Vil vise	$n! > 2^n$	$\mathrm{P}(n)$
Grunntilfelle	$4! > 2^4$	$\mathrm{P}(4)$

Husk: n er vilkårlig, så det holder med $\mathrm{P}(n-1) \implies \mathrm{P}(n)$

mlh@ntnu.no

Vil vise	$n! > 2^n$	P(n)
Grunntilfelle	$4! > 2^4$	P(4)

Generelt: For å vise A $\implies$ B, anta A og utled B

Vil vise	$n! > 2^n$	$\mathrm{P}(n)$
Grunntilfelle	$4! > 2^4$	$\mathrm{P}(4)$
Induksjonshypotese	$(n-1)! > 2^{n-1}$	$\mathrm{P}(n-1)$

Vi antar altså $\mathrm{P}(n-1)$ og vil utlede $\mathrm{P}(n)$

Vil vise	$n! > 2^n$	$\mathrm{P}(n)$
Grunntilfelle	$4! > 2^4$	$\mathrm{P}(4)$
Induksjonshypotese	$(n-1)! > 2^{n-1}$	$\mathrm{P}(n-1)$
Induksjonstrinn	$\Longrightarrow \; n! > 2^n$	$\mathrm{P}(n)$

Vi antar altså $\mathrm{P}(n-1)$ og vil utlede $\mathrm{P}(n)$

Vil vise	$n! > 2^n$	$\mathrm{P}(n)$
Grunntilfelle	$4! > 2^4$	$\mathrm{P}(4)$
Induksjonshypotese	$(n-1)! > 2^{n-1}$	$\mathrm{P}(n-1)$
Induksjonstrinn	$\Longrightarrow\ n! > 2^n$	$\mathrm{P}(n)$

For å gjøre det, bryter vi ned $n!$ og 2^n rekursivt

Vil vise	$n! > 2^n$	$\mathrm{P}(n)$
Grunntilfelle	$4! > 2^4$	$\mathrm{P}(4)$
Induksjonshypotese	$(n-1)! > 2^{n-1}$	$\mathrm{P}(n-1)$
Induksjonstrinn	$\Longrightarrow\ n! > 2^n$	$\mathrm{P}(n)$
Rekursjon	$n! = n \cdot (n-1)!$	
	$2^n = 2 \cdot 2^{n-1}$	

For å gjøre det, bryter vi ned $n!$ og 2^n rekursivt

Vil vise	$n! > 2^n$	$\mathrm{P}(n)$
Grunntilfelle	$4! > 2^4$	$\mathrm{P}(4)$
Induksjonshypotese	$(n-1)! > 2^{n-1}$	$\mathrm{P}(n-1)$
Induksjonstrinn	$\implies n! > 2^n$	$\mathrm{P}(n)$
Rekursjon	$n! = n \cdot (n-1)!$	
	$2^n = 2 \cdot 2^{n-1}$	

Vi antar $(n-1)! > 2^{n-1}$ og vet $n > 2$, så $\mathrm{P}(n-1) \implies \mathrm{P}(n)$

Vil vise	$n! > 2^n$	$\mathrm{P}(n)$
Grunntilfelle	$4! > 2^4$	$\mathrm{P}(4)$
Induksjonshypotese	$(n-1)! > 2^{n-1}$	$\mathrm{P}(n-1)$
Induksjonstrinn	$\implies n! > 2^n$	$\mathrm{P}(n)$
Rekursjon	$n! = n \cdot (n-1)!$	
	$2^n = 2 \cdot 2^{n-1}$	

Vi vet nå at $\mathrm{P}(n)$ for $n = 4$ og $\mathrm{P}(n-1) \implies \mathrm{P}(n)$ for $n > 4$

Vil vise	$n! > 2^n$	$\mathrm{P}(n)$
Grunntilfelle	$4! > 2^4$	$\mathrm{P}(4)$
Induksjonshypotese	$(n-1)! > 2^{n-1}$	$\mathrm{P}(n-1)$
Induksjonstrinn	$\Longrightarrow \; n! > 2^n$	$\mathrm{P}(n)$
Rekursjon	$n! = n \cdot (n-1)!$	
	$2^n = 2 \cdot 2^{n-1}$	

Altså har vi vist at $n! > 2^n$ for alle $n \geqslant 4$

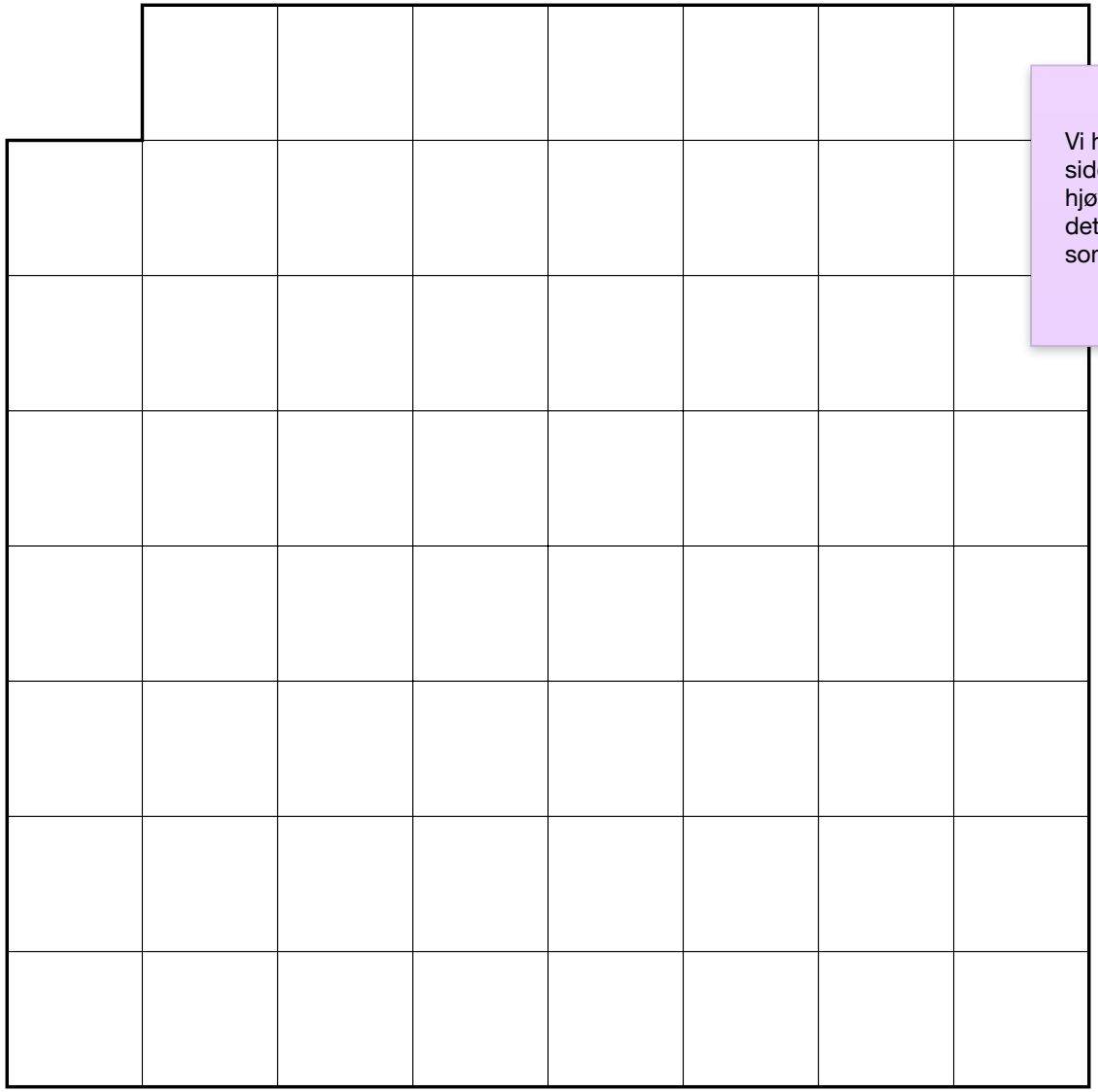
mlh@ntnu.no

Vil vise	$n! > 2^n$	$\mathrm{P}(n)$
Grunntilfelle	$4! > 2^4$	$\mathrm{P}(4)$
Induksjonshypotese	$(n-1)! > 2^{n-1}$	$\mathrm{P}(n-1)$
Induksjonstrinn	$\Longrightarrow \; n! > 2^n$	$\mathrm{P}(n)$
Rekursjon	$n! = n \cdot (n-1)!$	
	$2^n = 2 \cdot 2^{n-1}$	

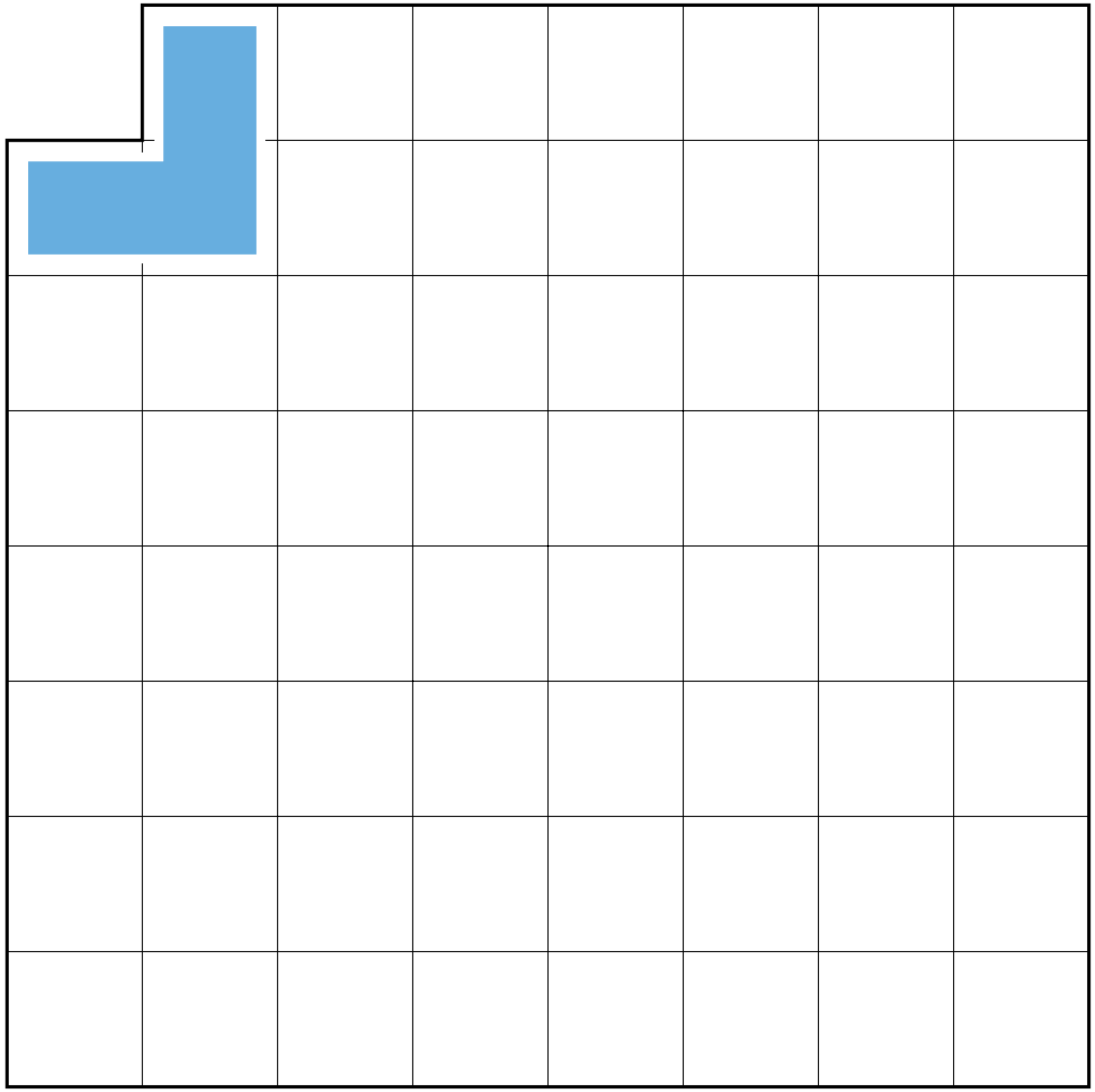
Den rekursive strukturen til $n!$ og 2^n var sentral!

