

Om algoritmers effektivitet

O - en omkostningsrigtig matematisk model for vurdering af algoritmers effektivitet

En velunderbygget teori, som giver estimater, der er uafhængige af den konkrete computer!

Vor indfaldsfinkel: Grundprincipperne og standard-eksempler

1

Opgørelse af effektivitet, eksempel

a=a+1
Tid = 1 (af en slags)

for(int i=1; i<=n; i++)
a=a+1;
Tid = n

for(int i=1; i<=n; i++)
for(int j=1; j<=m; j++)
a=a+1;
Tid = n*m

for(int i=1; i<=n; i++)
for(int j=1; j<=n; j++)
for(int k=1; k<=n; k++)
a=a+1;
Tid = n³

Generelt:

- funktion af input (n, m)

Mere kompliceret ved:

- while-løkker
- if(betingelse) omkring indre løkke
- indre løkke afh. af ydre

2

Matematik til at karakterisere effektivitet

- Normalt ikke interesseret i eksakt tal, men asymptotisk opførsel (dvs. $n \rightarrow \infty$)
- Hvis en funktion $T(n)$ står for eksakt tidsforbrug, bruges $O(F(n))$ til at karakterisere øvre mål for "opførsel"

Foreløbig definition (alternativ til bogen):

$T(n)$ er $O(F(n))$ hvis $T(n)/F(n) \leq$ konstant
når $n \rightarrow \infty$

(virker i alle fornuftige tilfælde)

3

Illustration af def. af $O(-)$

init; // tid = 25
for(i=1; i<=n; i++) // tid = n*17
noget;
for(i=1; i<=n; i++) // tid = n*3
for(j=1; j<=n; j++)
noget_andet;

Total tidsforbrug
 $T(n) = 25 + n*17 + n^2*3$

Påstand: $T(n)$ er $O(n^2)$

Udtales også:

"algoritmen er af orden $O(n^2)$ "
"algoritmen er kvadratisk"

Bevis: $T(n)/F(n) = T(n)/n^2$
 $= 25/n^2 + n*17/n^2 + n^2*3/n^2$
 $= 25/n^2 + 17/n + 3$
 ≤ 3 når $n \rightarrow \infty$

Generelt:

- dominerende led bestemmer O
- konstant uinteressant til generel karakteristik
- størrelsesorden god til estimater (eksempel...)
- snyder ved små n

4

Eksempel: estimat fra $n=1000$ til $n=10000$

$T(n) = 25 + n*17 + n^2*3$ og $F(n) = n^2$

$T(10n)/T(n) \approx F(10n)/F(n)$, dvs. $T(10n) \approx 100 * T(n)$

Antag $T(1000) = 10$ sek,

dvs. $T(10000) = 3.001.725$ t (hvor t er en passende brøkdel af et sek.)

Eksakt værdi for $T(10.000) = 300.170.025$ t

Estimeret ved "*100": $T(10.000) = 1000$ sek

$T(10.000)$ omregnet fra "t" til sek: $300.170.025/3.001.725 * 10$ sek = 999,9 sek

Fejl $\approx 0,1\%$

5

Regneregler om O

Definition: $F(n)$ dominerer over $G(n)$, $F(n) > G(n)$ såfremt
 $G(n)/F(n) \rightarrow 0$ når $n \rightarrow \infty$

I så fald: $O(F(n)+G(n)) = O(F(n))$

hvor $O(H(n)) = O(J(n))$ betyder $H(n)/J(n) \rightarrow c > 0$ når $n \rightarrow \infty$

Eksempel: $O(25 + n*17 + n^2*3) = O(n^2)$

Eksempler på regler:

$O(c*F(n)) = O(F(n))$

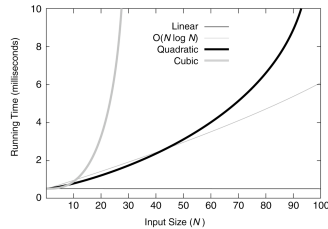
$O(n*F(n)) > O(F(n))$

$O(V(n)*F(n)) > O(F(n))$ såfremt $O(V(n)) > O(1)$

6

Ofte forekommende størrelsesordner

$O(1) < O(\log n) < O((\log n)^2) < O(n) < O(n \log n)$
 $< O(n^2) < O(n^3) < \dots < O(2^n)$



7

Små drilske led:

$O(n + 2^n / 1.000.000)$

n	T
10	10
20	21
30	1.104
40	1.099.541

Hvem ## kan finde den lille tidrøver inden afleveringsprøven i morgen formiddag?

8

Små drilske led, med endnu mindre konstant:

$O(n + 2^n / 1.000.000.000)$

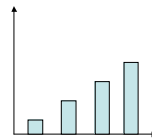
n	T
10	10
20	20
30	31
40	≈ 1.000
50	≈ 1.000.000

9

En korrekt definition

I tilfælde af en funktion/algorithmte med sær opførsel:

```
if(n er lige tal) ; else for(int i=n; i<=n; i++)
```



Vil vi gerne have til at være af orden n ... men grænseværdi ikke veldef.

Definition:

$T(n)$ er $O(F(n))$ hvis der findes positive konstanter c og n_0 , så

$$T(n) \leq c F(n) \quad \text{for alle } n \geq n_0$$

10

Eksempler på polynomielle alg. n^3 , n^2 , n

Find maksimal sum af delsekvens

Eksempel:

-2	11	-4	13	-5	2
----	----	----	----	----	---

11

Eksempler på polynomielle alg. n^3 , n^2 , n

Find maksimal sum af delsekvens

Eksempel:

-2	11	-4	13	-5	2
----	----	----	----	----	---

Den enkle algoritme: Generer samtligt mulige summer og hold rede på den hidtil største.

