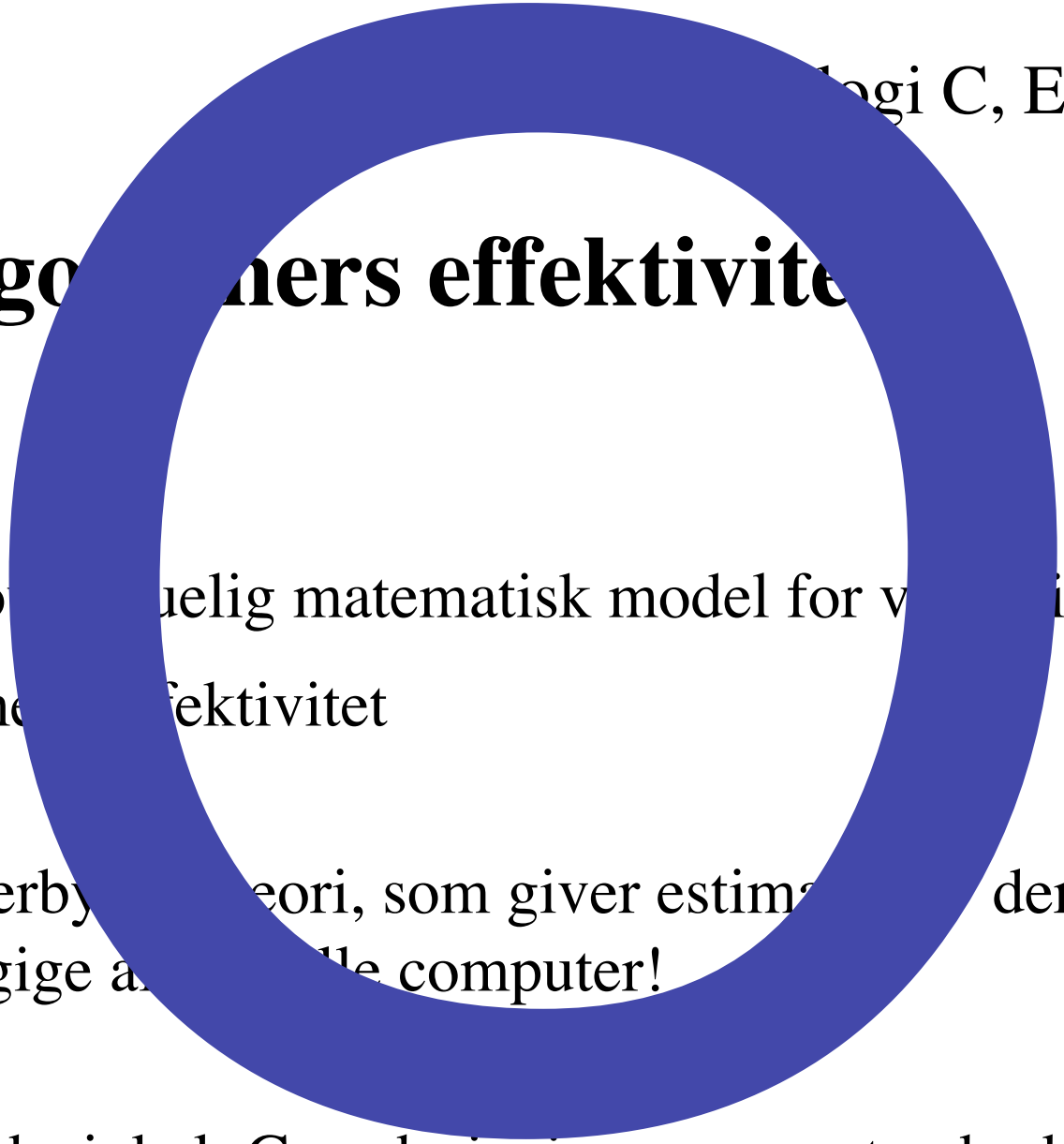
Om algo hers effektivite



O - en o uelig matematisk model for v ing af algoritme fektivitet

En velunderby eori, som giver estima der er uafhængige a le computer!