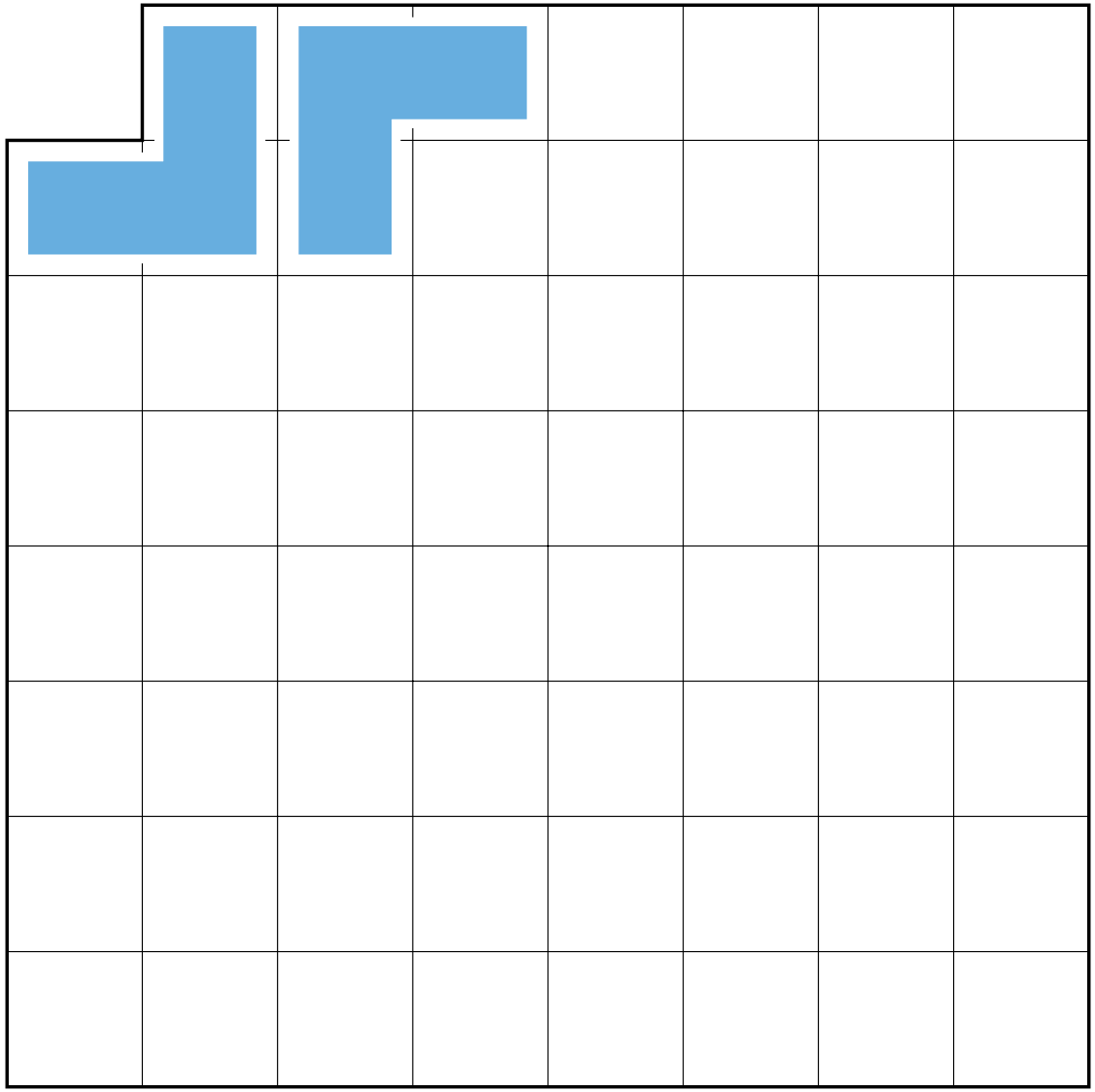
mlh@ntnu.no

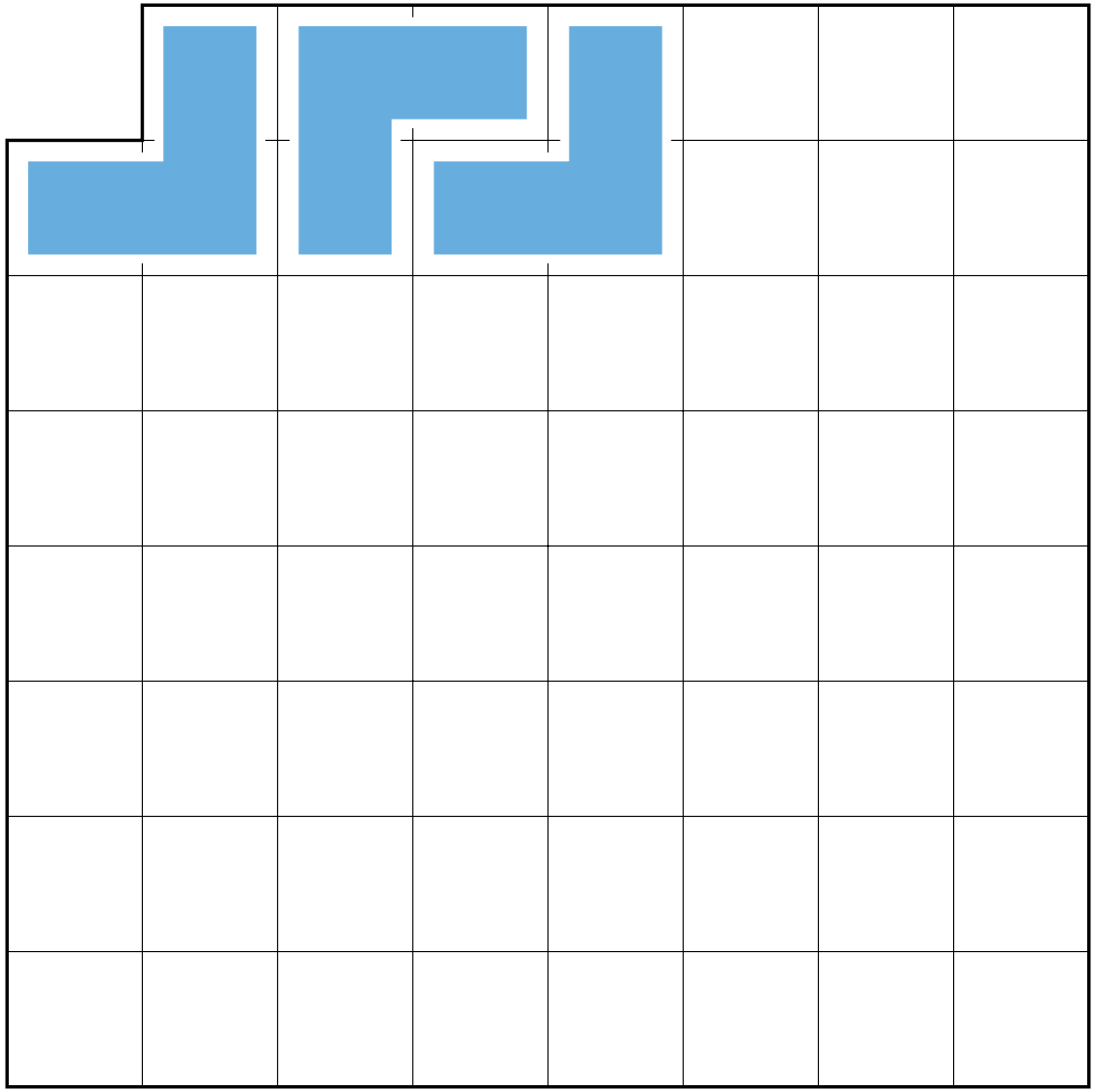
L-problemet

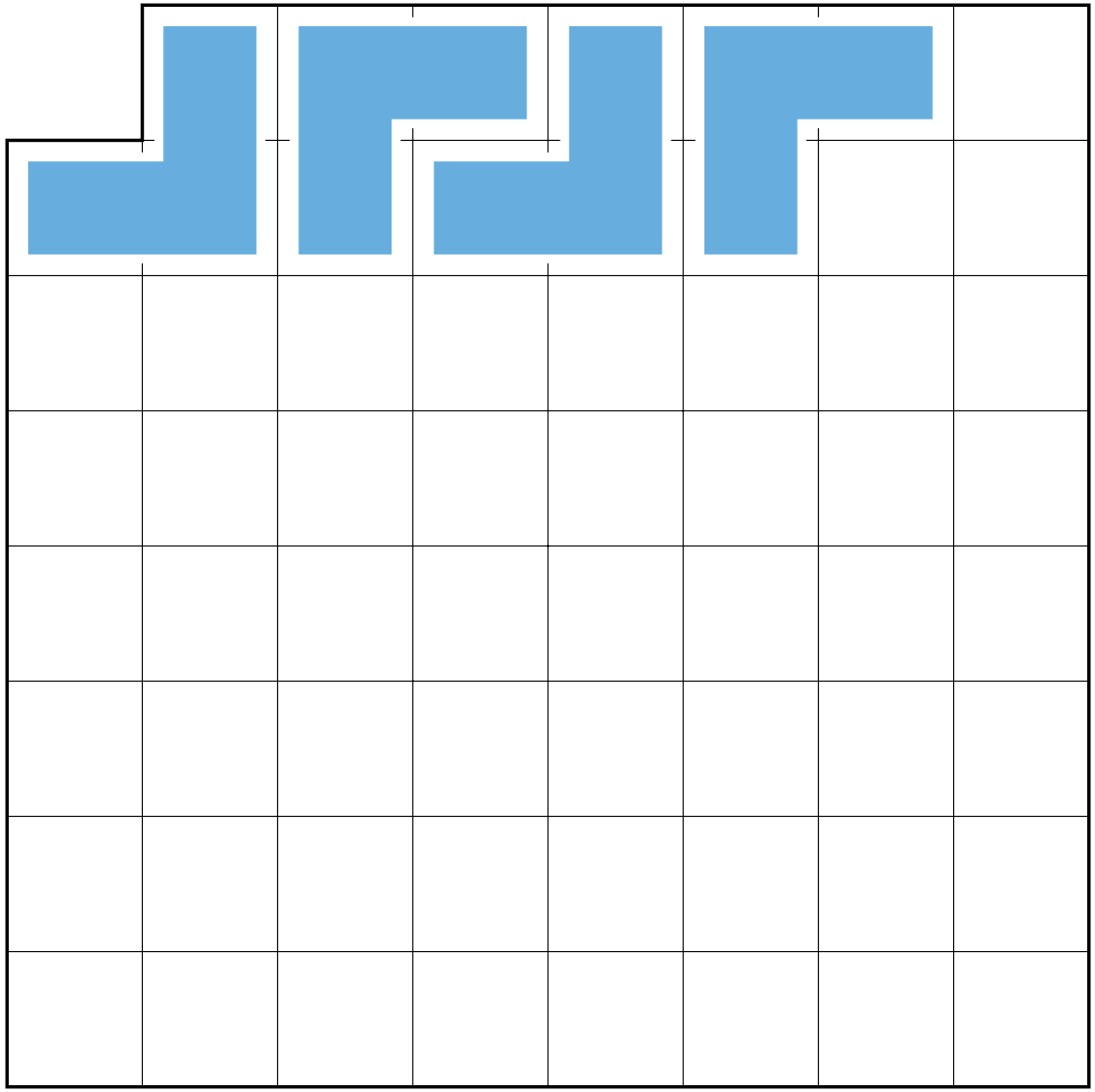


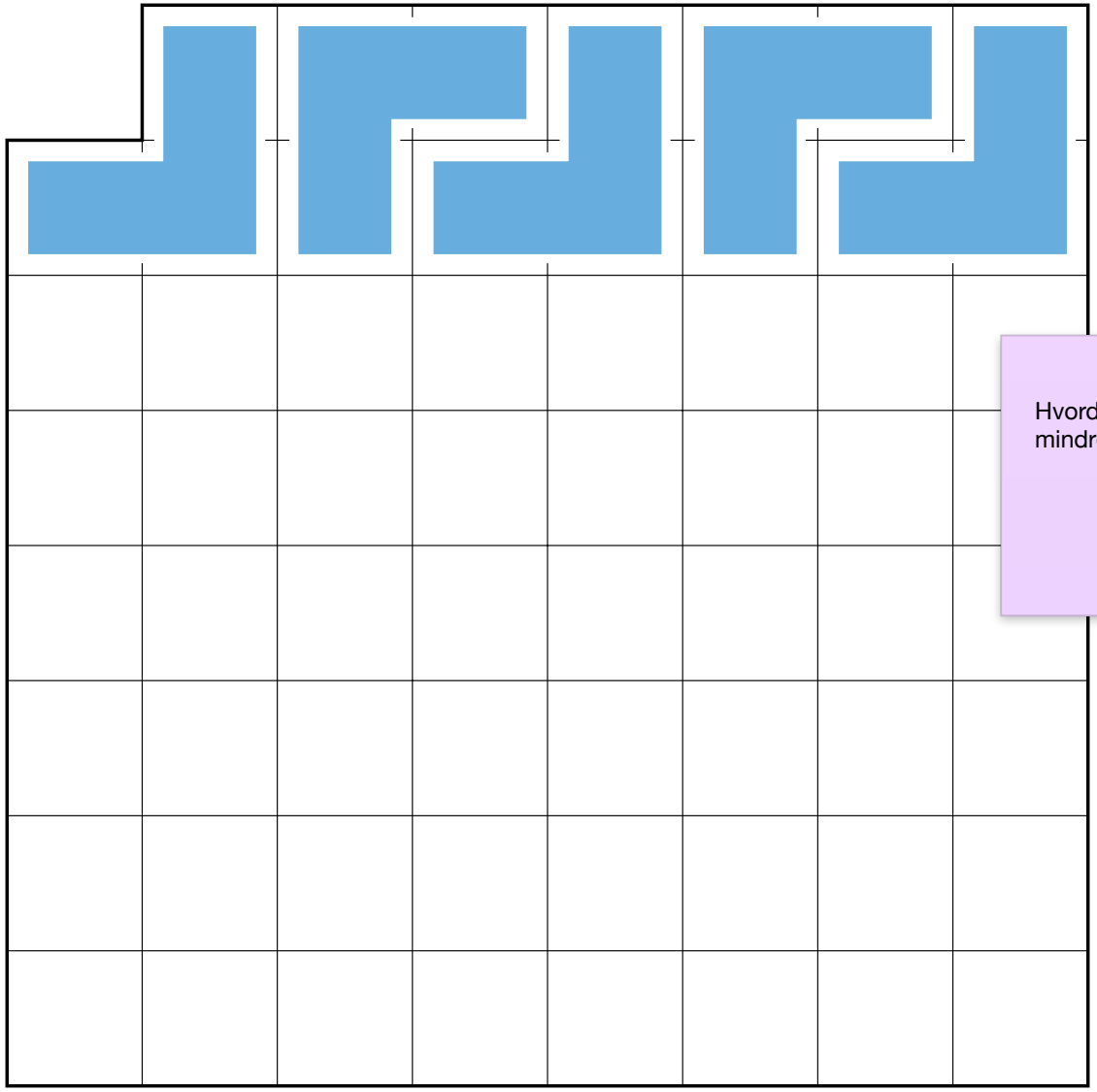
Vi har et kvadratisk rutenett der sidene er toerpotenser, og der ett hjørne mangler. Vi ønsker å dekke dette «brettet» med L-formede brikker som består av 3 ruter.



