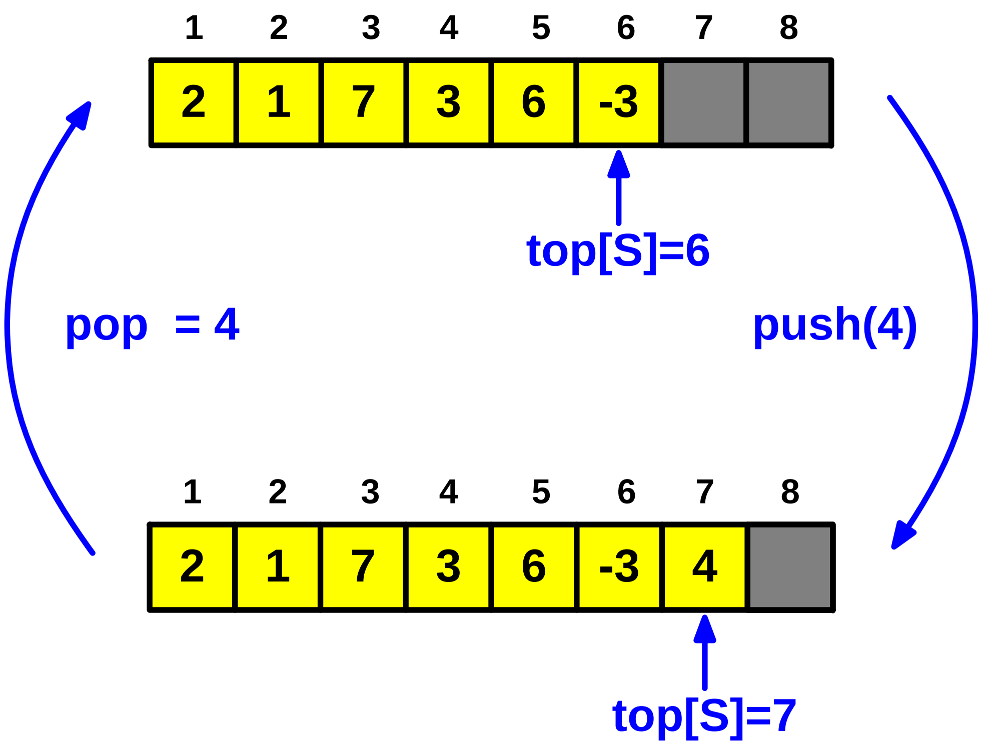
Algoritmer og Datastrukturer 1

Amortiseret Analyse [CLRS, kapitel 17]

Gerth Stølting Brodal
Aarhus Universitet

Stak

Stak : Array Implementation




```
STACK-EMPTY(S)
if top[S] = 0
    then return TRUE
    else  return FALSE

PUSH(S, x)
top[S] ← top[S] + 1
S[top[S]] ← x

POP(S)
if STACK-EMPTY(S)
    then error "underflow"
    else  top[S] ← top[S] − 1
          return S[top[S] + 1]
```

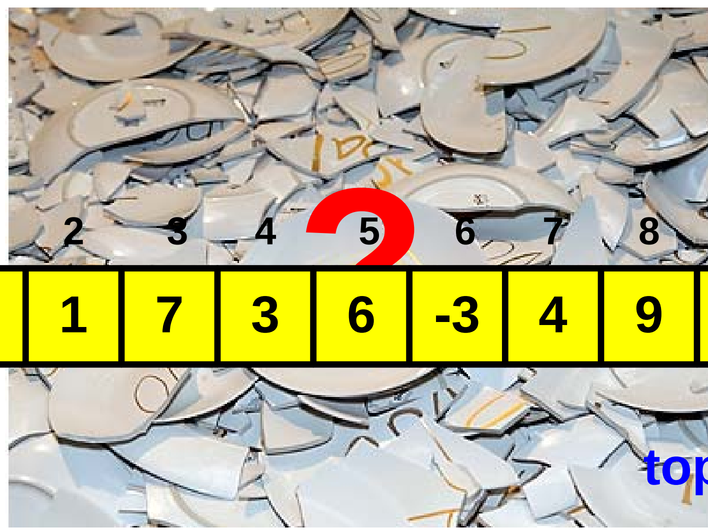
Stack-Empty, Push, Pop : $O(1)$ tid

Stak : Overløb

1	2	3	4	5	6	7	8
2	1	7	3	6	-3	4	9

top[S]=8

Push(13)