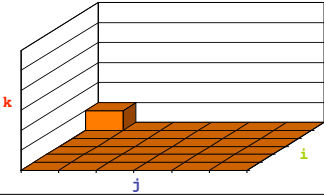
"Indlysende" kubisk, dvs. n^3

12

Kubisk algoritme for max-sum-delsekvens

```
maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}
```

k						
j						
i						
	-2	11	-4	13	-5	2

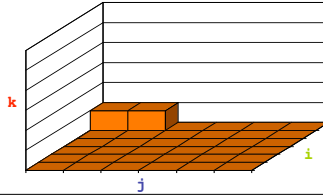


13

Kubisk algoritme for max-sum-delsekvens

```
maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}
```

k						
j						
i						
	-2	11	-4	13	-5	2

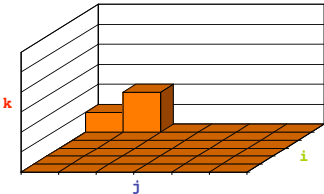


14

Kubisk algoritme for max-sum-delsekvens

```
maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}
```

k						
j						
i						
	-2	11	-4	13	-5	2

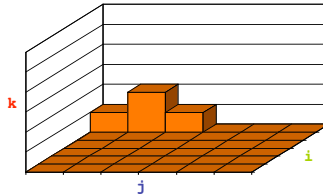


15

Kubisk algoritme for max-sum-delsekvens

```
maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}
```

k						
j						
i						
	-2	11	-4	13	-5	2

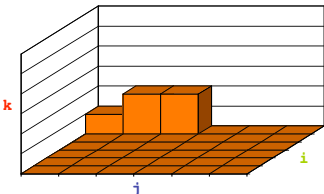


16

Kubisk algoritme for max-sum-delsekvens

```
maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}
```

k						
j						
i						
	-2	11	-4	13	-5	2

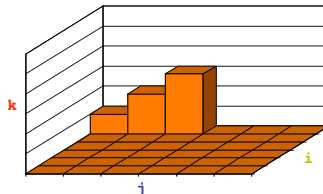


17

Kubisk algoritme for max-sum-delsekvens

```
maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}
```

k						
j						
i						
	-2	11	-4	13	-5	2



18

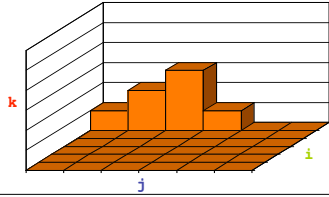
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k						
j						
i						
	-2	11	-4	13	-5	2



19

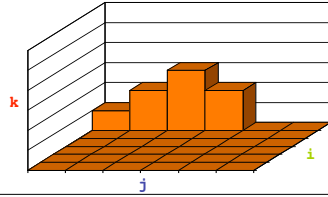
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k						
j						
i						
	-2	11	-4	13	-5	2



20

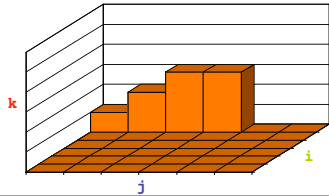
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k						
j						
i						
	-2	11	-4	13	-5	2



21

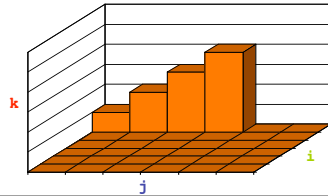
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k						
j						
i						
	-2	11	-4	13	-5	2



22

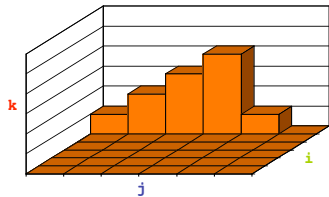
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k						
j						
i						
	-2	11	-4	13	-5	2



23

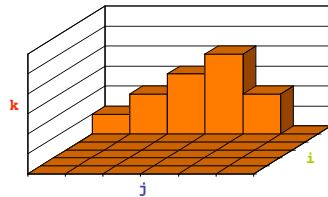
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k						
j						
i						
	-2	11	-4	13	-5	2



24

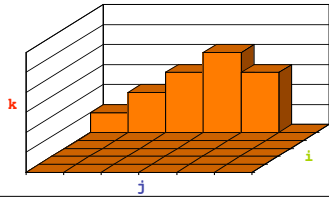
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k						
j						
i						
	-2	11	-4	13	-5	2



25

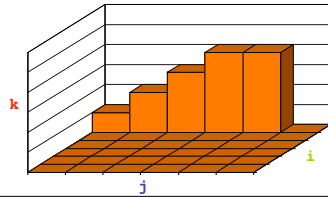
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k						
j						
i						
	-2	11	-4	13	-5	2



26

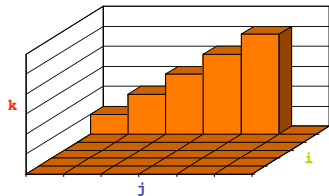
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k						
j						
i						
	-2	11	-4	13	-5	2



27

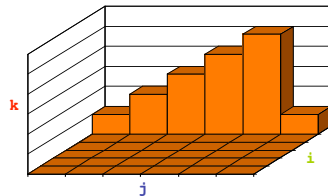
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k						
j						
i						
	-2	11	-4	13	-5	2



28

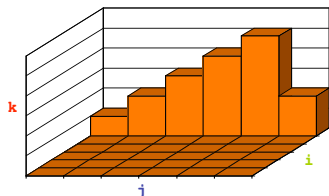
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k						
j						
i						
	-2	11	-4	13	-5	2



