

# Algoritmer og Datastrukturer 1

Gerth Stølting Brodal

Amortiseret Analyse [CLRS, kapitel 17]

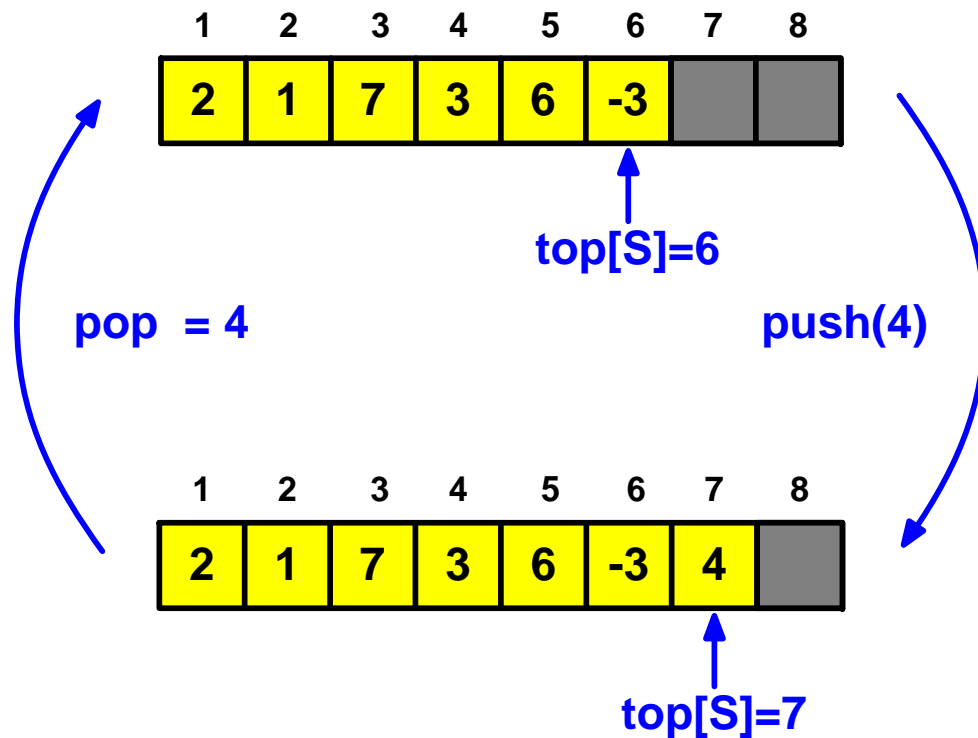


AARHUS UNIVERSITET



**Stak**

# Stak : Array Implementation



STACK-EMPTY( $S$ )

```
1  if  $top[S] = 0$ 
2    then return TRUE
3    else return FALSE
```

PUSH( $S, x$ )