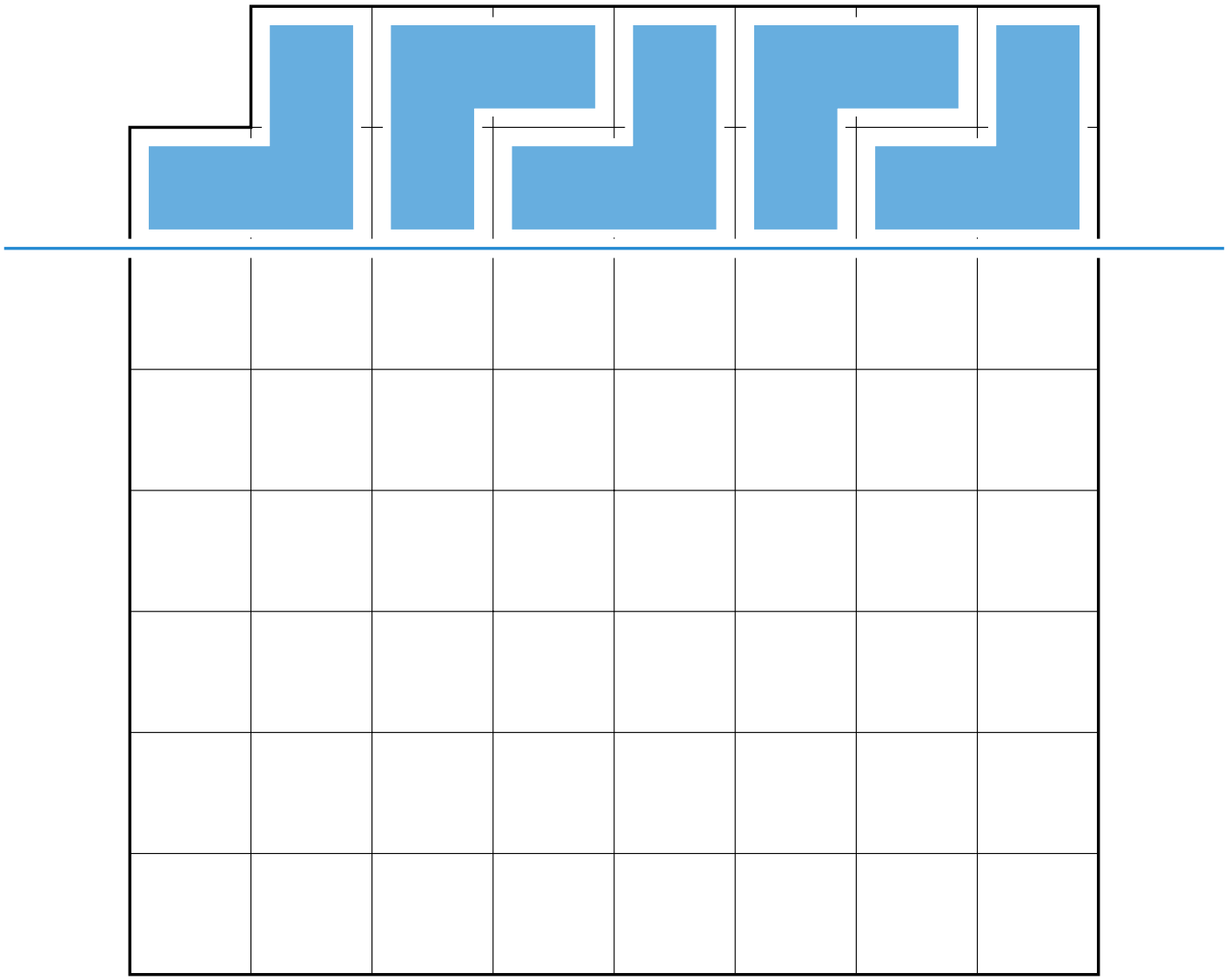


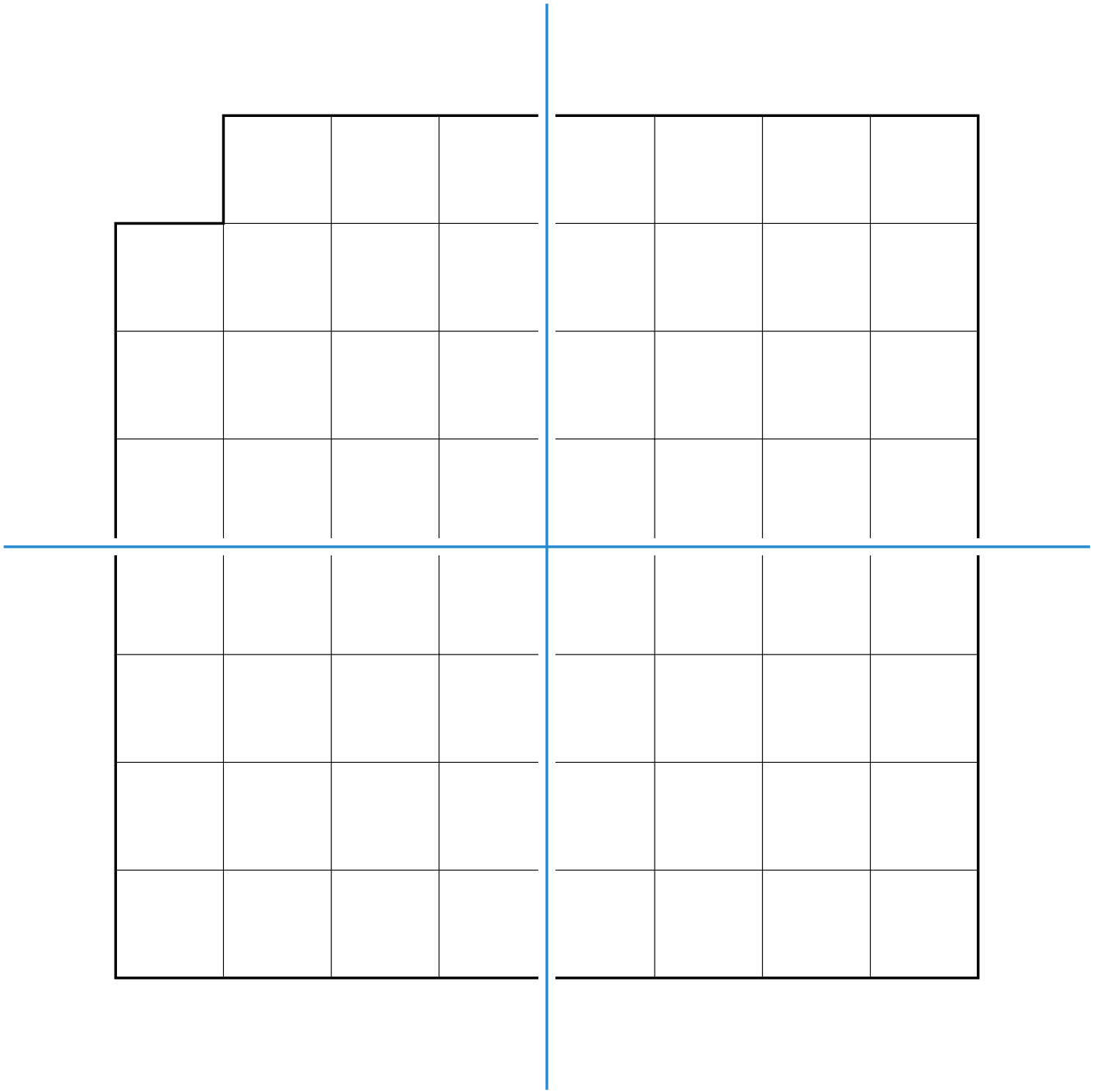


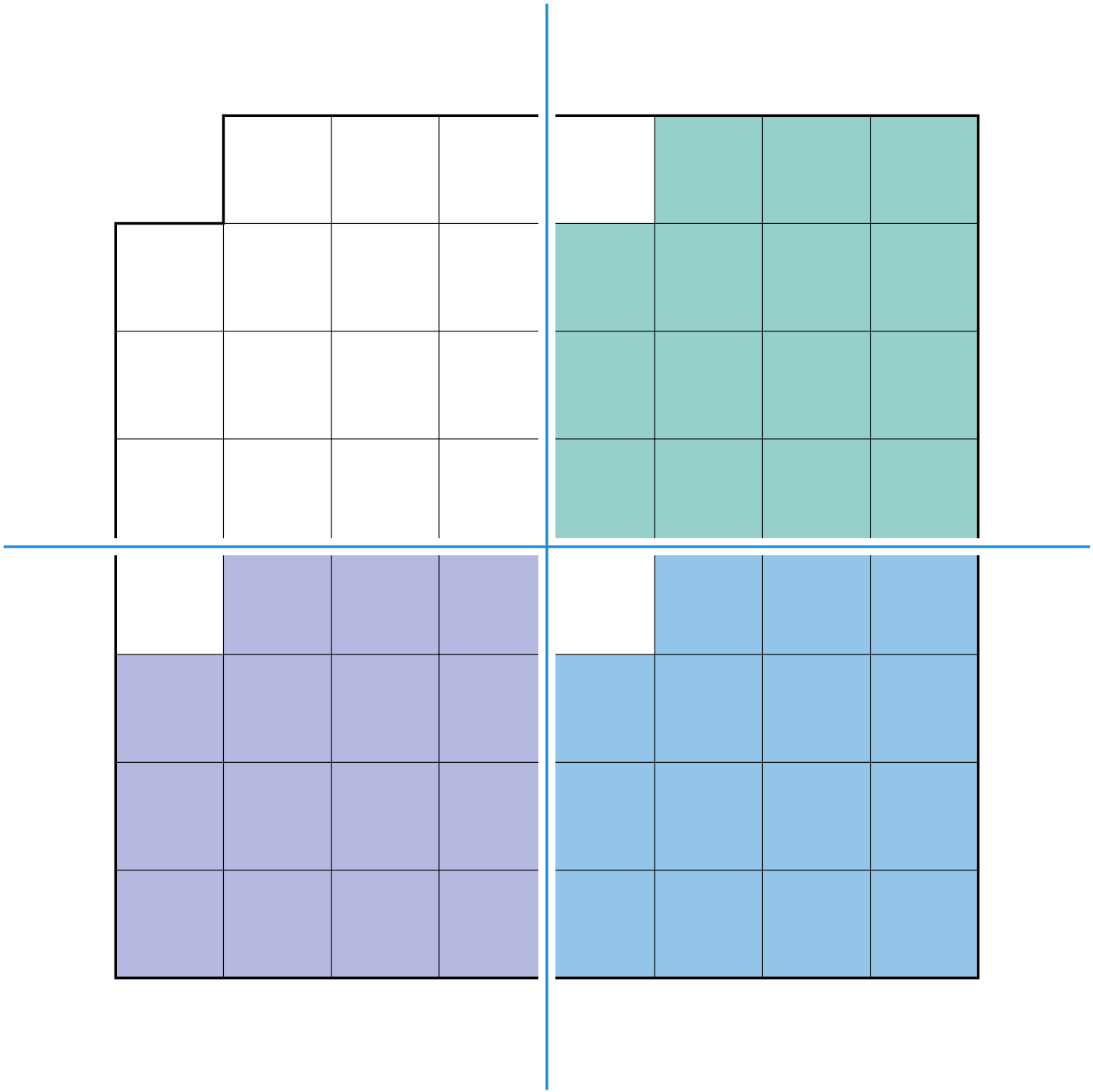


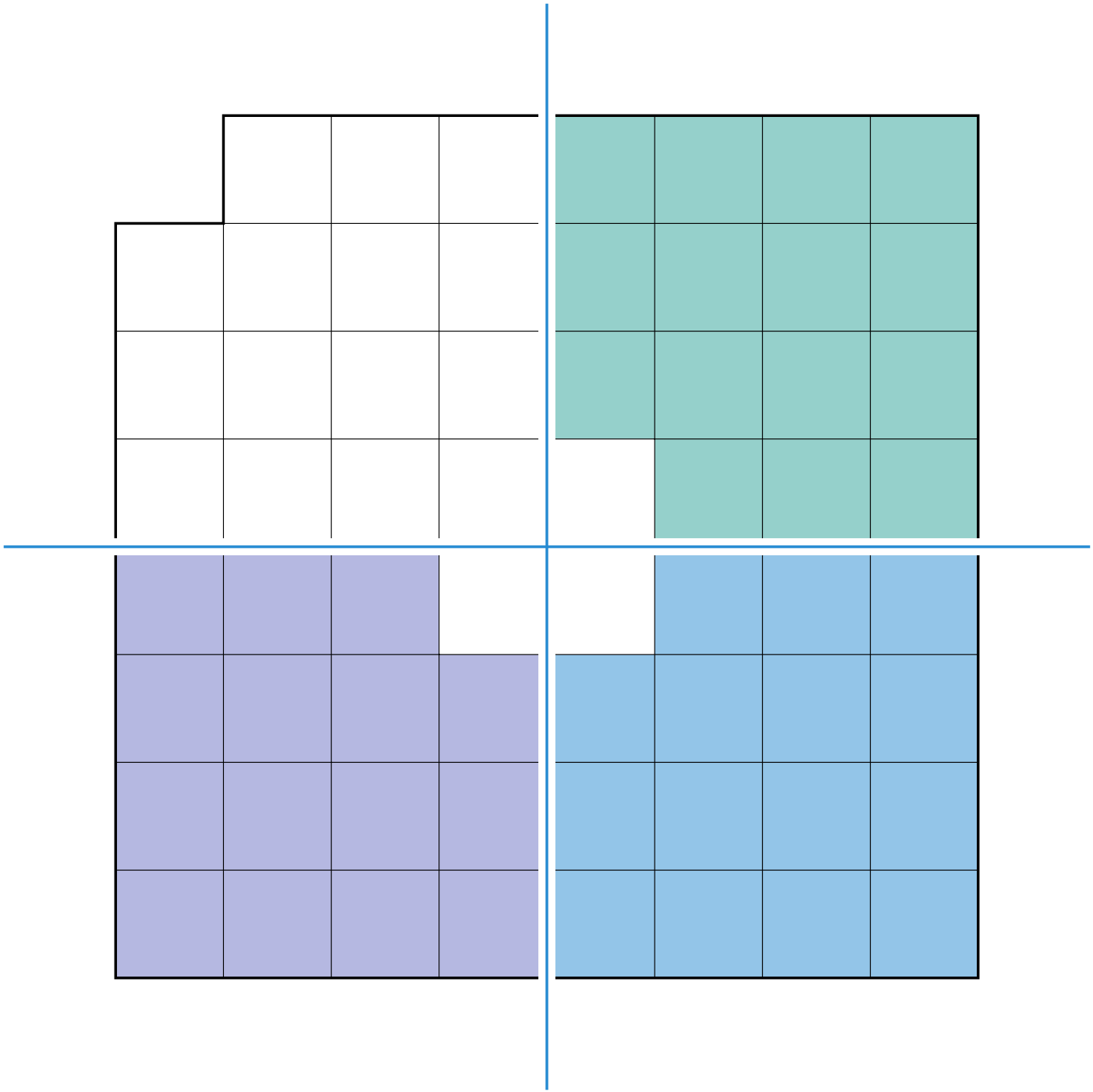


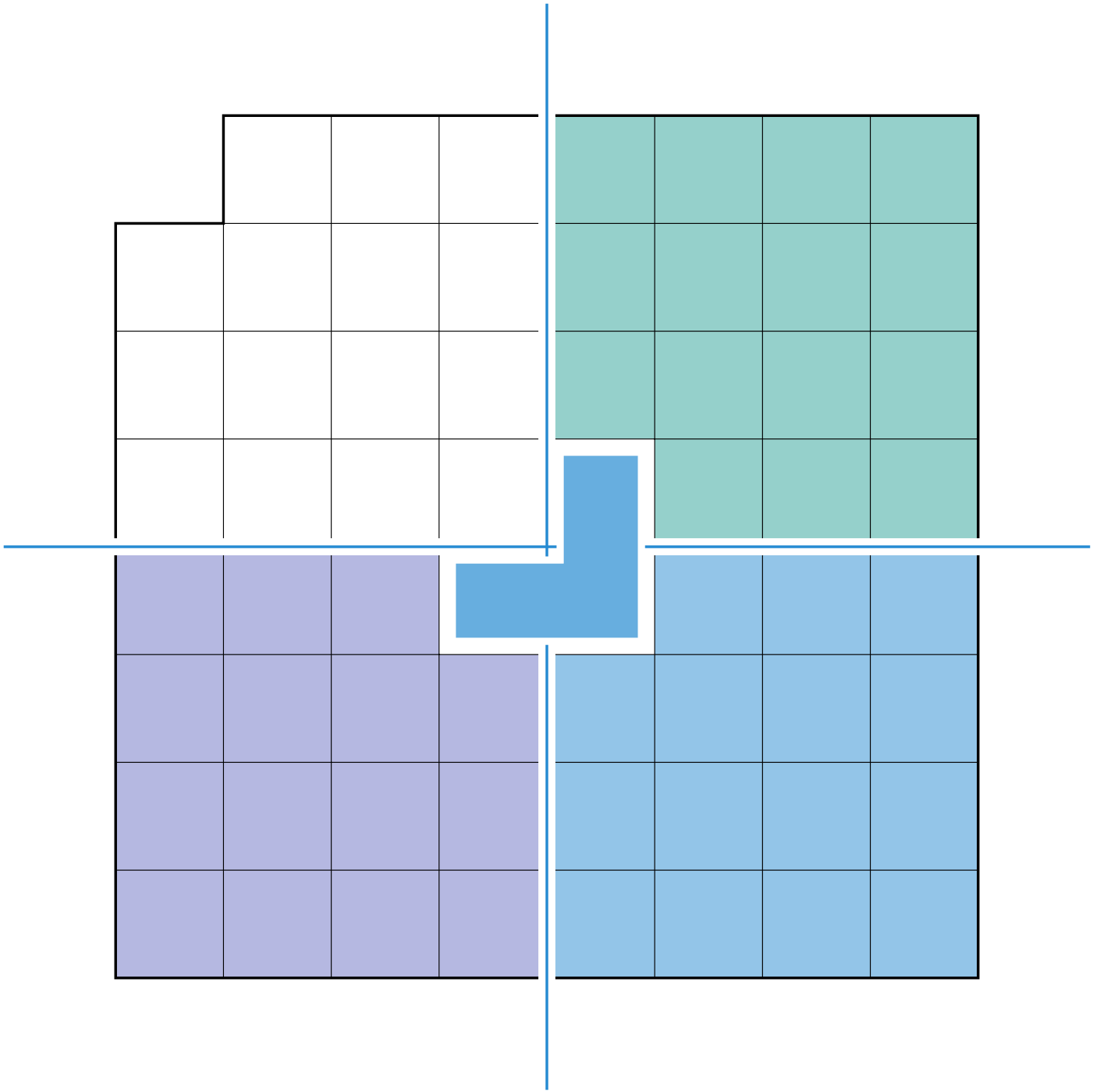
Hvordan kan vi spalte instansen i mindre instanser av samme problem?











Cover(A)

Dekk «nesten-kvadrat» A med L-brikker

COVER(A)
1 place middle L

Manglende hjørne i samme retning som A

COVER(A)
1 place middle L
2 **if** A is 2×2

Grunntilfelle: Vi har løst denne delinstansen!

```
COVER(A)
1  place middle L
2  if A is 2 × 2
3        return
```

Grunntilfelle: Vi har løst denne delinstansen!

```
COVER(A)
1  place middle L
2  if A is 2 × 2
3        return
4  for each quadrant Q
```

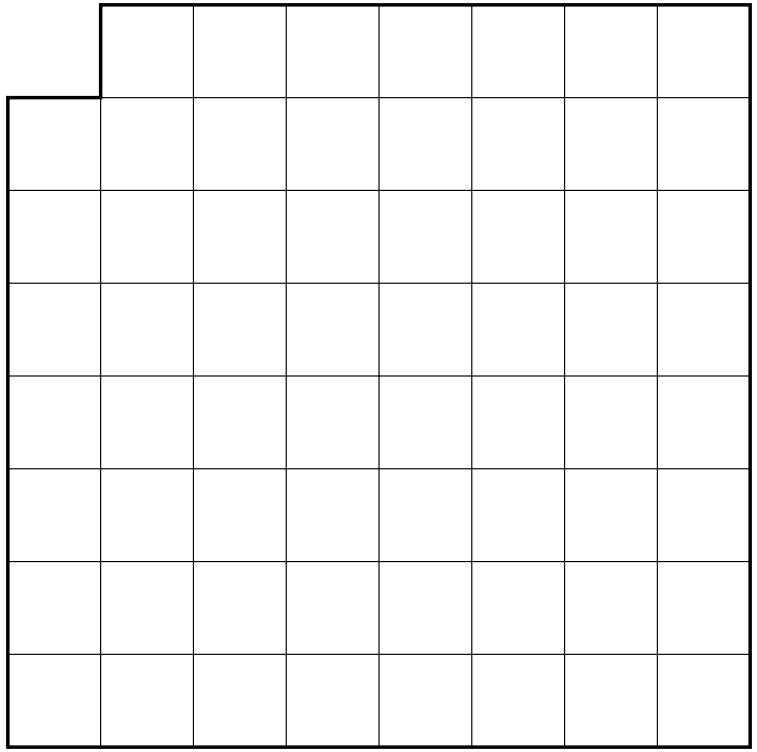
Spalt i fire delinstanser: Mindre kvadrater med manglende hjørne

```
COVER(A)
1  place middle L
2  if A is 2 × 2
3        return
4  for each quadrant Q
5        COVER(Q)
```

Løs disse fire. Nå vet vi jo hvordan!