Vor indfaldsvinkel: Grundprincipperne og standard-eksempler

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; j<=n; k++)
      a=a+1;
```

Tid = n^3

Generelt:

- funktion af input (*n, m*)

Mere kompliceret ved:

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

Matematik til at karakterisere effektivitet

- Normalt ikke interesseret i eksakt tal, men asymptotisk opførsel (dvs. $n \to \infty$)
- 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))* hviss $T(n)/F(N) \to$ konstant

når $n \to \infty$

(virker i alle fornuftige tilfælde)

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 = n²* 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: $T(n)/F(n) = T(n)/n^2$

$= 25/n^2 + n*17/n^2 + n^2*3/n^2$

$= 25/n^2 + 17/n + 3$

$\longrightarrow 3$ når $n \longrightarrow \infty$

Generelt:

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

Eksempel: estimat fra n= 1000 til n=10000

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

$T(10\ n)/T(n) \approx F(10\ n)/F(n) = (10n)^2/n^2 = 100n^2/n^2 = 100$,

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

*Antag T(1000) = **10 sek**,*

*dvs. T(1000) = **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) = **1000sek***

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

*Fejl ≈ **0,1‰***

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

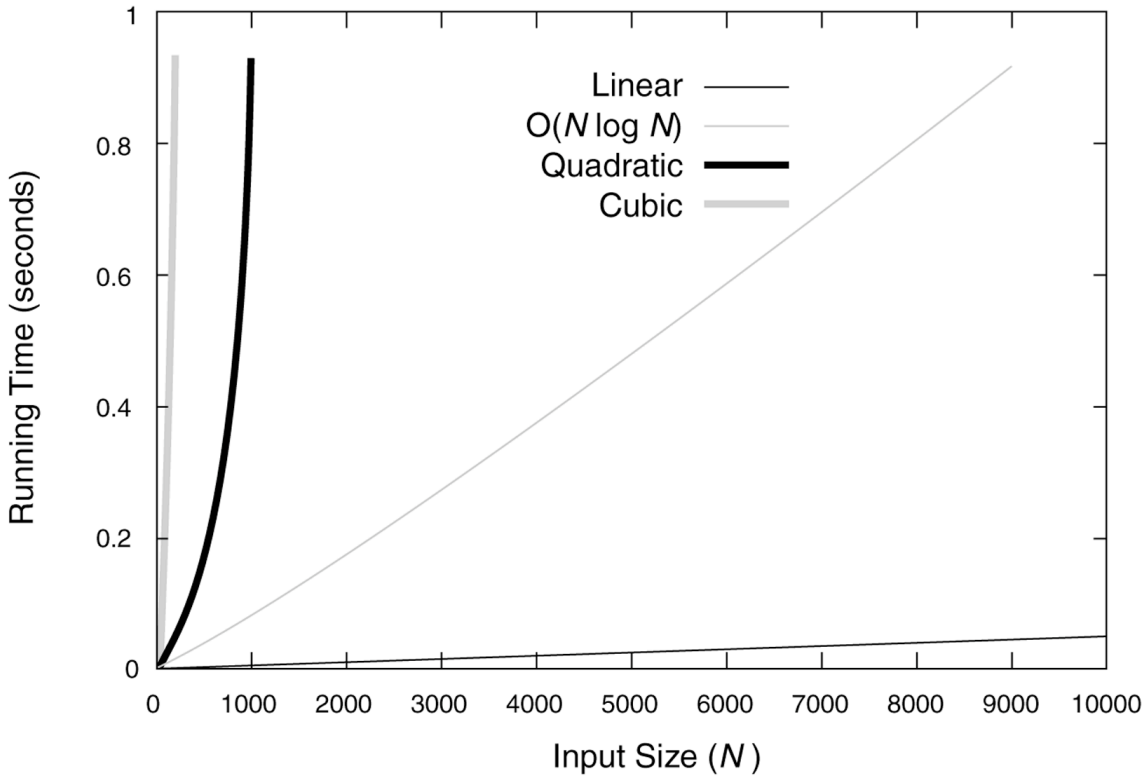
Eksempler på regler:

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)

Ofte forekommende størrelsesordner

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

$< O(n^2) < O(n^3) < \ldots < O(2^n)$



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?