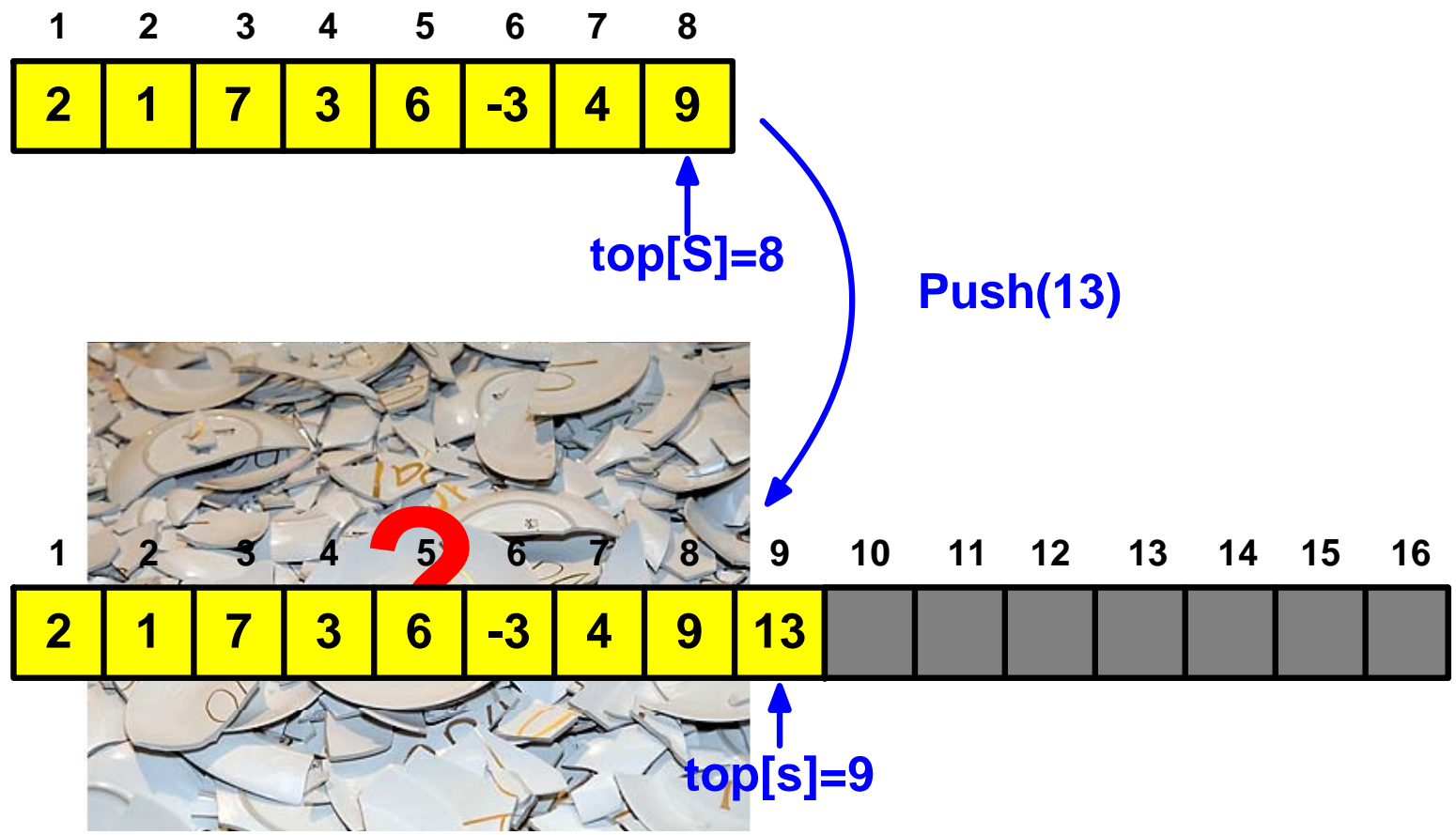
```
1   $top[S] \leftarrow top[S] + 1$ 
2   $S[top[S]] \leftarrow x$ 
```

POP( $S$ )

```
1  if STACK-EMPTY( $S$ )
2    then error "underflow"
3    else  $top[S] \leftarrow top[S] - 1$ 
4          return  $S[top[S] + 1]$ 
```

Stack-Empty, Push, Pop :  $O(1)$  tid

# Stak : Overløb



Array fordobling :  $O(n)$  tid

# Array Fordobling

**Fordoble** arrayet når det er fuld

1

2

4

8

**Tid** for  $n$  udvidelser:

16  $1+2+4+\dots+n/2+n = O(n)$

32

**Halver** arrayet når det er  $<1/4$  fyldt

32

16

16

8

**Tid** for  $n$  udvidelser/reduktioner:

8

16

$O(n)$

# Array Fordobling + Halvering

– en generel teknik

**Tid** for  $n$  udvidelser/reduktioner er  $O(n)$

**Plads**  $\leq 4 \cdot$  aktuelle antal elementer

**Array implementation af Stak:**  
 $n$  push og pop operationer tager  $O(n)$  tid

# Analyse teknik ønskes...

## Krav

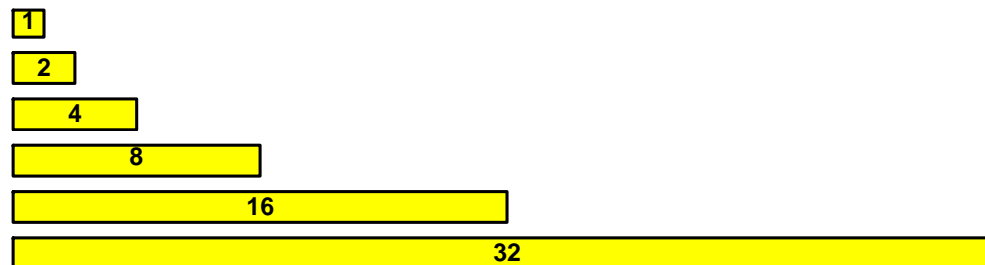
- Analysere **worst-case** tiden for en **sekvens** af operationer
- Behøver kun at analysere den **enkelte operation**