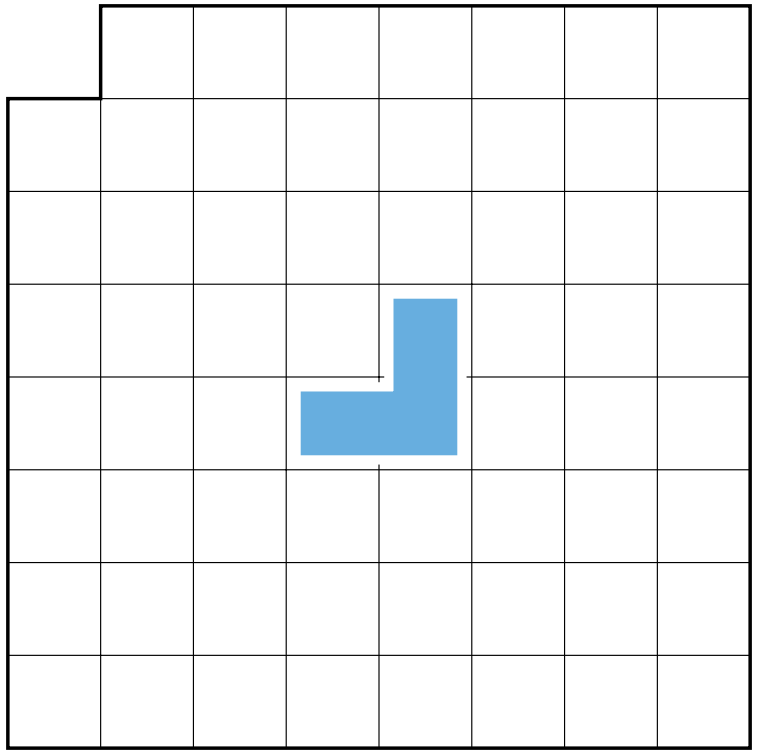
```
COVER(A)
1  place middle L
2  if A is 2 × 2
3        return
4  for each quadrant Q
5        COVER(Q)
```

A, Q = 



```
COVER(A)
1  place middle L
2  if A is 2 × 2
3        return
4  for each quadrant Q
5        COVER(Q)
```

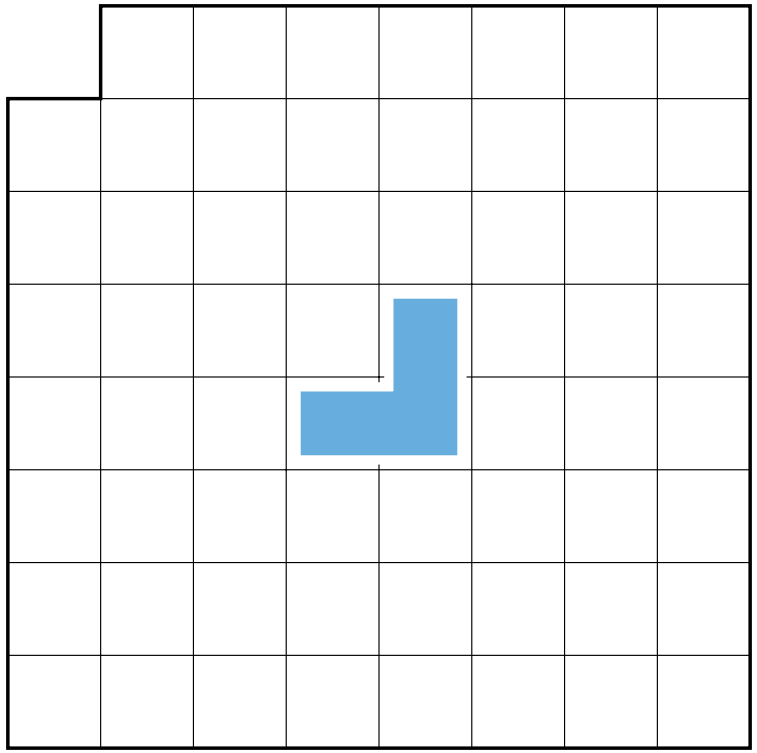
A, Q = ■, ⬚



mlh@ntnu.no

```
COVER(A)
1  place middle L
2  if A is 2 × 2
3        return
4  for each quadrant Q
5        COVER(Q)
```

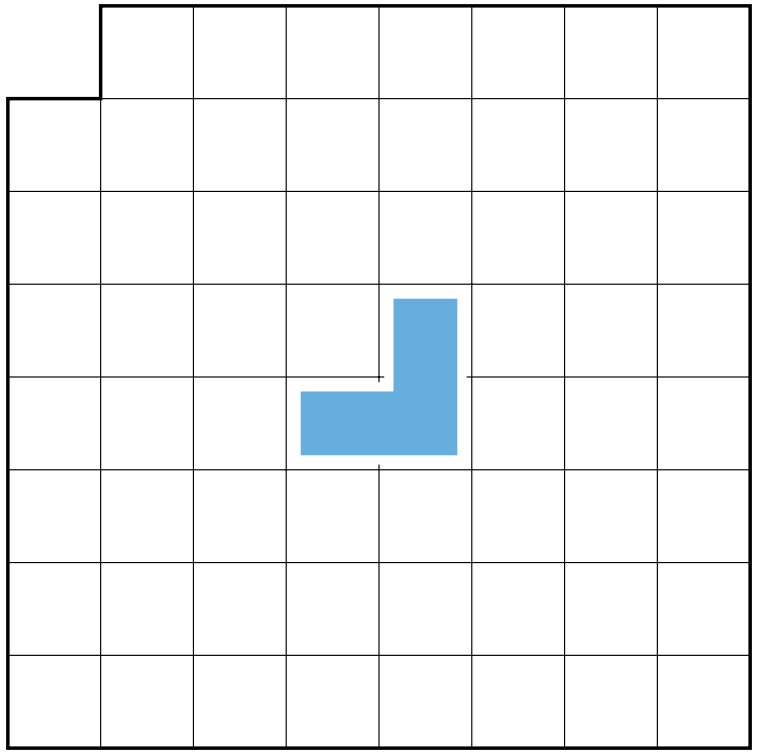
A, Q = ■, ⬚



mlh@ntnu.no

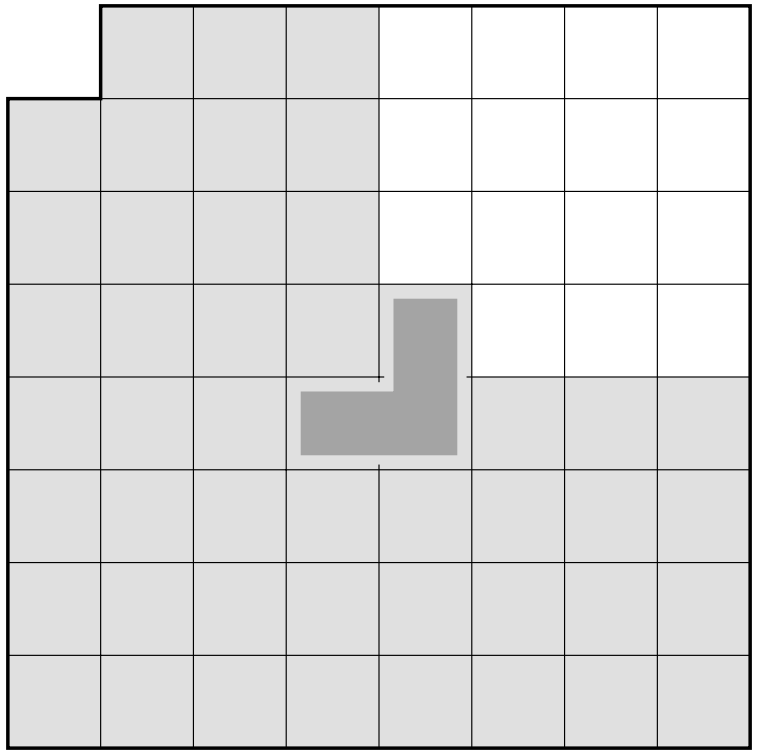
```
COVER(A)
1  place middle L
2  if A is 2 × 2
3        return
4  for each quadrant Q
5        COVER(Q)
```

A, Q = 



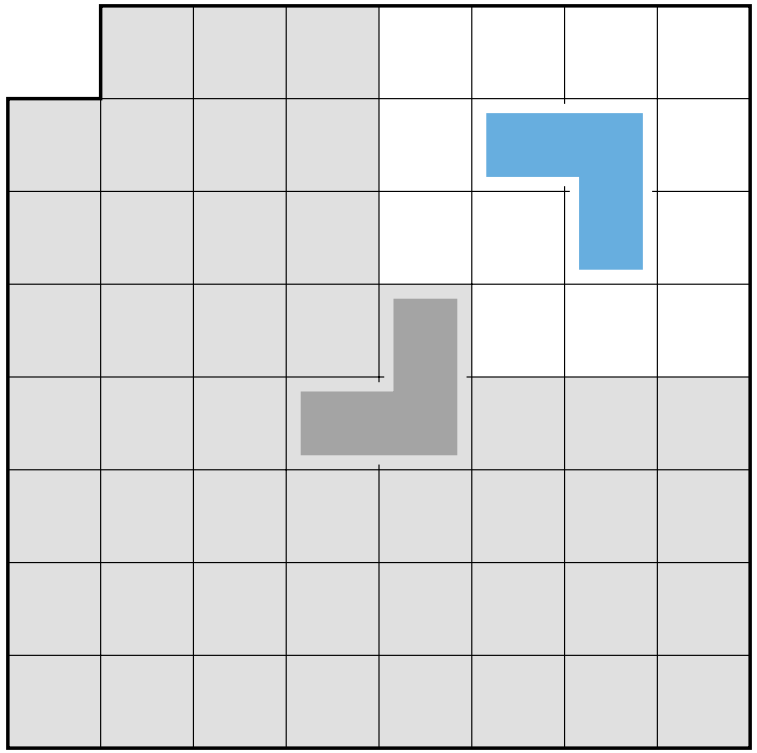
```
COVER(A)
1  place middle L
2  if A is 2 × 2
3        return
4  for each quadrant Q
5        COVER(Q)
```

A, Q = 



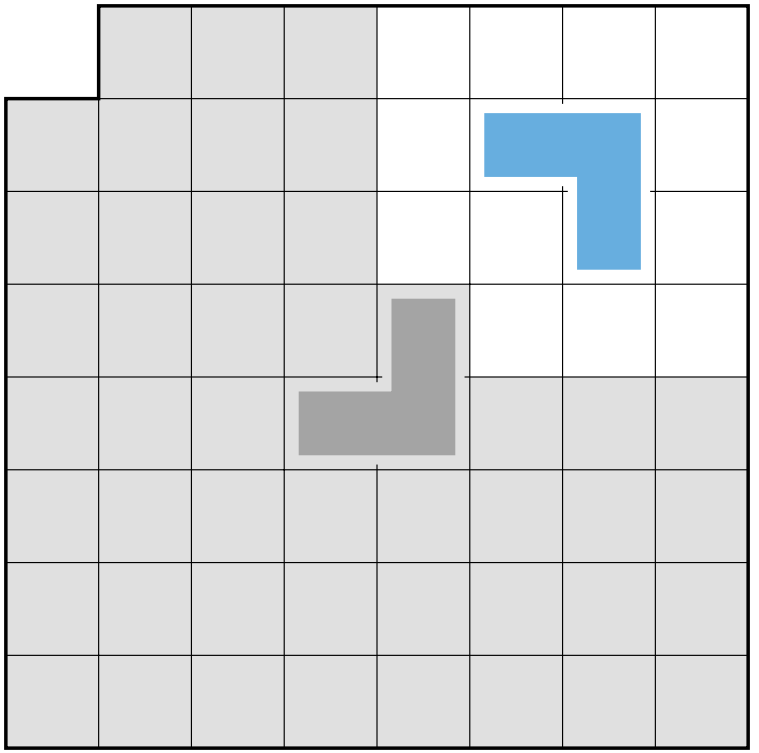
```
COVER(A)
1  place middle L
2  if A is 2 × 2
3        return
4  for each quadrant Q
5        COVER(Q)
```

A, Q = 



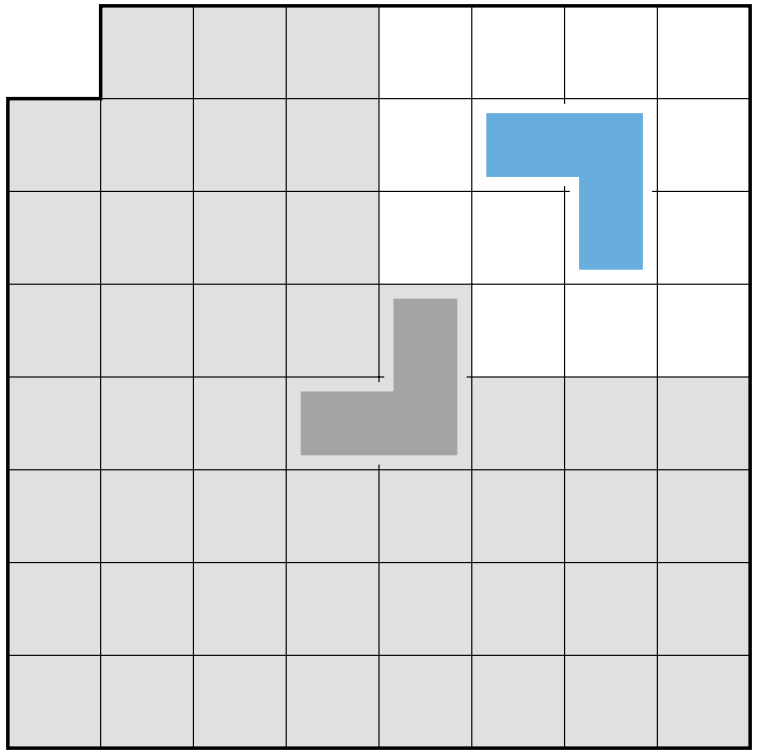
```
COVER(A)
1  place middle L
2  if A is 2 × 2
3        return
4  for each quadrant Q
5        COVER(Q)
```

A, Q = 



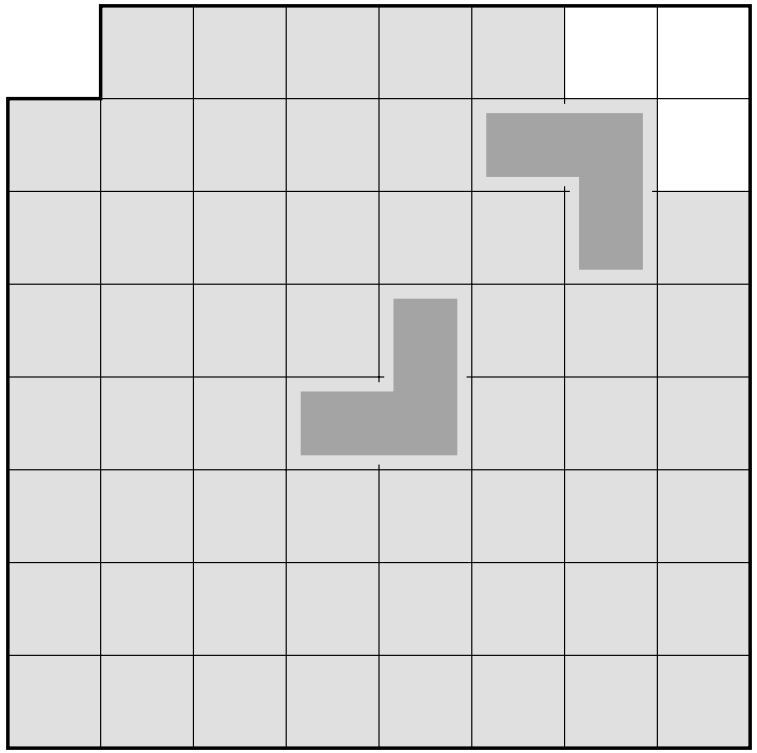
```
COVER(A)
1  place middle L
2  if A is 2 × 2
3        return
4  for each quadrant Q
5        COVER(Q)
```

A, Q =  ›



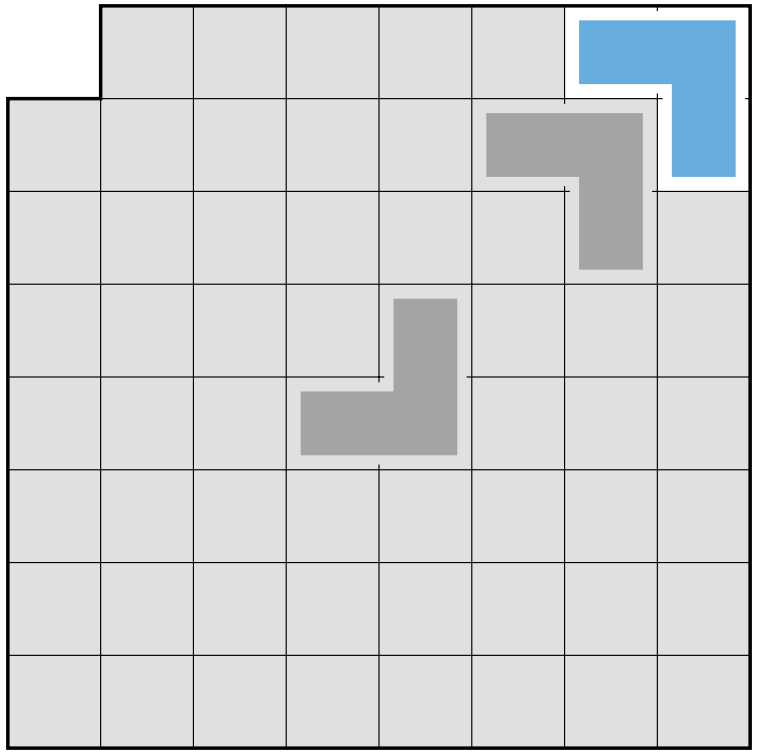
```
COVER(A)
1  place middle L
2  if A is 2 × 2
3        return
4  for each quadrant Q
5        COVER(Q)
```

A, Q = 



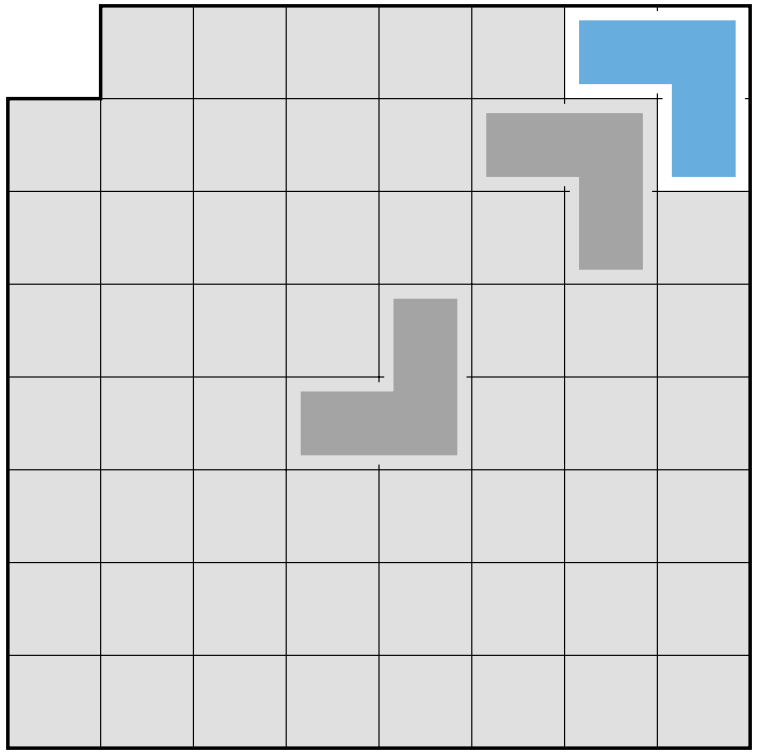
```
COVER(A)
1  place middle L
2  if A is 2 × 2
3        return
4  for each quadrant Q
5        COVER(Q)
```