29

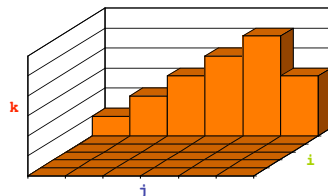
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k						
j						
i						
	-2	11	-4	13	-5	2



30

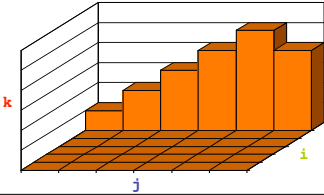
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k						
j						
i						
	-2	11	-4	13	-5	2



31

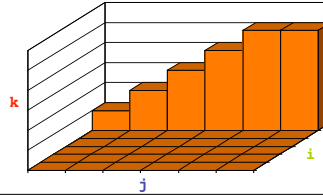
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k						
j						
i						
	-2	11	-4	13	-5	2



32

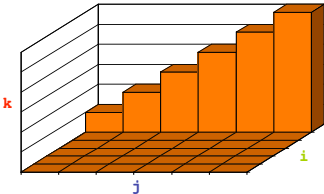
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k						
j						
i						
	-2	11	-4	13	-5	2



33

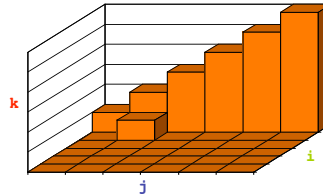
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k						
j						
i						
	-2	11	-4	13	-5	2



34

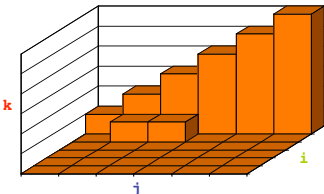
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k						
j						
i						
	-2	11	-4	13	-5	2



35

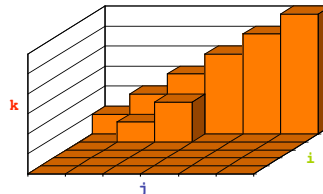
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k						
j						
i						
	-2	11	-4	13	-5	2



36

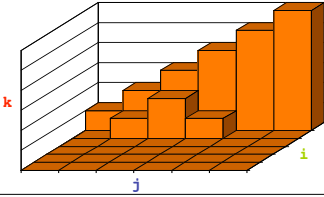
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k						
j						
i						
	-2	11	-4	13	-5	2



37

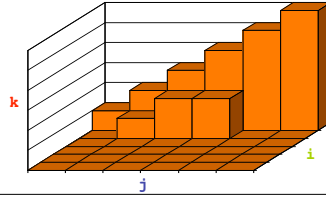
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k						
j						
i						
	-2	11	-4	13	-5	2



38

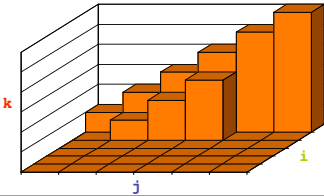
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k						
j						
i						
	-2	11	-4	13	-5	2



39

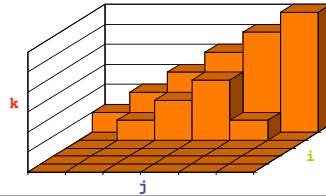
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k						
j						
i						
	-2	11	-4	13	-5	2



40

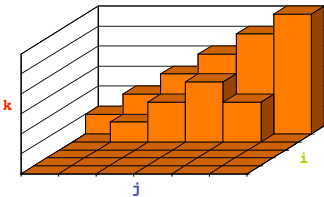
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k						
j						
i						
	-2	11	-4	13	-5	2



41

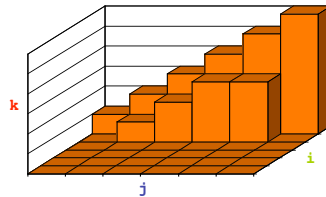
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k						
j						
i						
	-2	11	-4	13	-5	2



42

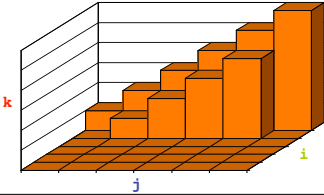
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k						
j						
i						
	-2	11	-4	13	-5	2



43

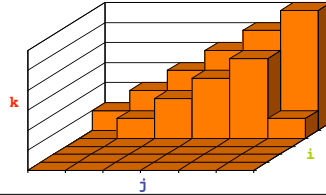
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k						
j						
i						
	-2	11	-4	13	-5	2



44

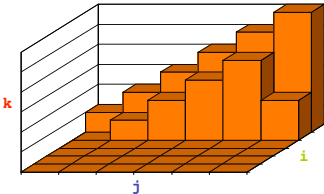
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k						
j						
i						
	-2	11	-4	13	-5	2



45

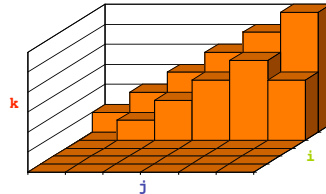
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k						
j						
i						
	-2	11	-4	13	-5	2



46

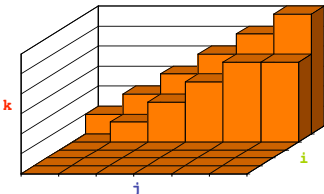
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k						
j						
i						
	-2	11	-4	13	-5	2



47

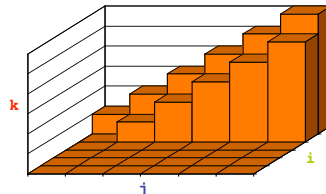
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k						
j						
i						
	-2	11	-4	13	-5	2