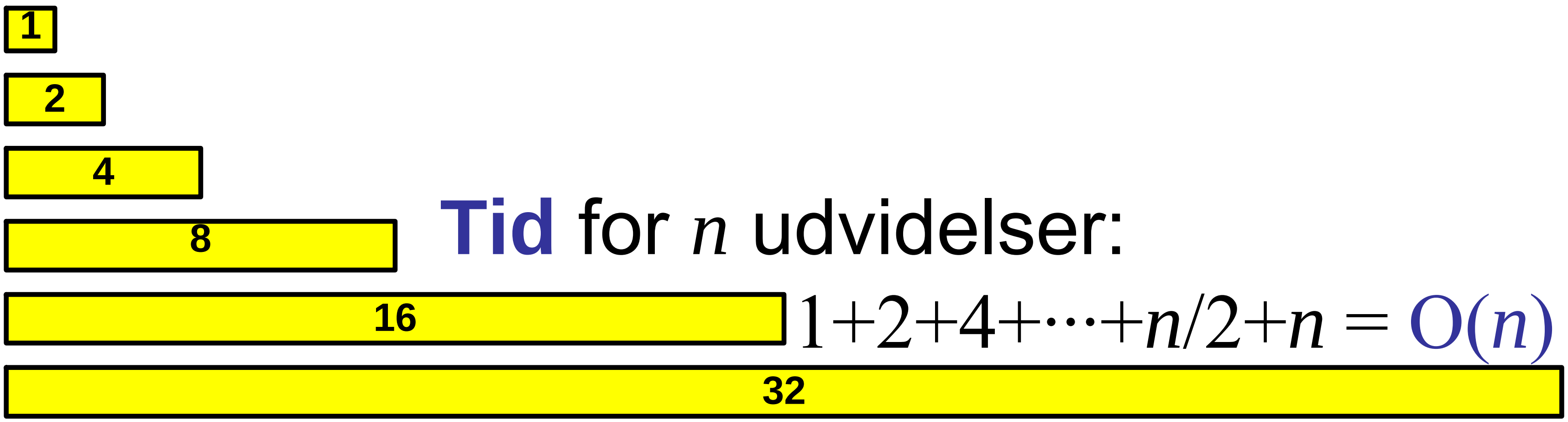
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	1	7	3	6	-3	4	9	13							