A, Q = 



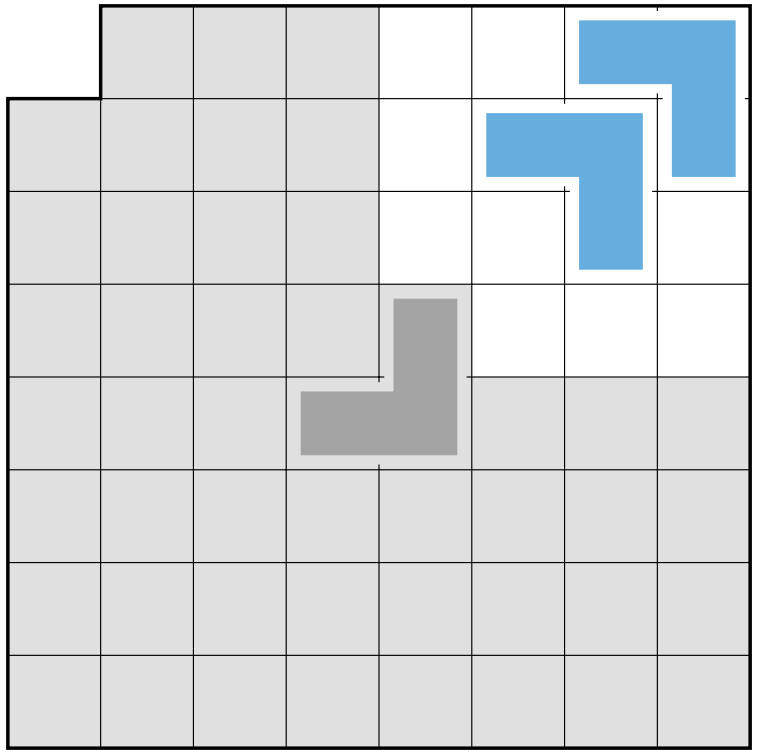
```
COVER(A)
1  place middle L
2  if A is 2 × 2
3        return
4  for each quadrant Q
5        COVER(Q)
```

A, Q = 



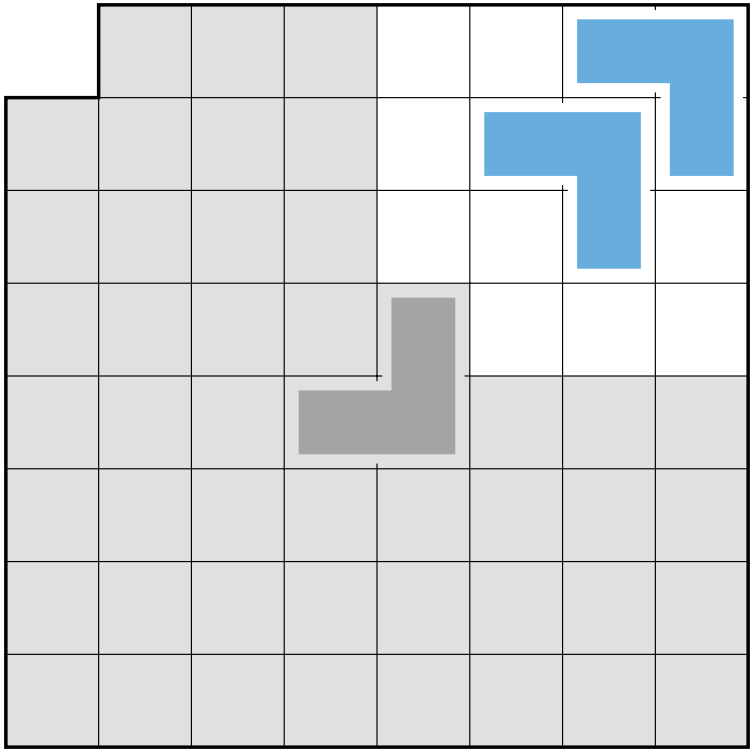
```
COVER(A)
1  place middle L
2  if A is 2 × 2
3        return
4  for each quadrant Q
5        COVER(Q)
```

A, Q = 



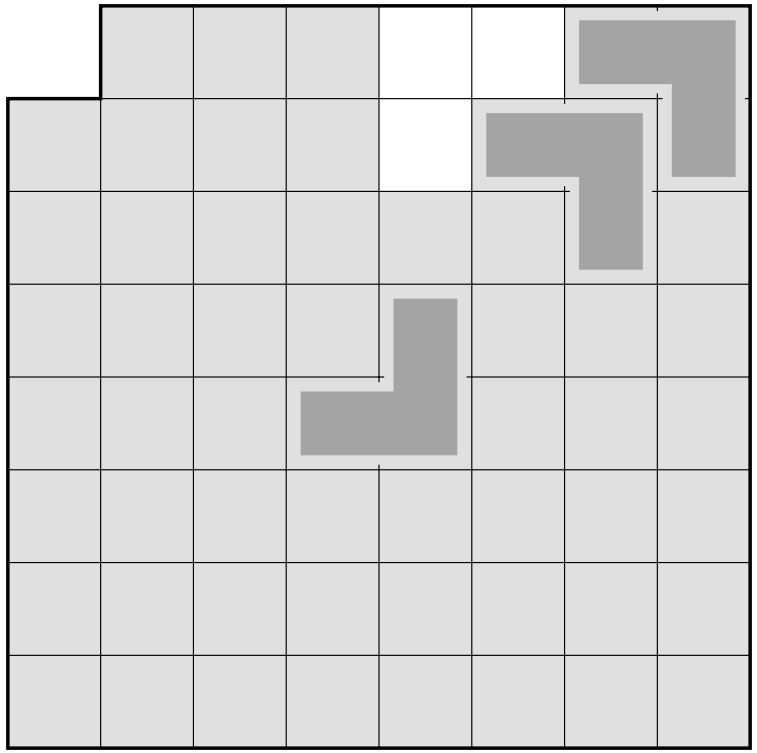
```
COVER(A)
1  place middle L
2  if A is 2 × 2
3        return
4  for each quadrant Q
5        COVER(Q)
```

A, Q = 



```
COVER(A)
1  place middle L
2  if A is 2 × 2
3        return
4  for each quadrant Q
5        COVER(Q)
```

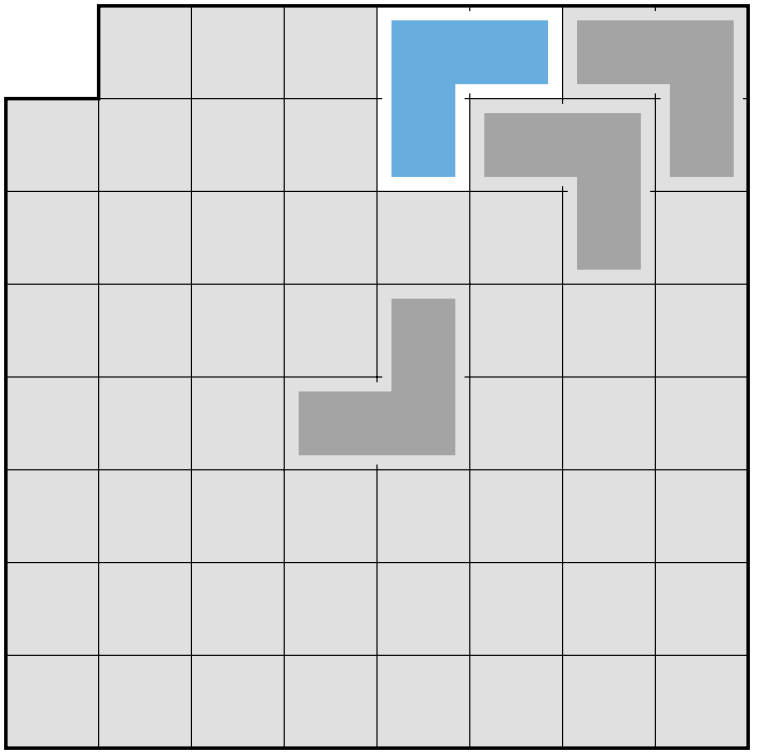
A, Q =



mlh@ntnu.no

```
COVER(A)
1  place middle L
2  if A is 2 × 2
3        return
4  for each quadrant Q
5        COVER(Q)
```

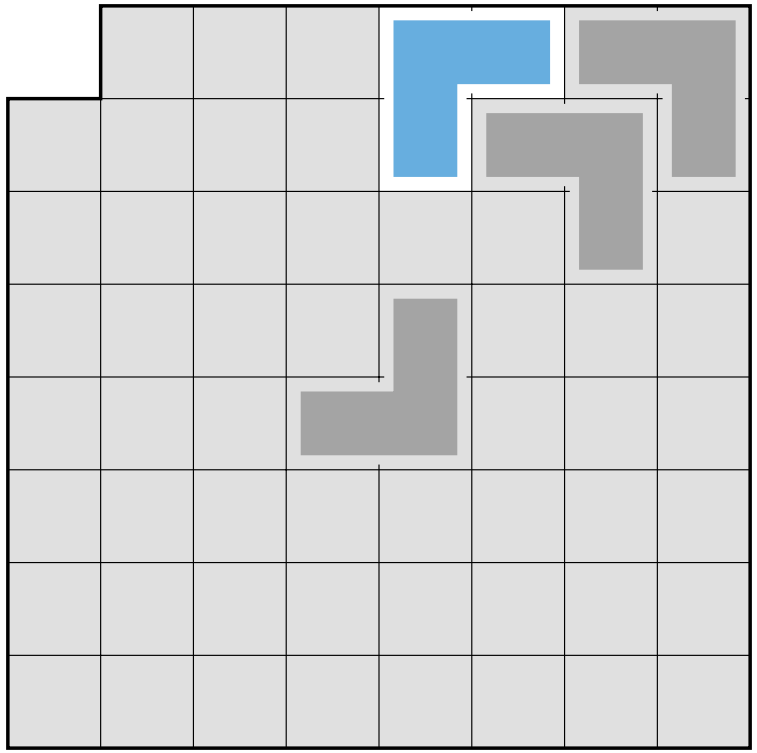
A, Q =



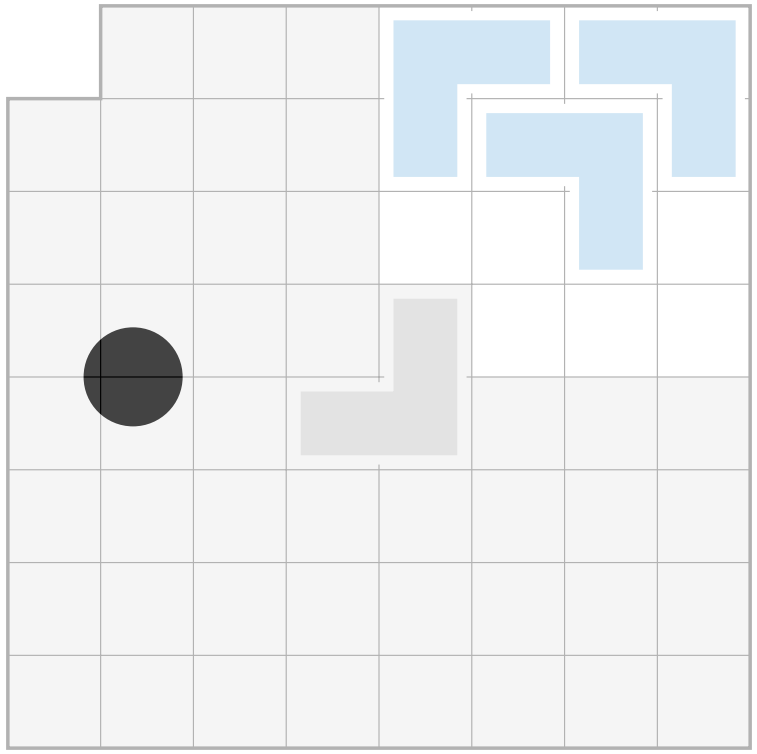
mlh@ntnu.no

```
COVER(A)
1  place middle L
2  if A is 2 × 2
3        return
4  for each quadrant Q
5        COVER(Q)
```

A, Q = 



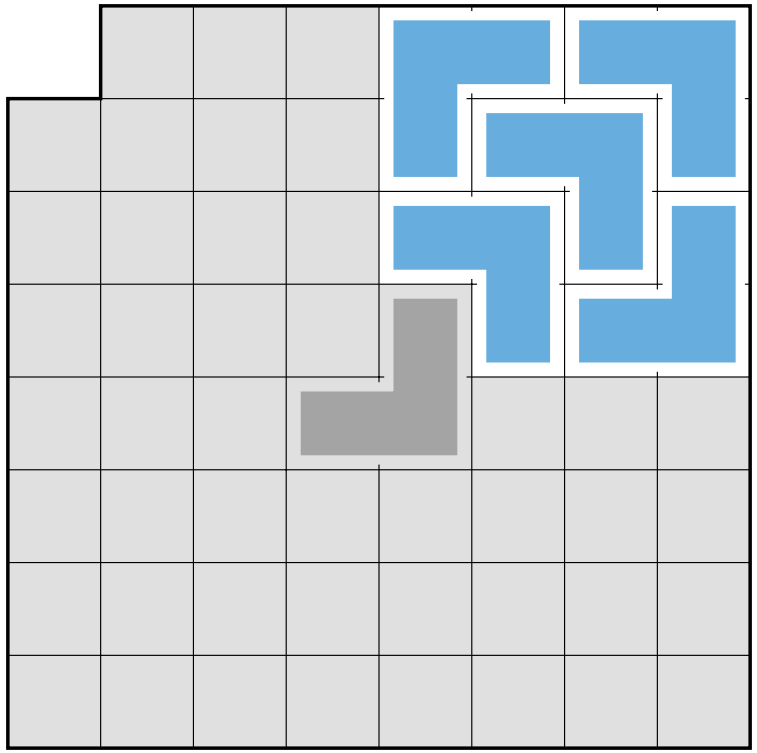
```
COVER(A)
1  place middle L
2  if A is 2 × 2
3        return
4  for each quadrant Q
5        COVER(Q)
```



A, Q = 

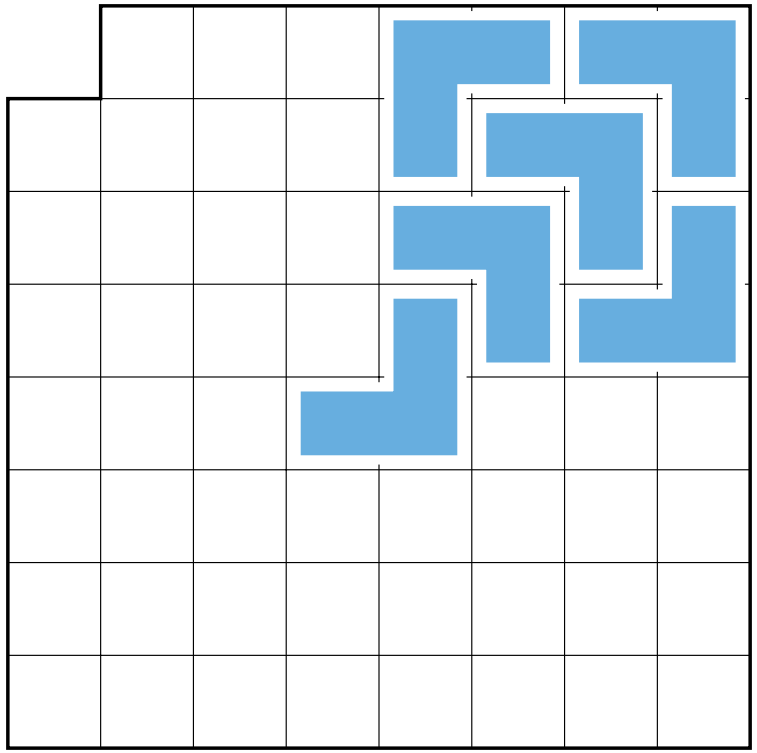
```
COVER(A)
1  place middle L
2  if A is 2 × 2
3        return
4  for each quadrant Q
5        COVER(Q)
```