## Fordel

- Behøver **ikke** overveje andre operationer i sekvens og deres **indbyrdes påvirkninger**
- Gælder for alle sekvenser med de givne operationer

# Intuition

- Der findes ”**gode**”/”**balancerede**” tilstande og ”**dårlige**”/”**ubalancerede**”
- At komme fra en ”**dårlig**” tilstand til en ”**god**” tilstand er **dyrt**
- Det tager mange operationer fra en ”**god**” tilstand før man er i en ”**dårlig**”
- For de (mange) **billige** operationer ”betaler” vi lidt ekstra for senere at kunne lave en **dyr** operation næsten **gratis**



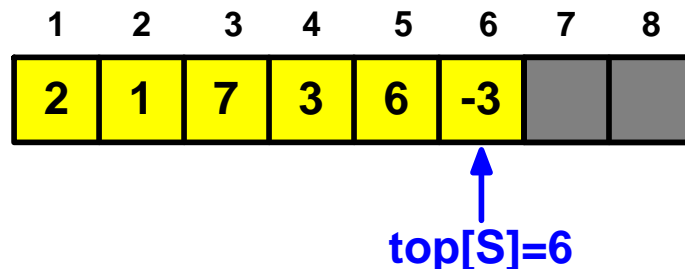


# Amortiseret Analyse

- **1 €** kan betale for  $O(1)$  **arbejde**
- En operation der tager tid  $O(t)$  koster  $t$  €
- Hvornår vi betaler/sparer op er ligegyldigt – bare pengene er der når vi skal bruge dem!
- **Opsparing = Potentiale =  $\Phi$**
- Vi kan ikke låne penge, dvs. vi skal spare op før vi bruger pengene,  **$\Phi \geq 0$**
- **Amortiseret tid** for en operation = hvad vi er **villige** til at betale – men vi skal have råd til operationen!
- Brug **invarianter** til at beskrive sammenhængen mellem **opsparingen** og **datastrukturen**

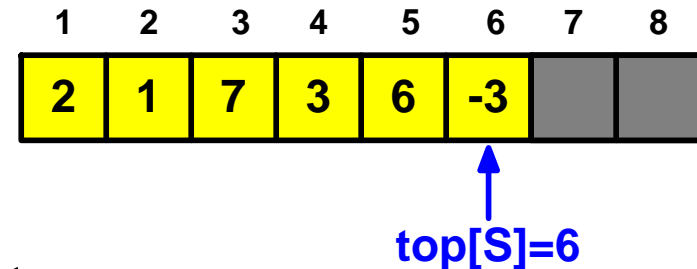
# Eksempel: Stak

- En **god** stak er halv fuld – kræver ingen opsparing
  - Invariant :  $\Phi = 2 \cdot | \text{top}[S] - |S|/2 |$
  - Antag: 1 € per element indsættelse/kopiering
  - Amortiseret tid per push: 3 € ?
- (har vi altid penge til at udføre operationen?)
- Hvis ja:  $n$  push operationer koster  $\leq 3n$  €



# Eksempel: Stak

## Push = Amortiseret 3€



- Push uden kopiering:
  - Et nyt element : **1 €**
  - $\|S\|/2 - \text{top}[S]$  vokser med højst 1, så invarianten holder hvis vi sparer **2 €** op
  - Amortiseret tid:  $1+2=3$
- Push med kopiering
  - Kopier  $S$ :  **$|S|$  €**
  - Indsæt nye element: **1 €**
  - $\Phi$  før =  $|S|$ ,  $\Phi$  efter = 2, dvs  **$|S|-2$  € frigives**
  - Amortiseret tid  $|S|+1-(|S|-2)=3$