top[s]=9

Array fordobling : $O(n)$ tid

Array Fordobling

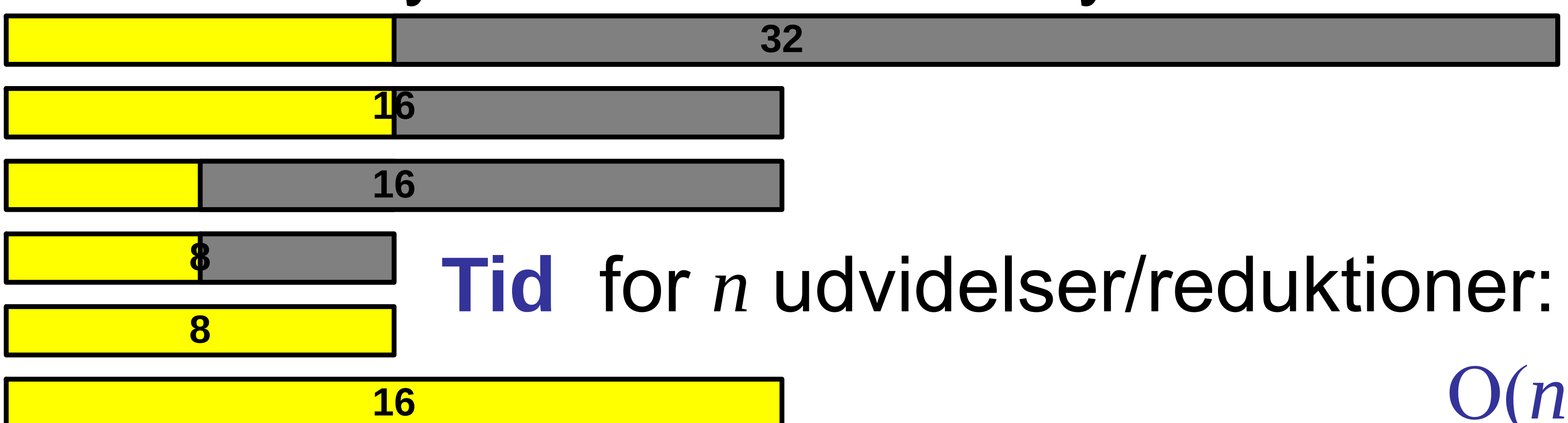
Fordoble arrayet når det er fuld



Tid for n udvidelser:

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

Halver arrayet når det er <1/4 fyldt



Tid for n udvidelser/reduktioner:

$$O(n)$$

Array Fordobling + Halvering
– en generel teknik

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

Plads ≤ 4 · aktuelle antal elementer

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

Analyse teknik ønskes...

Krav

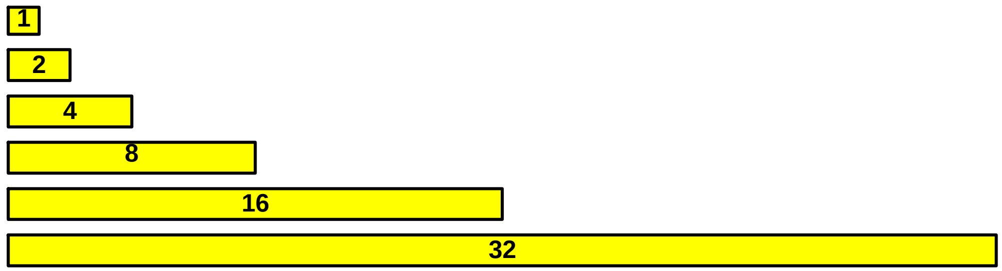
- Analysere **worst-case** tiden for en **sekvens** af opertationer
- Behøver kun at analysere den **enkelte operation**

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



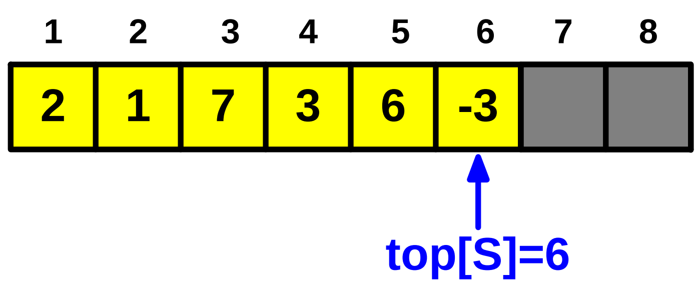
Amortiseret Analyse

- **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 = Φ
- 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 : $\Phi = 2\cdot|\ \mathrm{top}[S]-|S|/2\ |$
- 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 $3n$ €

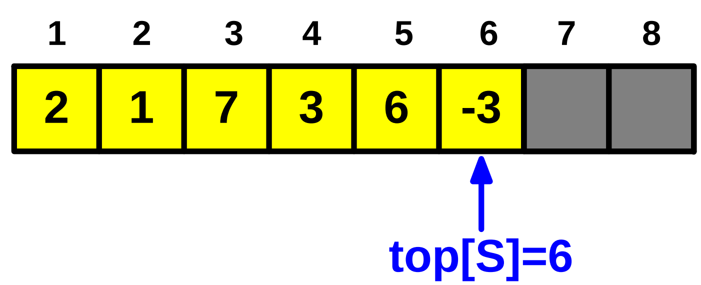


Eksempel: Stak
Push = Amortiseret 3€

- 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 €**
  - Φ før = $|S|$, Φ efter = 2, dvs **$|S|$-2 € frigives**
  - Amortiseret tid $|S|+1-(|S|-2)=3$

Invariant:

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



Amortiseret Analyse

- Teknik til at argumenter om **worst-case** tiden for en sekvens af operationer
- Behøver kun at analysere operationerne enkeltvis

- Kunsten: Find den rigtige invariant for Φ