A, Q = 



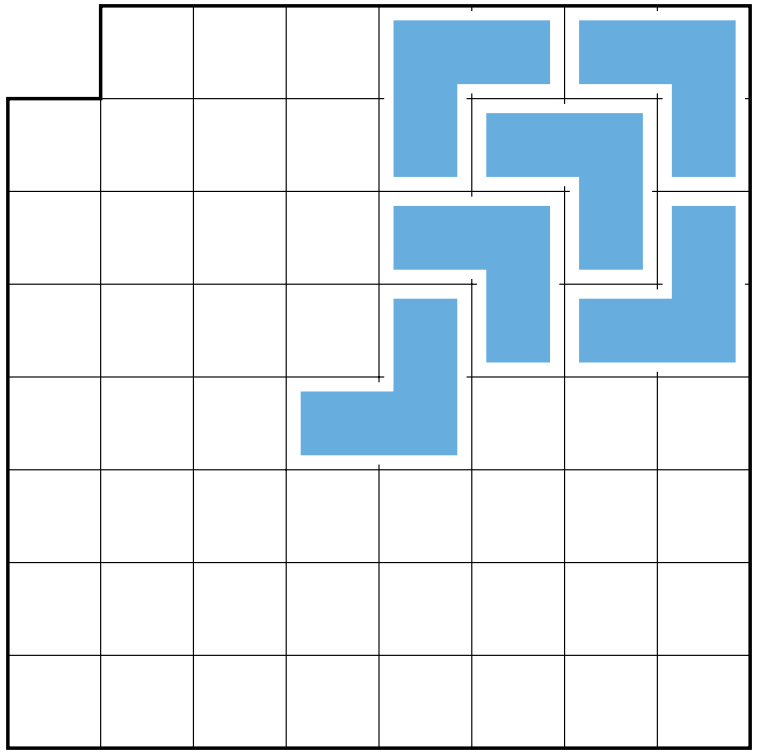
```
COVER(A)
1  place middle L
2  if A is 2 × 2
3        return
4  for each quadrant Q
5        COVER(Q)
```

A, Q = 

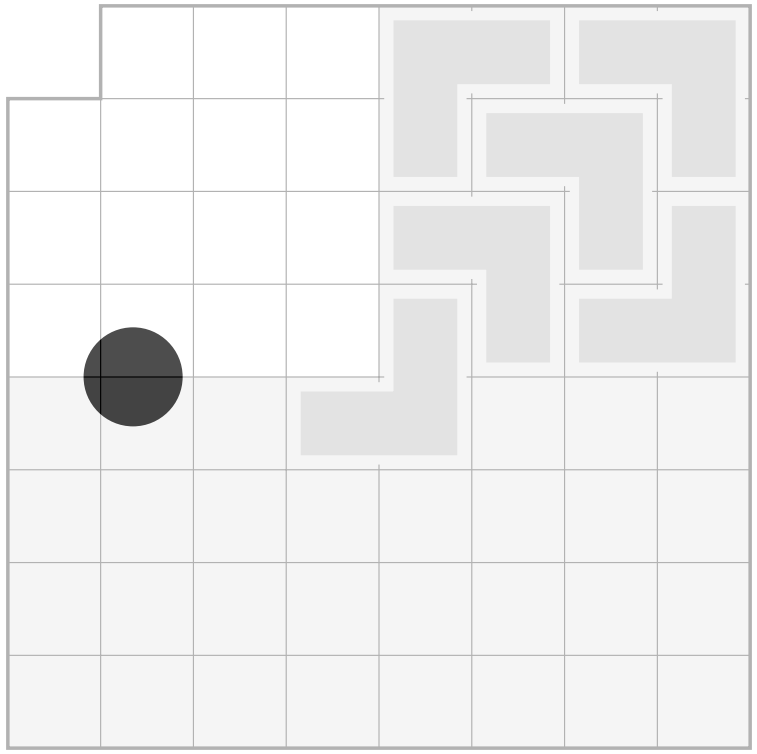


```
COVER(A)
1  place middle L
2  if A is 2 × 2
3        return
4  for each quadrant Q
5        COVER(Q)
```

A, Q = 



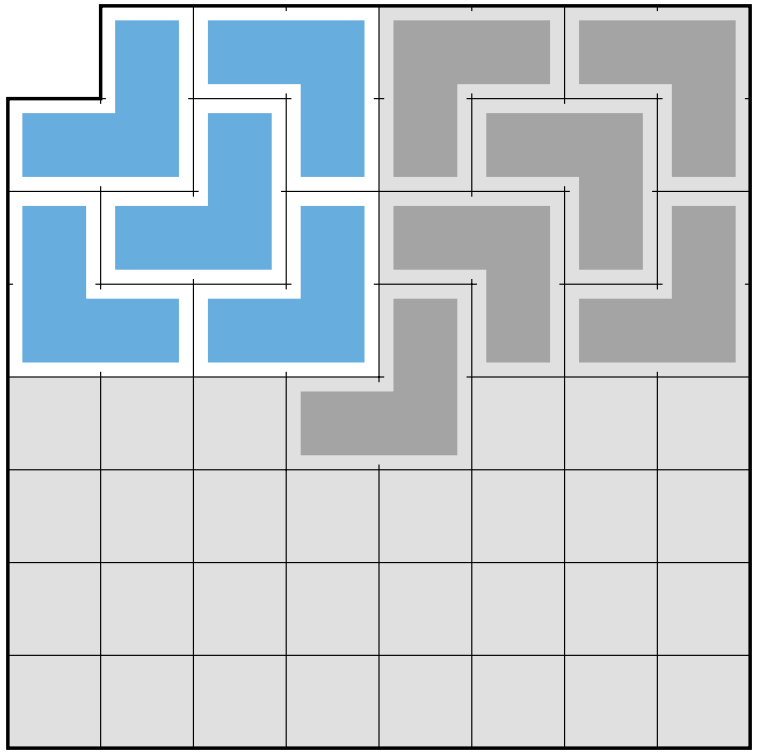
```
COVER(A)
1  place middle L
2  if A is 2 × 2
3        return
4  for each quadrant Q
5        COVER(Q)
```



A, Q = 

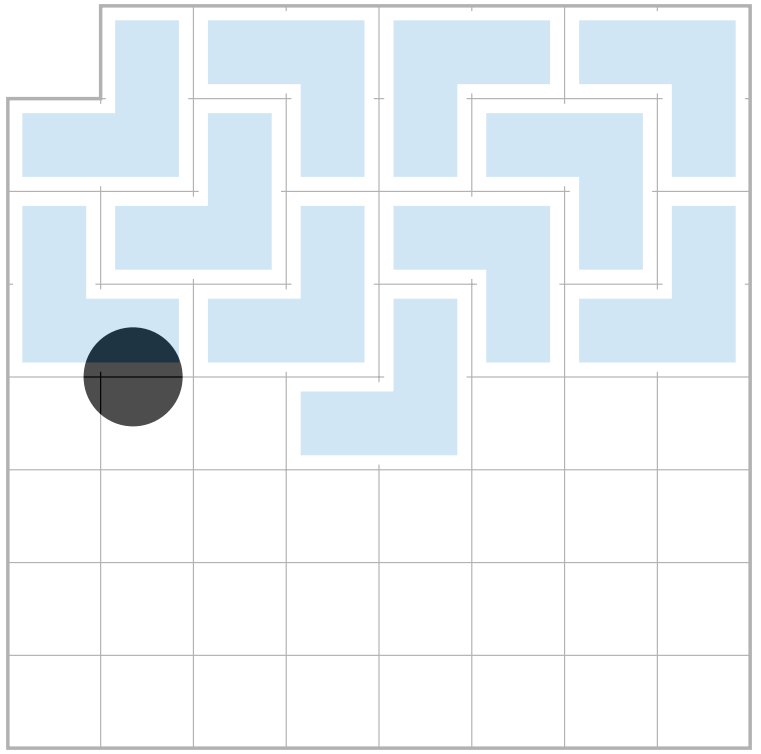
```
COVER(A)
1  place middle L
2  if A is 2 × 2
3        return
4  for each quadrant Q
5        COVER(Q)
```

A, Q =



mlh@ntnu.no

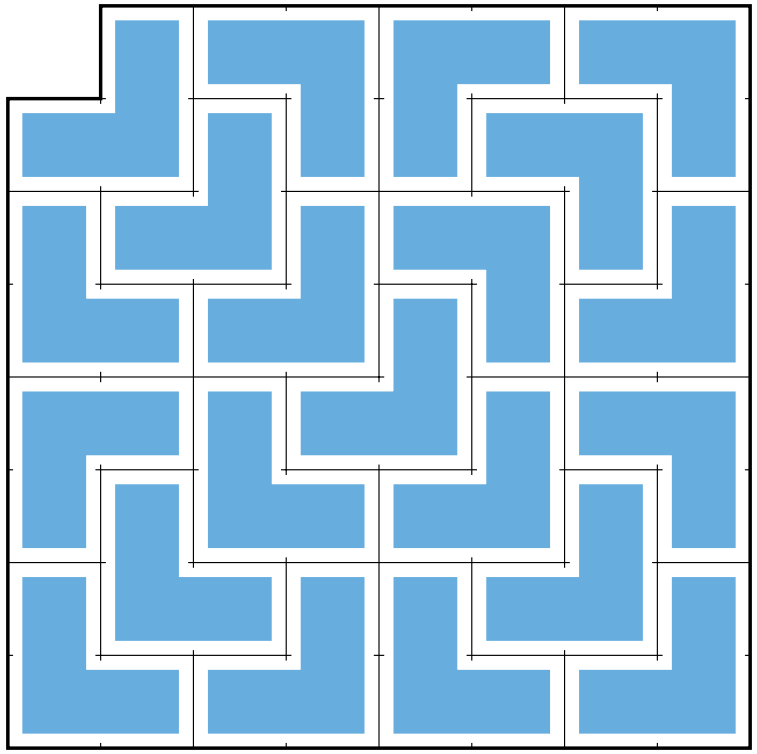
```
COVER(A)
1  place middle L
2  if A is 2 × 2
3        return
4  for each quadrant Q
5        COVER(Q)
```

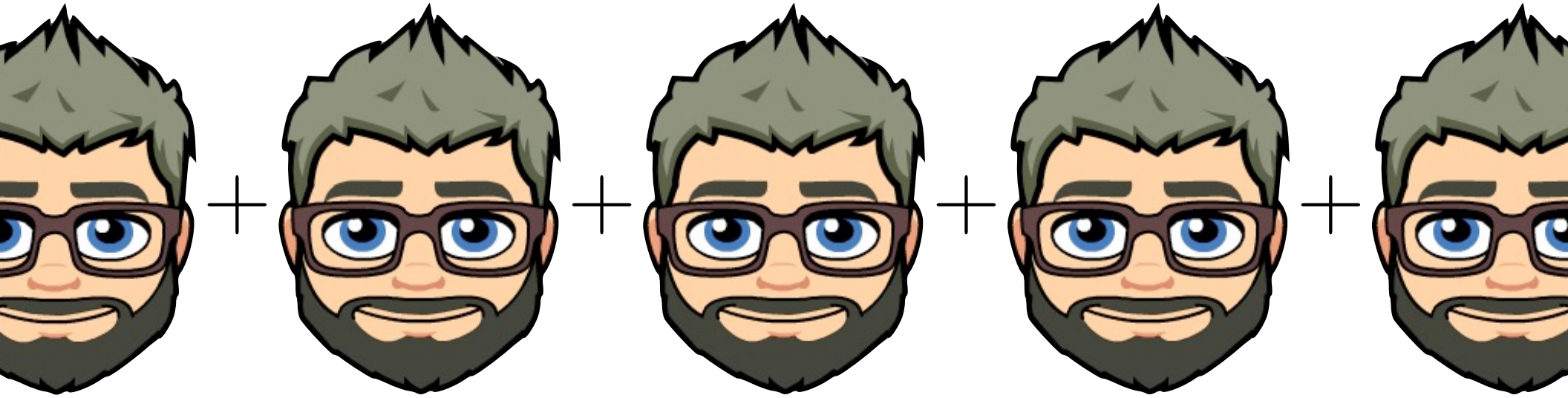


A, Q = 

```
COVER(A)
1  place middle L
2  if A is 2 × 2
3        return
4  for each quadrant Q
5        COVER(Q)
```

A, Q = 





Eksempel: Sum

Sum

Rekursiv

Dekomponering

- Vi vil summere elementene i en tabell
- Rekursjon: Summér alle unntatt siste
- Grunntilfelle: Tom sum er null
- Induktivt premiss: Summen er rett
- Induksjonstrinn: Legg til siste element

$\textsc{Sum}(\mathrm{A}, i)$

Summen av $\mathrm{A}[1\,..\,i]$

```
SUM(A, i)
1  if i < 1
```

Grunntilfelle

```
SUM(A, i)
1  if i < 1
2        return 0
```

Summen av en tom sekvens

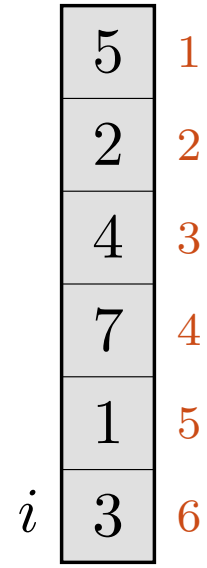
```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
```

Induksjonshypotese: SUM(A, $i-1$) er summen av A[1 . . $i-1$]

```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```

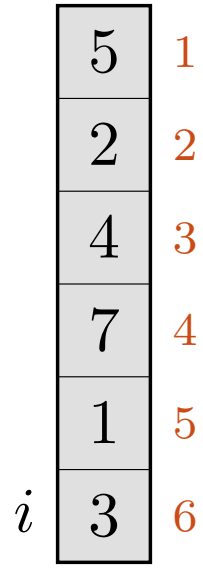
Induktivt trinn: Sørg for at SUM(A, i) er summen av A[1 . . i]

```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```



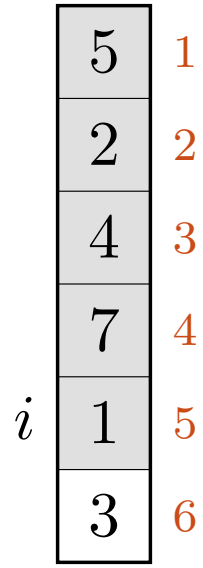
tmp = –

```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```



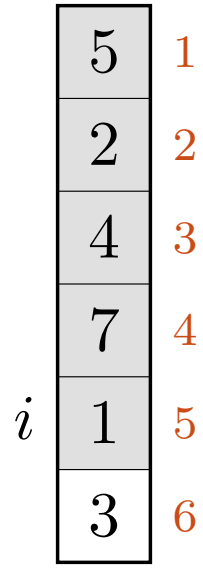
tmp = –

```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```



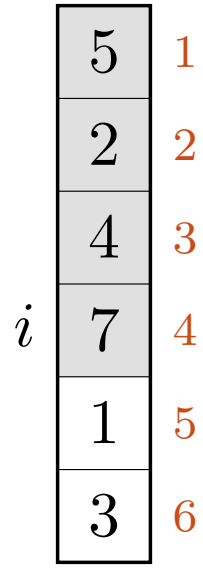
tmp = – › –

```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```



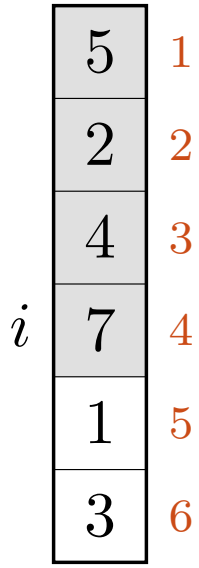
tmp = – › –

```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```



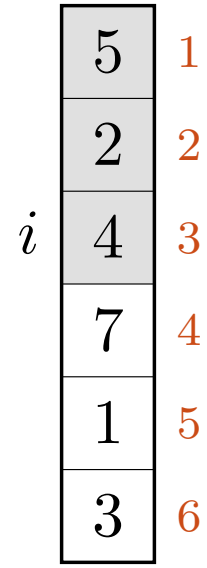
tmp = – › – › –

```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```



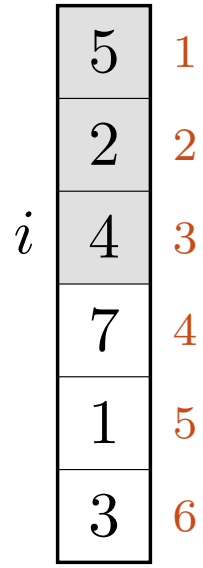
tmp = – › – › –

```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```



tmp = – › – › – › –

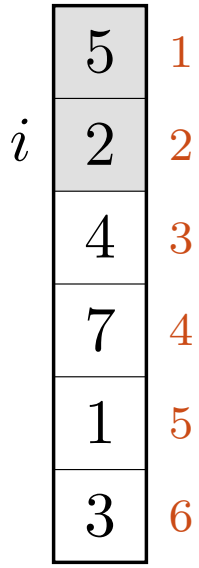
```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```



tmp = – › – › – › –

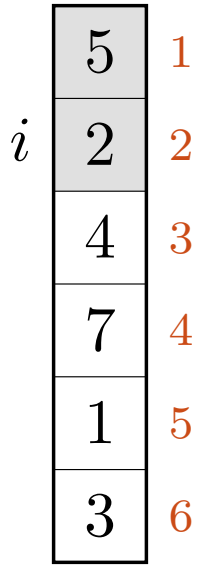
```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```

tmp = – › – › – › – › –

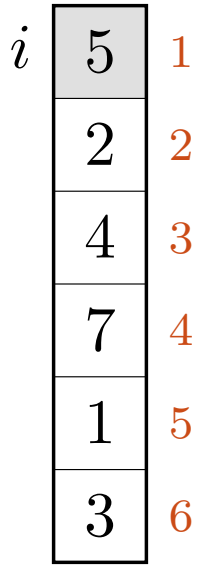


```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```

tmp = – › – › – › – › –

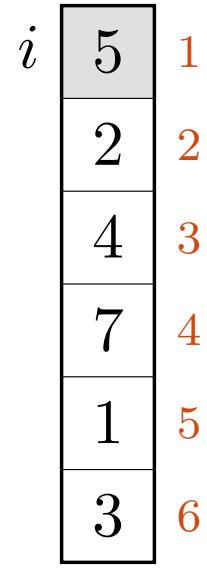