48

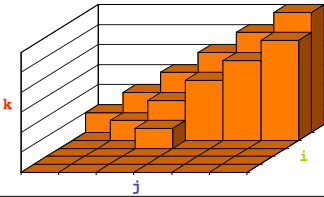
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k				□		
j				□		
i			□			
	-2	11	-4	13	-5	2



49

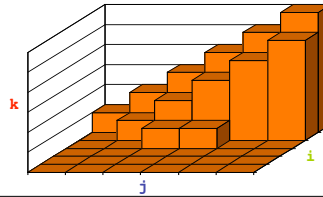
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k				□		
j				□		
i			□			
	-2	11	-4	13	-5	2



50

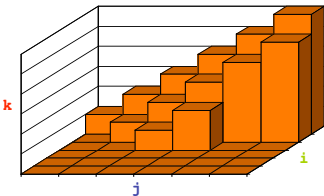
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k				□		
j				□		
i			□			
	-2	11	-4	13	-5	2



51

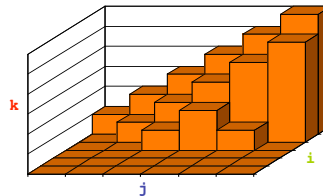
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k				□		
j				□		
i			□			
	-2	11	-4	13	-5	2



52

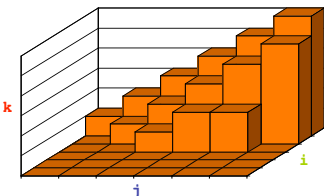
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k				□		
j				□		
i			□			
	-2	11	-4	13	-5	2



53

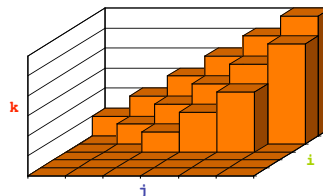
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k				□		
j				□		
i			□			
	-2	11	-4	13	-5	2



54

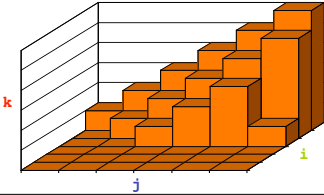
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k						
j						
i						
	-2	11	-4	13	-5	2



55

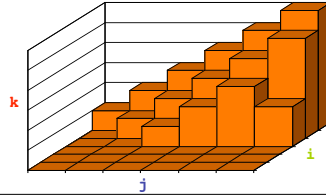
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k						
j						
i						
	-2	11	-4	13	-5	2



56

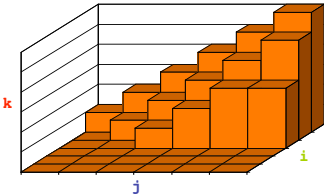
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k						
j						
i						
	-2	11	-4	13	-5	2



57

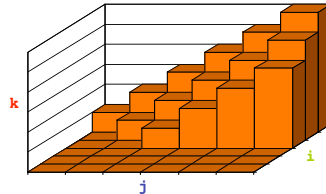
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k						
j						
i						
	-2	11	-4	13	-5	2



58

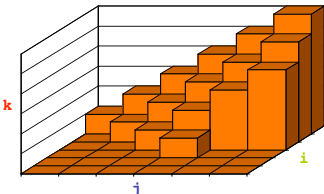
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k						
j						
i						
	-2	11	-4	13	-5	2



59

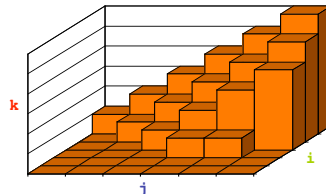
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k						
j						
i						
	-2	11	-4	13	-5	2



60

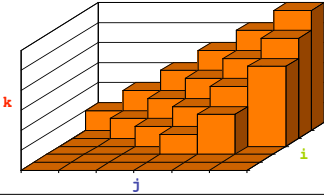
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k						
j						
i						
	-2	11	-4	13	-5	2



61

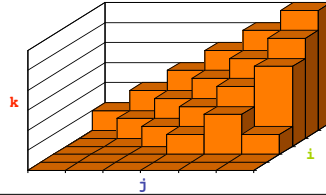
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k						
j						
i						
	-2	11	-4	13	-5	2



62

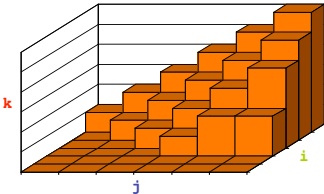
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k						
j						
i						
	-2	11	-4	13	-5	2



63

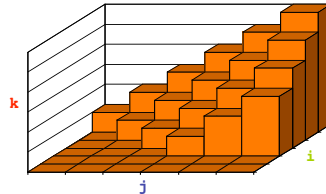
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k						
j						
i						
	-2	11	-4	13	-5	2



64

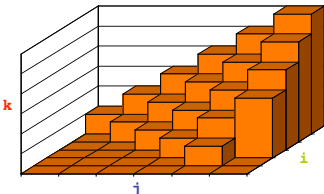
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k						
j						
i						
	-2	11	-4	13	-5	2



65

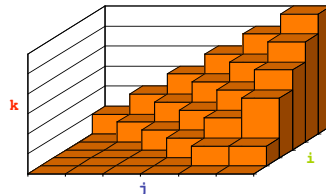
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k						
j						
i						
	-2	11	-4	13	-5	2