Invariant:  $\Phi = 2 \cdot | \text{top}[S] - |S|/2 |$

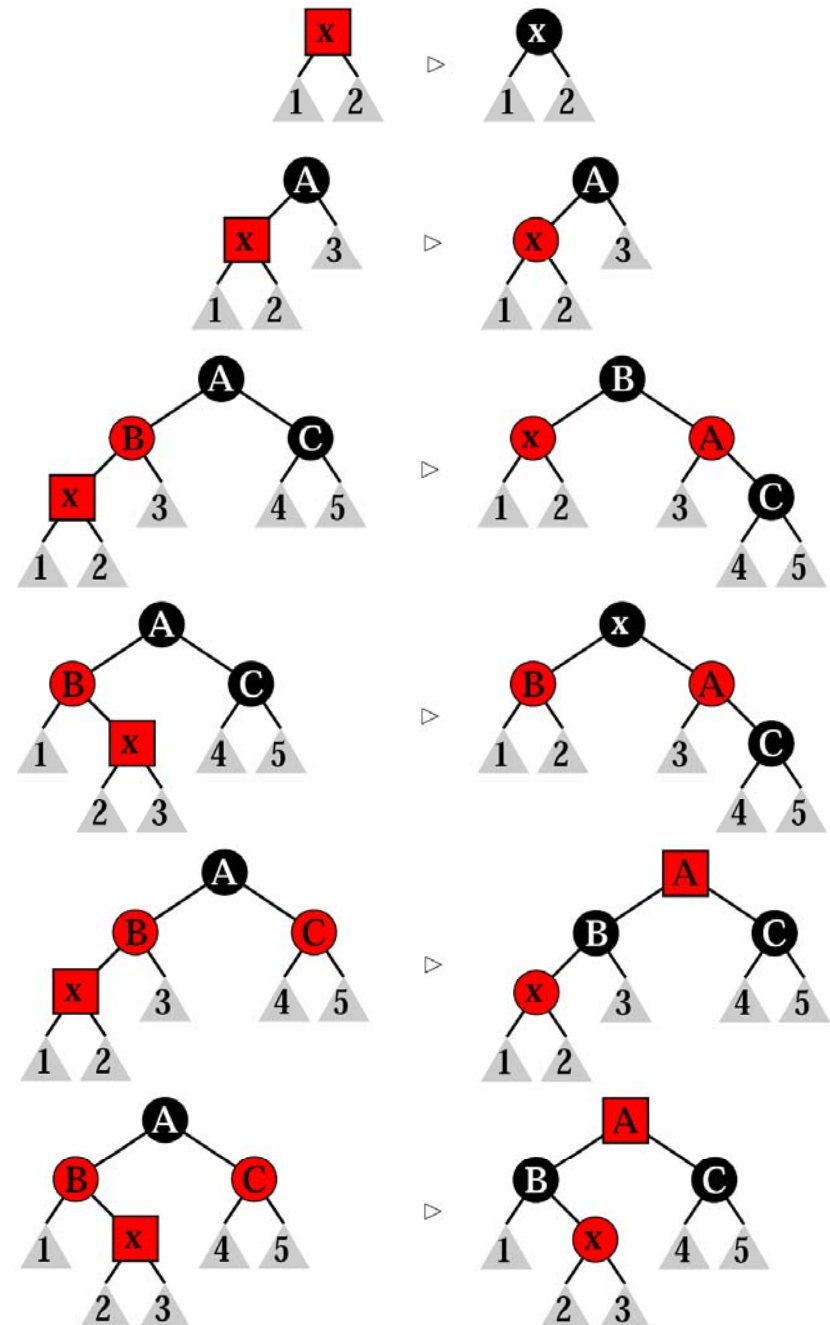
# Eksempel: Rød-Sorte Træer

Insert( $x$ )  
 =  
 Søgning  $\leftarrow O(\log n)$   
 +  
 Opret nyt blad  $\leftarrow O(1)$   
 +  
 Rebalancering  $\leftarrow \# \text{ transitioner}$

# transitioner = amortiseret  $O(1)$

$\Phi = \# \text{ røde knuder}$

**Korollar:** Indsættelse i rød-sorter træer tager amortiseret  $O(1)$  tid, hvis indsættelsespositionen er kendt



# Amortiseret Analyse

- Teknik til at argumentere om **worst-case** ?tiden for en sekvens af operationer
- Behøver kun at analysere operationerne enkeltvis
- **Kunsten**: Find den rigtige invariant for  $\Phi$