```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```



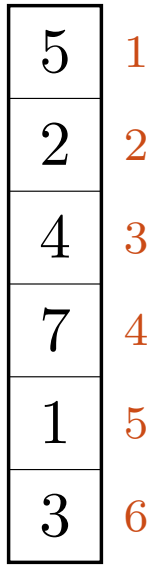
tmp = – › – › – › – › – › –

```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```



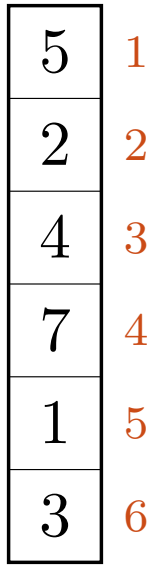
tmp = – › – › – › – › – › –

```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```



tmp = – › – › – › – › – › – › –

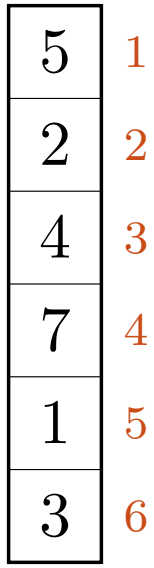
```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```



tmp = – › – › – › – › – › – › –

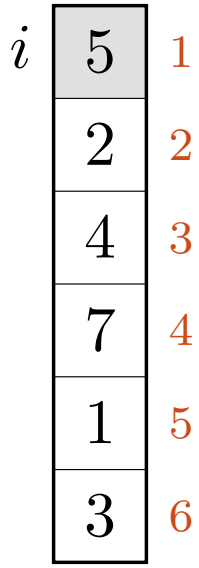
```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```

→ 0



tmp = – › – › – › – › – › – › –

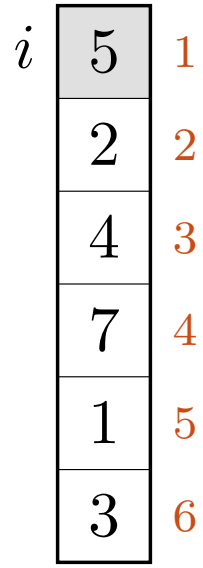
```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```



tmp = – › – › – › – › – › 0

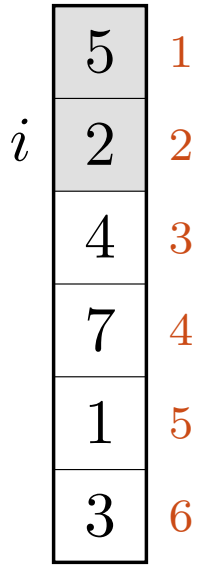
```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```

→ 5



tmp = – › – › – › – › – › 0

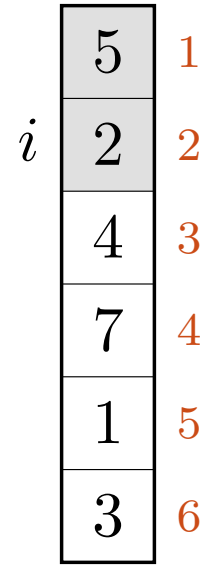
```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```



tmp = – › – › – › – › 5

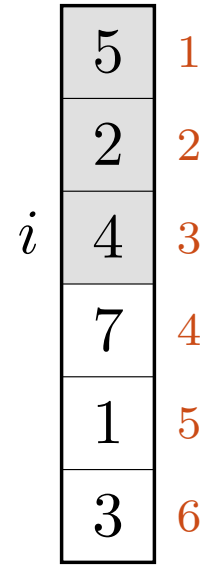
```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```

→ 7



tmp = – › – › – › – › 5

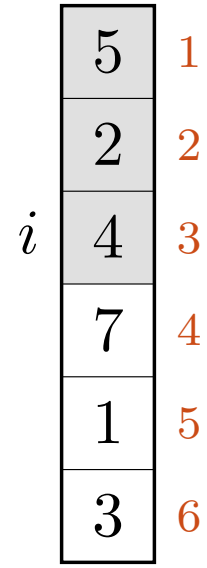
```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```



tmp = – › – › – › 7

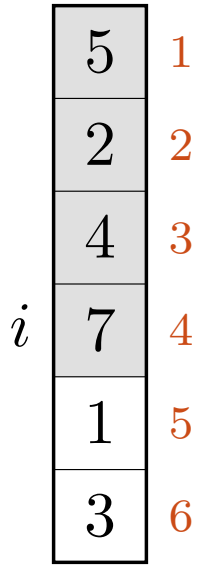
```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```

→ 11



tmp = – › – › – › 7

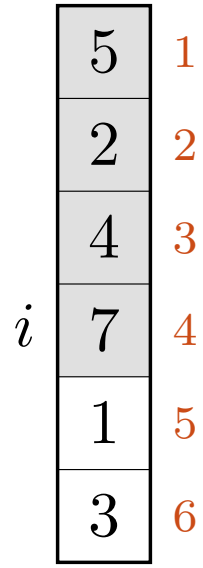
```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```



tmp = – › – › 11

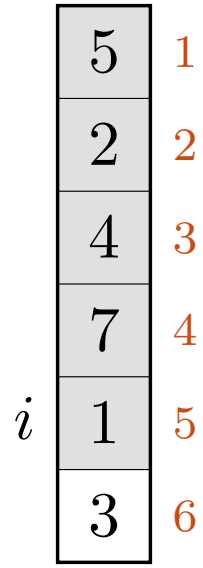
```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```

→ 18



tmp = – › – › 11

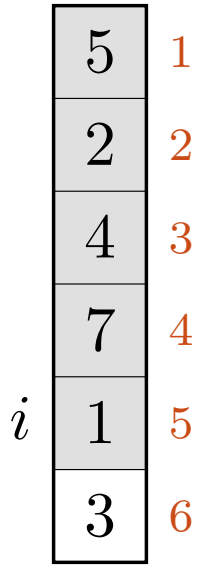
```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```



tmp = – › 18

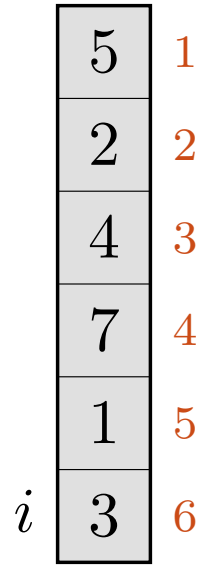
```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```

→ 19



tmp = – › 18

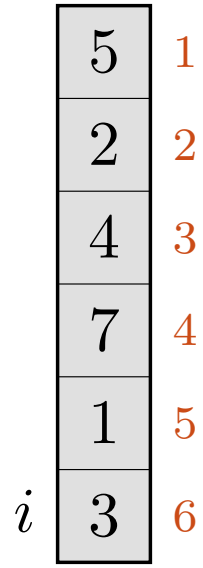
```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```



$tmp = 19$

```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```

→ 22



$tmp = 19$

Sum

Iterativ

Dekomponering

- **Invariant: Vi har summert rett så langt**
- **Initialisering: Tom sum er null**
- **Vedlikehold:**
  - **Induktivt premiss: Summen er rett før iterasjonen**
  - **Induksjonstrinn: Legg til neste element**
- **Terminering: Til slutt har vi summert alle**

SUM(A)

Vi vil summere elementene i A

```
SUM(A)
1  res = 0
```

Initialisering: Sum så langt er 0

```
SUM(A)
1  res = 0
2  for j = 1 to A.length
```

Induksjonshypotese: res er summen av $A[1..j-1]$

```
SUM(A)
1  res = 0
2  for j = 1 to A.length
3        res = res + A[j]
```

Induktivt trinn: Sørg for at *res* er summen av $A[1 \,..\, j]$

```
SUM(A)
1  res = 0
2  for j = 1 to A.length
3        res = res + A[j]
4  return res
```

Terminering: $j = n$, så res er summen av $A[1..n]$

```
SUM(A)
1  res = 0
2  for j = 1 to A.length
3        res = res + A[j]
4  return res
```

5	1
2	2
4	3
7	4
1	5
3	6

res = -

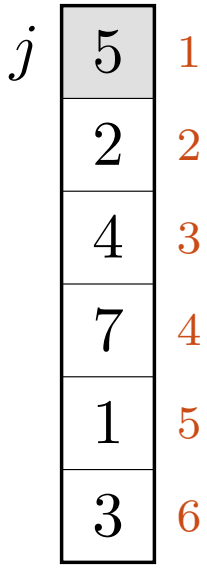
```
SUM(A)
1  res = 0
2  for j = 1 to A.length
3        res = res + A[j]
4  return res
```

5	1
2	2
4	3
7	4
1	5
3	6

res = 0

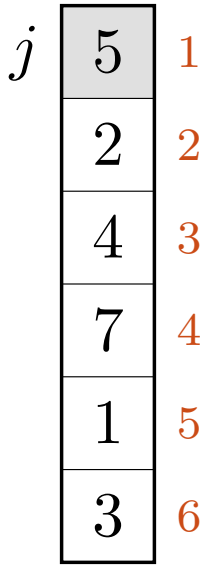
```
SUM(A)
1  res = 0
2  for j = 1 to A.length
3        res = res + A[j]
4  return res
```

$res = 0$

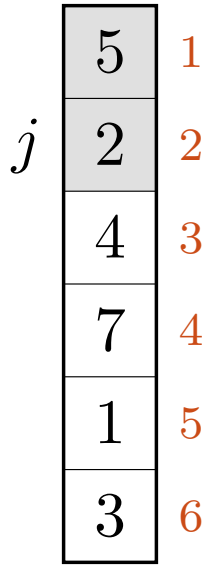


```
SUM(A)
1  res = 0
2  for j = 1 to A.length
3        res = res + A[j]
4  return res
```

$res = 5$

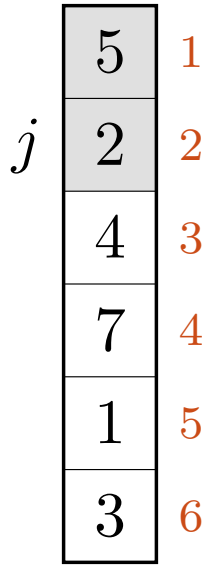


```
SUM(A)
1  res = 0
2  for j = 1 to A.length
3        res = res + A[j]
4  return res
```



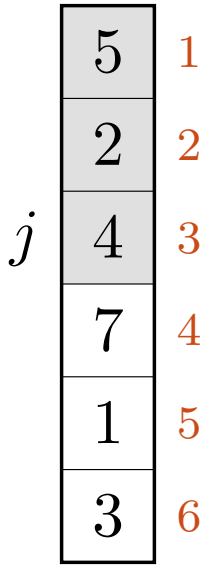
$res = 5$

```
SUM(A)
1  res = 0
2  for j = 1 to A.length
3        res = res + A[j]
4  return res
```



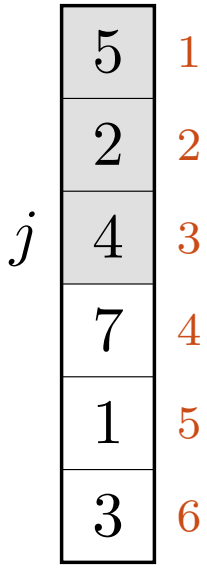
res = 7

```
SUM(A)
1  res = 0
2  for j = 1 to A.length
3        res = res + A[j]
4  return res
```



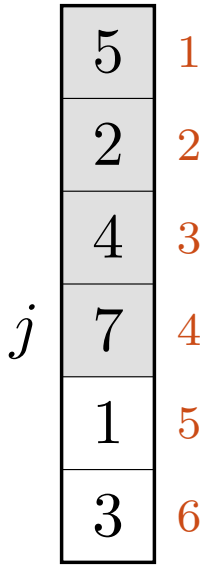
$res = 7$

```
SUM(A)
1  res = 0
2  for j = 1 to A.length
3        res = res + A[j]
4  return res
```



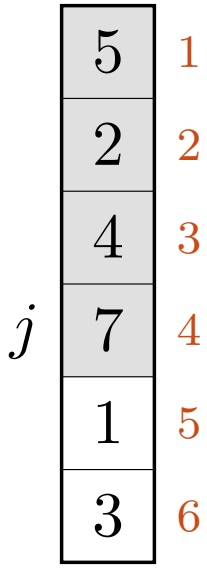
res = 11

```
SUM(A)
1  res = 0
2  for j = 1 to A.length
3        res = res + A[j]
4  return res
```



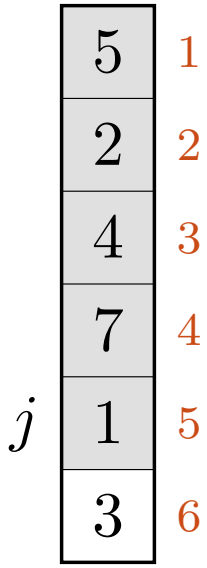
res = 11

```
SUM(A)
1  res = 0
2  for j = 1 to A.length
3        res = res + A[j]
4  return res
```



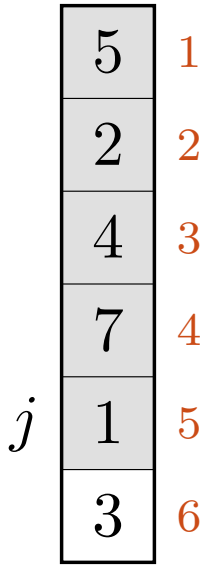
res = 18

```
SUM(A)
1  res = 0
2  for j = 1 to A.length
3        res = res + A[j]
4  return res
```



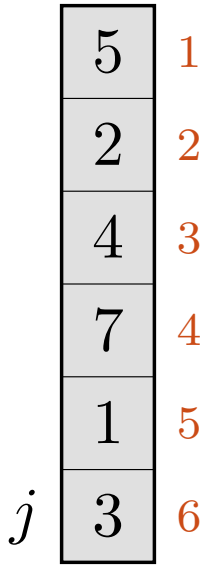
res = 18

```
SUM(A)
1  res = 0
2  for j = 1 to A.length
3        res = res + A[j]
4  return res
```



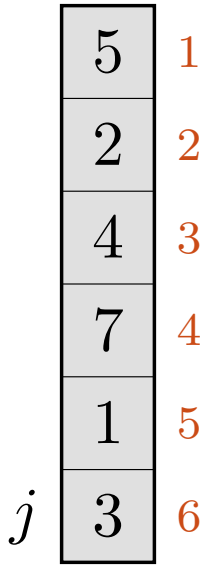
$res = 19$

```
SUM(A)
1  res = 0
2  for j = 1 to A.length
3        res = res + A[j]
4  return res
```



$res = 19$

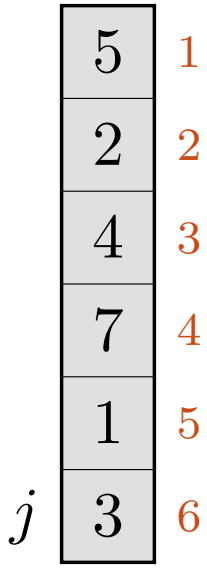
```
SUM(A)
1  res = 0
2  for j = 1 to A.length
3        res = res + A[j]
4  return res
```



$res = 22$

```
SUM(A)
1  res = 0
2  for j = 1 to A.length
3        res = res + A[j]
4  return res
```

→ 22



res = 22

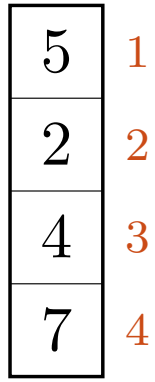
Rek. vs. Iter.

Et spørsmål om perspektiv

Rekursjon og iterasjon er i all hovedsak ekvivalente ting. Her har vi en sammenligning av hvordan de to variantene oppfører seg; for begge to er det induktive premisset at den grå biten er summert allerede. I den rekursive varianten gjør vi det rekursivt før vi legger til det siste elementet. I den iterative varianten har vi allerede gjort det iterativt når vi skal legge til det siste elementet. Men … det er jo nesten samme sak, da.