66

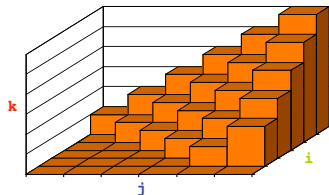
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k							□
j							□
i							□
	-2	11	-4	13	-5	2	



67

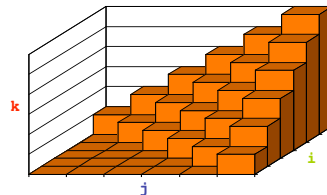
Kubisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  for(j=i; j<=n; j++) {
    sum = 0;
    for(k=i; k<=j; k++)
      sum += a[k];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

k							□
j							□
i							□
	-2	11	-4	13	-5	2	



$$\begin{aligned}
 &1 + (1+2) + (1+2+3) + \dots \\
 &+ (1+2+\dots+n) \\
 &= n * (n+1) * (n+2) / 6 \\
 &\text{dvs. } O(n^3)
 \end{aligned}$$

68

Kvadratisk algoritme for max-sum-delsekvens

Udnyt at "næste sum" = "gammel sum" + "næste led"

$$\text{Dvs. } \text{Sum}_{i,j+1} = \text{Sum}_{i,j} + a[j+1]$$

```

maxSum = 0;
for(i=1; i<=n; i++) {
  sum = 0;
  for(j=i; j<=n; j++) {
    sum += a[j];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

69

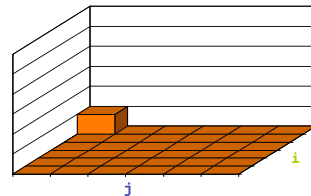
Kvadratisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  sum = 0;
  for(j=i; j<=n; j++) {
    sum += a[j];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

j	□						
i	□						
	-2	11	-4	13	-5	2	



70

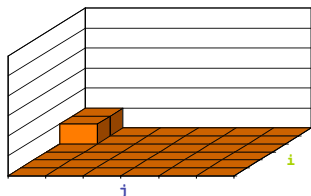
Kvadratisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  sum = 0;
  for(j=i; j<=n; j++) {
    sum += a[j];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

j		□					
i	□						
	-2	11	-4	13	-5	2	



71

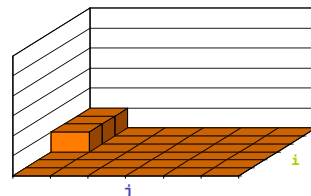
Kvadratisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  sum = 0;
  for(j=i; j<=n; j++) {
    sum += a[j];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

j			□				
i	□						
	-2	11	-4	13	-5	2	



72

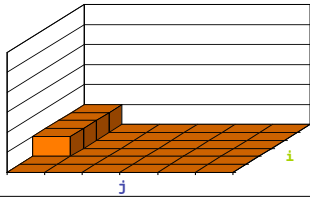
Kvadratisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  sum = 0;
  for(j=i; j<=n; j++) {
    sum += a[j];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

j							□
i	□						
	-2	11	-4	13	-5	2	



73

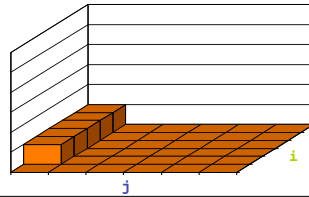
Kvadratisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  sum = 0;
  for(j=i; j<=n; j++) {
    sum += a[j];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

j							□
i	□						
	-2	11	-4	13	-5	2	



74

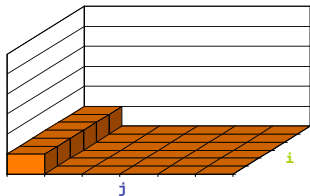
Kvadratisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  sum = 0;
  for(j=i; j<=n; j++) {
    sum += a[j];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

j							□
i	□						
	-2	11	-4	13	-5	2	



75

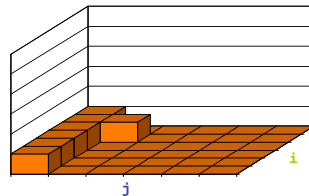
Kvadratisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  sum = 0;
  for(j=i; j<=n; j++) {
    sum += a[j];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

j		□					
i	□	□					
	-2	11	-4	13	-5	2	



76

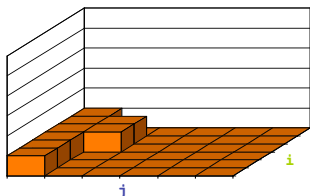
Kvadratisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  sum = 0;
  for(j=i; j<=n; j++) {
    sum += a[j];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

j			□				
i		□	□				
	-2	11	-4	13	-5	2	



77

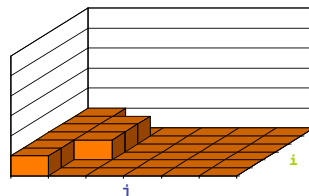
Kvadratisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  sum = 0;
  for(j=i; j<=n; j++) {
    sum += a[j];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

j				□			
i			□	□			
	-2	11	-4	13	-5	2	



78

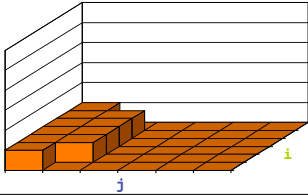
Kvadratisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  sum = 0;
  for(j=i; j<=n; j++) {
    sum += a[j];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

j							□
i		□					
	-2	11	-4	13	-5	2	



79

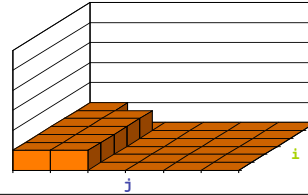
Kvadratisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  sum = 0;
  for(j=i; j<=n; j++) {
    sum += a[j];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

j							□
i		□					
	-2	11	-4	13	-5	2	



80

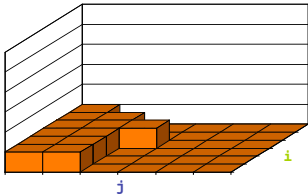
Kvadratisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  sum = 0;
  for(j=i; j<=n; j++) {
    sum += a[j];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

j			□				
i			□				
	-2	11	-4	13	-5	2	



81

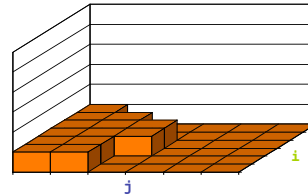
Kvadratisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  sum = 0;
  for(j=i; j<=n; j++) {
    sum += a[j];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

j				□			
i				□			
	-2	11	-4	13	-5	2	



82

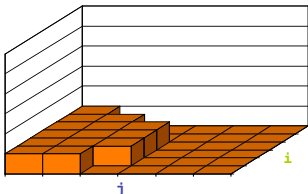
Kvadratisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  sum = 0;
  for(j=i; j<=n; j++) {
    sum += a[j];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

j					□		
i					□		
	-2	11	-4	13	-5	2	



83

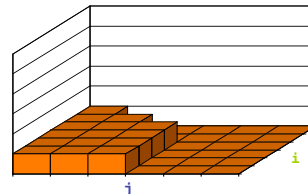
Kvadratisk algoritme for max-sum-delsekvens

```

maxSum = 0;
for(i=1; i<=n; i++) {
  sum = 0;
  for(j=i; j<=n; j++) {
    sum += a[j];
    if(sum>maxSum)
      maxSum = sum;
  }
}

```

j						□	
i						□	
	-2	11	-4	13	-5	2	

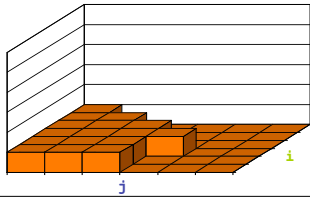


84

Kvadratisk algoritme for max-sum-delsekvens

```
maxSum = 0;
for(i=1; i<=n; i++) {
  sum = 0;
  for(j=i; j<=n; j++) {
    sum += a[j];
    if(sum>maxSum)
      maxSum = sum;
  }
}
```

j						
i						
	-2	11	-4	13	-5	2

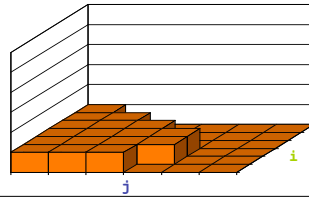


85

Kvadratisk algoritme for max-sum-delsekvens

```
maxSum = 0;
for(i=1; i<=n; i++) {
  sum = 0;
  for(j=i; j<=n; j++) {
    sum += a[j];
    if(sum>maxSum)
      maxSum = sum;
  }
}
```

j						
i						
	-2	11	-4	13	-5	2

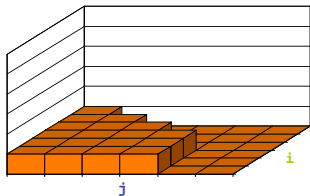


86

Kvadratisk algoritme for max-sum-delsekvens

```
maxSum = 0;
for(i=1; i<=n; i++) {
  sum = 0;
  for(j=i; j<=n; j++) {
    sum += a[j];
    if(sum>maxSum)
      maxSum = sum;
  }
}
```

j						
i						
	-2	11	-4	13	-5	2

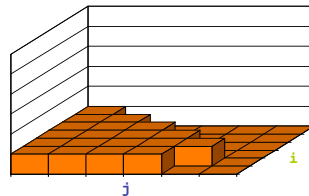


87

Kvadratisk algoritme for max-sum-delsekvens

```
maxSum = 0;
for(i=1; i<=n; i++) {
  sum = 0;
  for(j=i; j<=n; j++) {
    sum += a[j];
    if(sum>maxSum)
      maxSum = sum;
  }
}
```

j						
i						
	-2	11	-4	13	-5	2

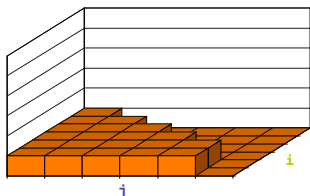


88

Kvadratisk algoritme for max-sum-delsekvens