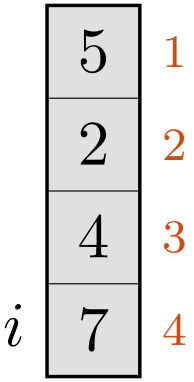
```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```

```
SUM(A)
1  res = 0
2  for j = 1 to A.length
3        res = res + A[j]
4  return res
```



```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```

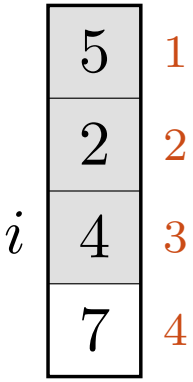
```
SUM(A)
1  res = 0
2  for j = 1 to A.length
3        res = res + A[j]
4  return res
```



tmp = –

```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```

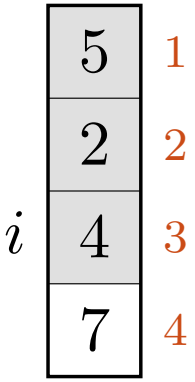
```
SUM(A)
1  res = 0
2  for j = 1 to A.length
3        res = res + A[j]
4  return res
```



tmp = – › –

```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```

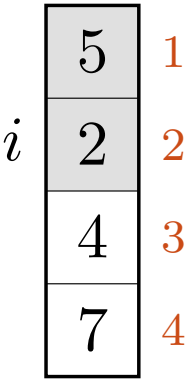
```
SUM(A)
1  res = 0
2  for j = 1 to A.length
3        res = res + A[j]
4  return res
```



tmp = – › –

```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```

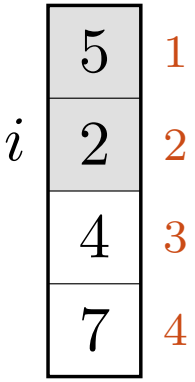
```
SUM(A)
1  res = 0
2  for j = 1 to A.length
3        res = res + A[j]
4  return res
```



tmp = – › – › –

```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```

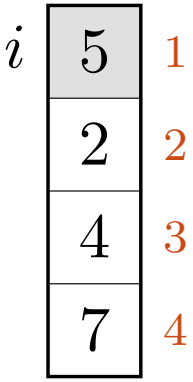
```
SUM(A)
1  res = 0
2  for j = 1 to A.length
3        res = res + A[j]
4  return res
```



tmp = – › – › –

```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```

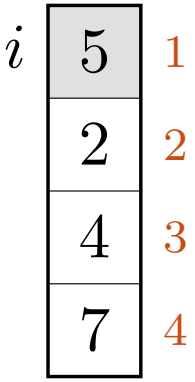
```
SUM(A)
1  res = 0
2  for j = 1 to A.length
3        res = res + A[j]
4  return res
```



tmp = – › – › – › –

```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```

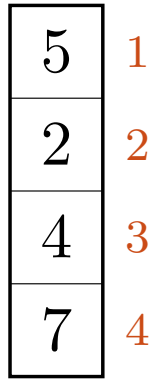
```
SUM(A)
1  res = 0
2  for j = 1 to A.length
3        res = res + A[j]
4  return res
```



tmp = – › – › – › –

```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```

```
SUM(A)
1  res = 0
2  for j = 1 to A.length
3        res = res + A[j]
4  return res
```



tmp = – › – › – › – › –

```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```

tmp = – › – › – › – › –

```
SUM(A)
1  res = 0
2  for j = 1 to A.length
3        res = res + A[j]
4  return res
```

res = 0

5	1
2	2
4	3
7	4

```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```

→ 0

tmp = – › – › – › – › –

```
SUM(A)
1  res = 0
2  for j = 1 to A.length
3        res = res + A[j]
4  return res
```

res = 0

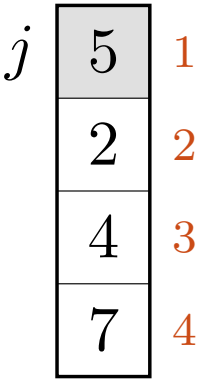
j	
1	5
2	2
3	4
4	7

```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```

tmp = – › – › – › 0

```
SUM(A)
1  res = 0
2  for j = 1 to A.length
3        res = res + A[j]
4  return res
```

res = 5



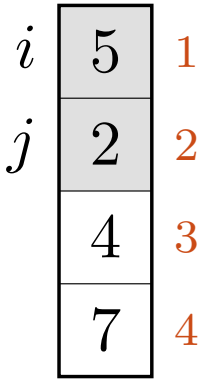
```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```

→ 5

tmp = – › – › – › 0

```
SUM(A)
1  res = 0
2  for j = 1 to A.length
3        res = res + A[j]
4  return res
```

res = 5

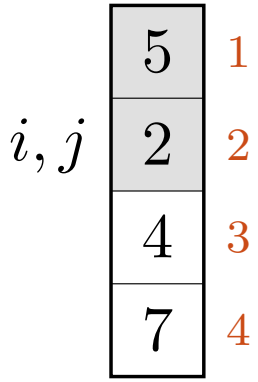


```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```

tmp = – › – › 5

```
SUM(A)
1  res = 0
2  for j = 1 to A.length
3        res = res + A[j]
4  return res
```

res = 7



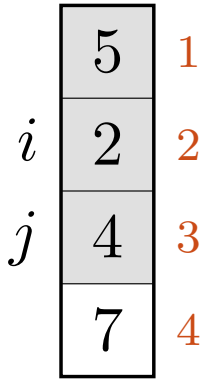
```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```

→ 7

tmp = – › – › 5

```
SUM(A)
1  res = 0
2  for j = 1 to A.length
3        res = res + A[j]
4  return res
```

res = 7

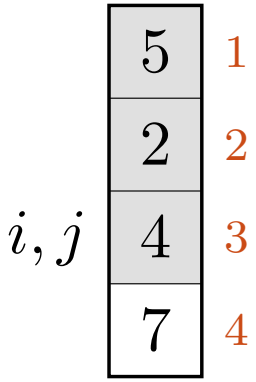


```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```

tmp = → 7

```
SUM(A)
1  res = 0
2  for j = 1 to A.length
3        res = res + A[j]
4  return res
```

res = 11



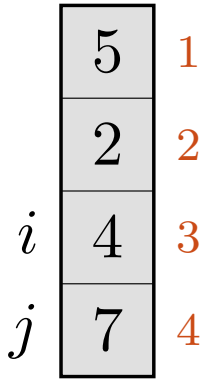
```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```

→ 11

tmp = – › 7

```
SUM(A)
1  res = 0
2  for j = 1 to A.length
3        res = res + A[j]
4  return res
```

res = 11

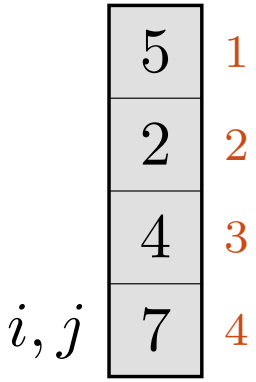


```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```

tmp = 11

```
SUM(A)
1  res = 0
2  for j = 1 to A.length
3        res = res + A[j]
4  return res
```

res = 18



```
SUM(A, i)
1  if i < 1
2        return 0
3  tmp = SUM(A, i − 1)
4  return tmp + A[i]
```

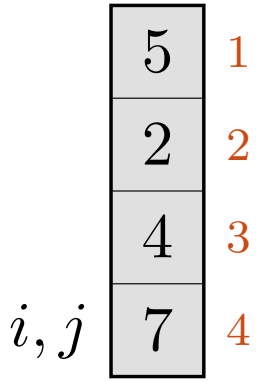
→ 18

tmp = 11

```
SUM(A)
1  res = 0
2  for j = 1 to A.length
3        res = res + A[j]
4  return res
```

→ 18

res = 18

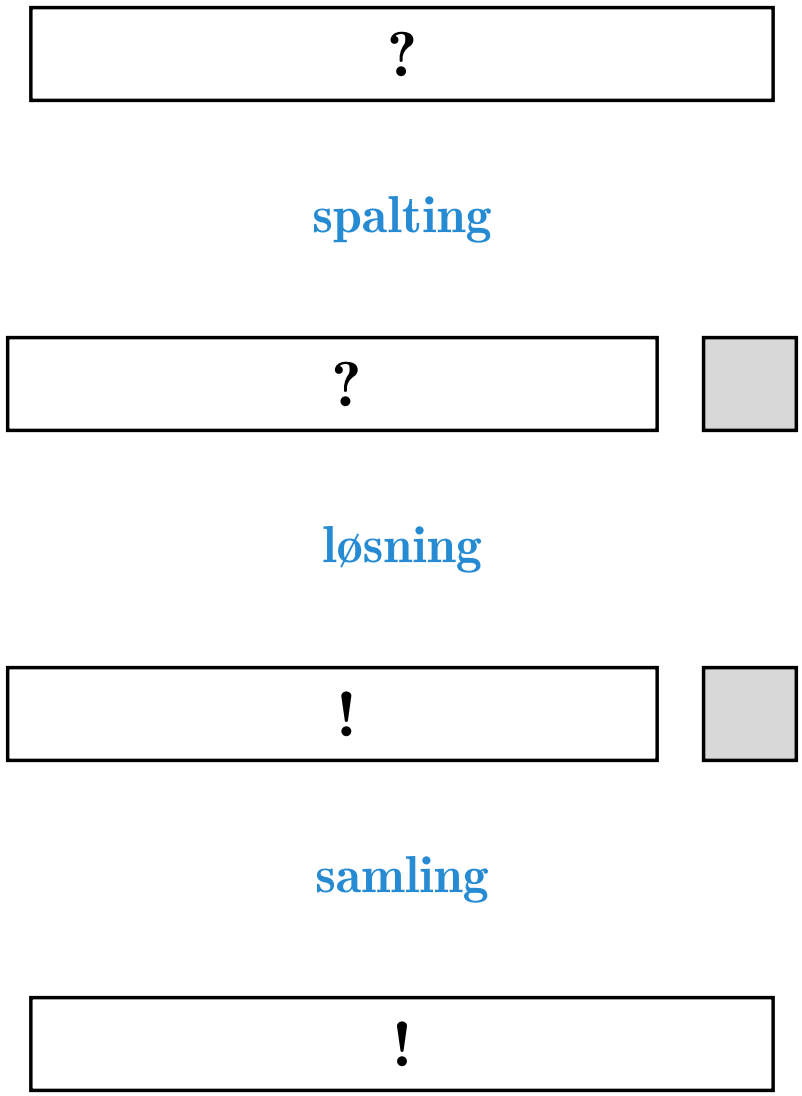


Noen vanlige dekomponeringer

(*i*) Splitt av siste element

(*ii*) Løs rekursivt for resten

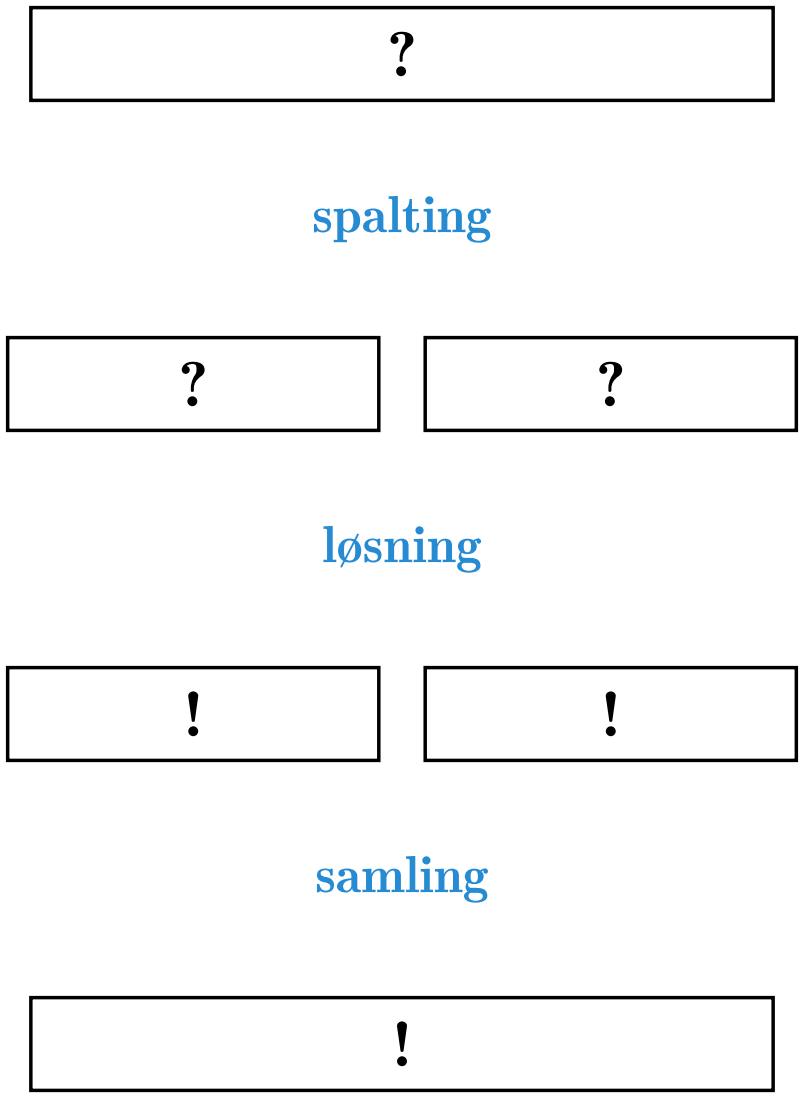
(*iii*) Sett inn siste element



(*i*) Del f.eks. på midten

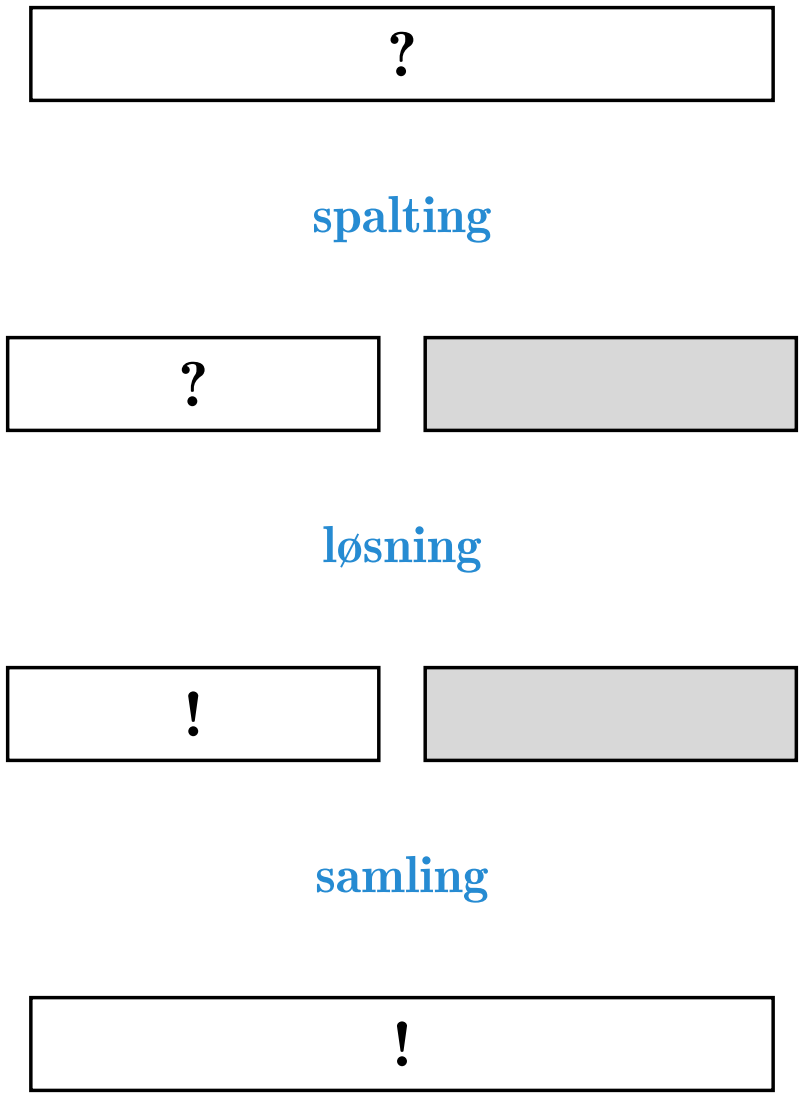
(*ii*) Løs halvdeler rekursivt

(*iii*) Flett dem sammen



(*i*) Del f.eks. på midten

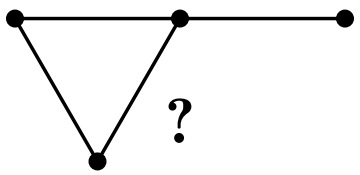
(*ii*) Løs én halvdel rekursivt



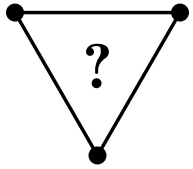
(*i*) Splitt av siste node

(*ii*) Løs rekursivt for resten

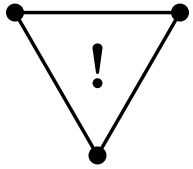
(*iii*) Sett inn siste node



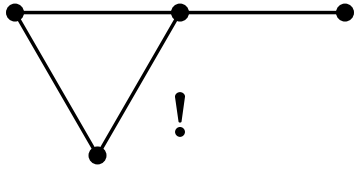
spalting



løsning



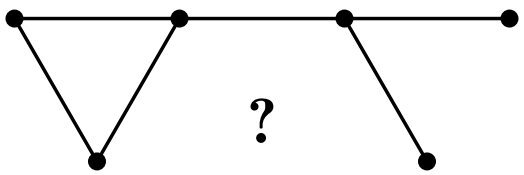
samling



(*i*) Fjern en kant

(*ii*) Løs hver del rekursivt

(*iii*) Sett inn kanten



spalting

løsning

samling