```
maxSum = 0;
for(i=1; i<=n; i++) {
  sum = 0;
  for(j=i; j<=n; j++) {
    sum += a[j];
    if(sum>maxSum)
      maxSum = sum;
  }
}
```

j						
i						
	-2	11	-4	13	-5	2



89

Kvadratisk algoritme for max-sum-delsekvens

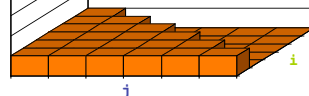
```
maxSum = 0;
for(i=1; i<=n; i++) {
  sum = 0;
  for(j=i; j<=n; j++) {
    sum += a[j];
    if(sum>maxSum)
      maxSum = sum;
  }
}
```

j						
i						
	-2	11	-4	13	-5	2

Læs selv om lineær algoritme i bog:
 Eksempel på at optimere algoritmer
 kan være svært gennemskuelig!

$$1 + 2 + \dots + n = n * (n-1) / 2$$

altså $O(n^2)$



90

Logaritmisk, typisk for del-og-hersk algoritmer

Eksempel: Binær søgning for at finde element i **sorteret** sekvens
~ a la telefonbog

Princippet:

- at *finde* x i sekvens af lgd. 1: test " $=x$ " ja/nej
- ellers, hvis $x \leq$ midt-element, så *find* x i venstre halvdel
ellers *find* x i højre halvdel

Tidsforbrug

- hvert skridt ("•" ovenfor) konstant
- hvert skridt halverer længden af sekvensen

Total tid for lgd. n : så mange gange n skal halveres for at blive "1"
 $\approx \log_2 n$

Hvordan var det nu?

91

Lidt om logaritmfunktionen

Defineret som omvendt til eksponentialfunktionen:

hvis $2^x = m$ så $\log_2 m = x$

Eksempler:

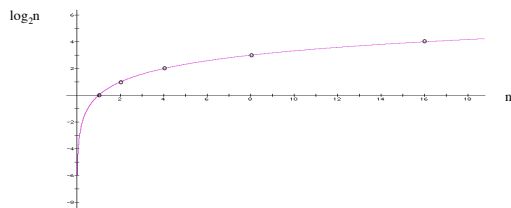
n	binært	$\log_2 n$	Halveringer
1	1	0	-
2	10	1	□ 1
4	100	2	□ 2 □ 1
8	1000	3	□ 4 □ 2 □ 1
16	10000	4	□ 8 □ 4 □ 2 □ 1

Egenskaber:

- fordobles argumentet stiger logaritmen med 1
- $\log_2 n \approx$ antal tegn i n 's binære repræsentation
- $\log_2 n \approx$ antal gange n skal halveres for at blive 1

92

Funktionskurven for $\log_2 n$



93

Binær søgning i Java I: Rekursiv for "int []"

Kaldes: `binarySearch(0, a.length-1, a, x)`

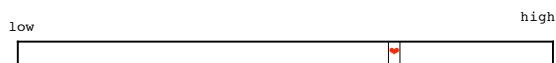
```
int binarySearch(int low, int high, int [] a, int x){
    if(low>high) return -1;
    int mid = (low+high)/2;
    if( a[mid] < x )
        binarySearch(mid+1,high,a,x)
    else if( a[mid] > x )
        binarySearch(low,mid-1,a,x);
    else return mid; } //a[mid]==x
```

94

Binær søgning i Java I: Rekursiv for "int []"

Kaldes: `binarySearch(0, a.length-1, a, x)`

```
int binarySearch(int low, int high, int [] a, int x){
    if(low>high) return -1;
    int mid = (low+high)/2;
    if( a[mid] < x )
        binarySearch(mid+1,high,a,x)
    else if( a[mid] > x )
        binarySearch(low,mid-1,a,x);
    else return mid; } //a[mid]==x
```

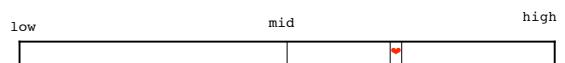


95

Binær søgning i Java I: Rekursiv for "int []"

Kaldes: `binarySearch(0, a.length-1, a, x)`

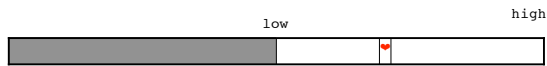
```
int binarySearch(int low, int high, int [] a, int x){
    if(low>high) return -1;
    int mid = (low+high)/2;
    if( a[mid] < x )
        binarySearch(mid+1,high,a,x)
    else if( a[mid] > x )
        binarySearch(low,mid-1,a,x);
    else return mid; } //a[mid]==x
```



96

Binær søgning i Java I: Rekursiv for "int []"

```
Kaldes: binarySearch(0,a.length-1,a,x)
int binarySearch(int low, int high, int [] a, int x){
  if(low>high) return -1;
  int mid = (low+high)/2;
  if( a[mid] < x )
    binarySearch(mid+1,high,a,x)
  else if( a[mid] > x )
    binarySearch(low,mid-1,a,x);
  else return mid; } //a[mid]==x
```



97

Binær søgning i Java I: Rekursiv for "int []"

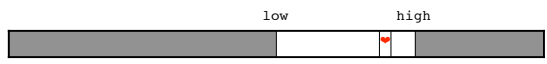
```
Kaldes: binarySearch(0,a.length-1,a,x)
int binarySearch(int low, int high, int [] a, int x){
  if(low>high) return -1;
  int mid = (low+high)/2;
  if( a[mid] < x )
    binarySearch(mid+1,high,a,x)
  else if( a[mid] > x )
    binarySearch(low,mid-1,a,x);
  else return mid; } //a[mid]==x
```



98

Binær søgning i Java I: Rekursiv for "int []"

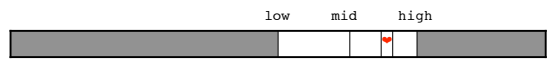
```
Kaldes: binarySearch(0,a.length-1,a,x)
int binarySearch(int low, int high, int [] a, int x){
  if(low>high) return -1;
  int mid = (low+high)/2;
  if( a[mid] < x )
    binarySearch(mid+1,high,a,x)
  else if( a[mid] > x )
    binarySearch(low,mid-1,a,x);
  else return mid; } //a[mid]==x
```



99

Binær søgning i Java I: Rekursiv for "int []"

```
Kaldes: binarySearch(0,a.length-1,a,x)
int binarySearch(int low, int high, int [] a, int x){
  if(low>high) return -1;
  int mid = (low+high)/2;
  if( a[mid] < x )
    binarySearch(mid+1,high,a,x)
  else if( a[mid] > x )
    binarySearch(low,mid-1,a,x);
  else return mid; } //a[mid]==x
```



100

Binær søgning i Java I: Rekursiv for "int []"

```
Kaldes: binarySearch(0,a.length-1,a,x)
int binarySearch(int low, int high, int [] a, int x){
  if(low>high) return -1;
  int mid = (low+high)/2;
  if( a[mid] < x )
    binarySearch(mid+1,high,a,x)
  else if( a[mid] > x )
    binarySearch(low,mid-1,a,x);
  else return mid; } //a[mid]==x
```



101

Binær søgning i Java I: Rekursiv for "int []"

```
Kaldes: binarySearch(0,a.length-1,a,x)
int binarySearch(int low, int high, int [] a, int x){
  if(low>high) return -1;
  int mid = (low+high)/2;
  if( a[mid] < x )
    binarySearch(mid+1,high,a,x)
  else if( a[mid] > x )
    binarySearch(low,mid-1,a,x);
  else return mid; } //a[mid]==x
```



102

Binær søgning i Java I: Rekursiv for "int []"

```
Kaldes: binarySearch(0,a.length-1,a,x)
int binarySearch(int low, int high, int [] a, int x){
    if(low>high) return -1;
    int mid = (low+high)/2;
    if( a[mid] < x )
        binarySearch(mid+1,high,a,x)
    else if( a[mid] > x )
        binarySearch(low,mid-1,a,x);
    else return mid; } //a[mid]==x
```



103

Binær søgning i Java I: Rekursiv for "int []"

```
Kaldes: binarySearch(0,a.length-1,a,x)
int binarySearch(int low, int high, int [] a, int x){
    if(low>high) return -1;
    int mid = (low+high)/2;
    if( a[mid] < x )
        binarySearch(mid+1,high,a,x)
    else if( a[mid] > x )
        binarySearch(low,mid-1,a,x);
    else return mid; } //a[mid]==x
```



104

Binær søgning i Java I: Rekursiv for "int []"

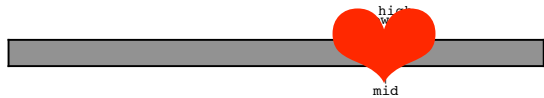
```
Kaldes: binarySearch(0,a.length-1,a,x)
int binarySearch(int low, int high, int [] a, int x){
    if(low>high) return -1;
    int mid = (low+high)/2;
    if( a[mid] < x )
        binarySearch(mid+1,high,a,x)
    else if( a[mid] > x )
        binarySearch(low,mid-1,a,x);
    else return mid; } //a[mid]==x
```



105

Binær søgning i Java I: Rekursiv for "int []"

```
Kaldes: binarySearch(0,a.length-1,a,x)
int binarySearch(int low, int high, int [] a, int x){
    if(low>high) return -1;
    int mid = (low+high)/2;
    if( a[mid] < x )
        binarySearch(mid+1,high,a,x)
    else if( a[mid] > x )
        binarySearch(low,mid-1,a,x);
    else return mid; } //a[mid]==x
```



106

Binær søgning i Java II: Rekursiv, generisk

```
package java.lang;
public interface Comparable {in compareTo(Object other)};

public static int binarySearch(Comparable [] a, Comparable x){
    binarySearch(0,a.length-1,a,x)
}

public static int binarySearch(int low,int high,
    Comparable [] a, Comparable x){
    if(low>high) return -1;
    int mid = (low+high)/2
    if(a[mid].compareTo(x)<0)
        binarySearch!(mid+1,high,a,x)
    else if(a[mid].compareTo(x)>0)
        binarySearch!(low,mid-1,a,x);
    else return mid; }

class Elephant implements Comparable
{...; public int compareTo(...){...} }

zoo = new Elephant []{...};
dumbo = new Elephant;
int where_is_dumbo =
    binarySearch(zoo,dumbo);
```

107

Binær søgning i Java III: Iterativ, generisk

Lærebogens version:

```
int binarySearch(Comparable[] a, Comparable x) {
    int low = 0;
    int high = a.length-1;
    int mid;
    while( low<=high){
        mid = (low+high)/2;
        if(a[mid].compareTo(x)<0)
            low = mid+1;
        else if(a[mid].compareTo(x)>0)
            high = mid-1;
        else return mid; }
    return -1; }
```

Iterativ vs. rekursiv:
En smagssag, men rekursive alg.
ofte simple for af-natur-rek.
problemer

Hvad er mest effektivt?
Er rekursion ikke meget dyrt?

Det afhænger af compileren!
Med en god Java compiler,
identiske køretider!

108

Afsluttende om O-notation

- udvide med flere parametre, f.eks. $O(m \cdot 2^n)$
- O-notation benyttes også for pladsforbrug
 - ofte trade-off plads- vs. tidskompleksitet
- Algoritmer har bedste, værste og gennemsnitligt tidsforbrug
 - f.eks. quicksort, værst $O(n^2)$, gennemsnit $O(n \log n)$
- "T(n) er $O(F(n))$ " angiver en overgrænse for T(n)'s asymptotiske opførsel; alternative karakteristiker:
 - "T(n) er $\Omega(F(n))$ " angiver undergrænse
 - "T(n) er $\Theta(F(n))$ " angiver eksakt karakteristik
 - "T(n) er $o(F(n))$ " angiver karakteristik som er klart for høj

109

Opgaver til øvelserne

Opg. 1: Se på egenskaber a la $1+2+3+4+\dots+n = n(n-1)/2$

Opg. 5.7 i bogen: Store-O for regning med blyant og papir

Opg. 5.8 m. tilføjelse: Store-O for to måder at regne x^n på

Opg. 2, afleveringsopgave: Implementér to måder at implementere mængder af tal, med metoder

```
public void indsæt(int x)
public boolean med_i(int x)
```

Afleveringsfrist 14. oktober

110