Små drilske led, med endnu mindre konstant:

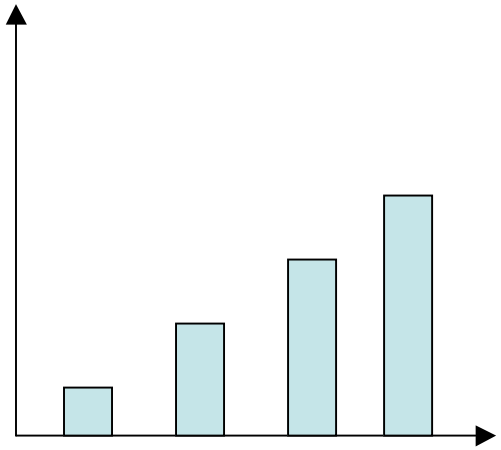
$O(n + 2^n/1.000.000.000)$

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

En korrekt definition

I tilfælde af en funktion/algoritme 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))* hviss der findes positive konstanter *c* og n_0, så

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

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

Find maksimal sum af delsekvens

Eksempel:

–2	11	–4	13	–5	2

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

Find maksimal sum af delsekvens

Eksempel:

–2	**11**	**–4**	**13**	–5	2

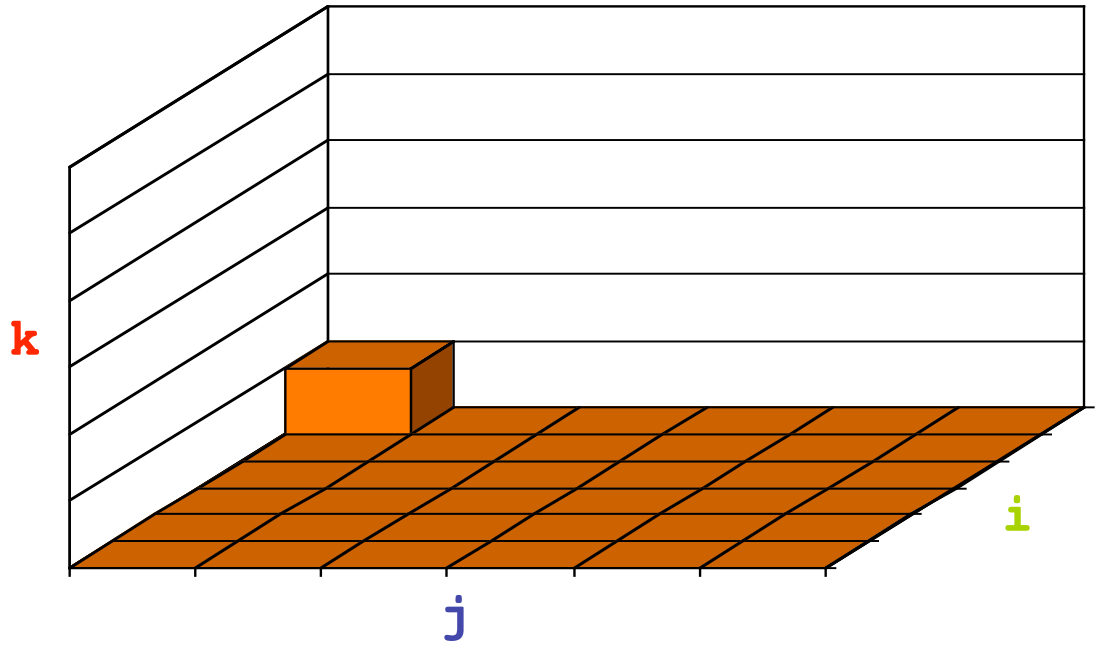
Den enkle algoritme: Generer samtligt mulige summer og hold rede på den hidtil største.

"Indlysende" kubisk, dvs. n^3

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;    }}
```

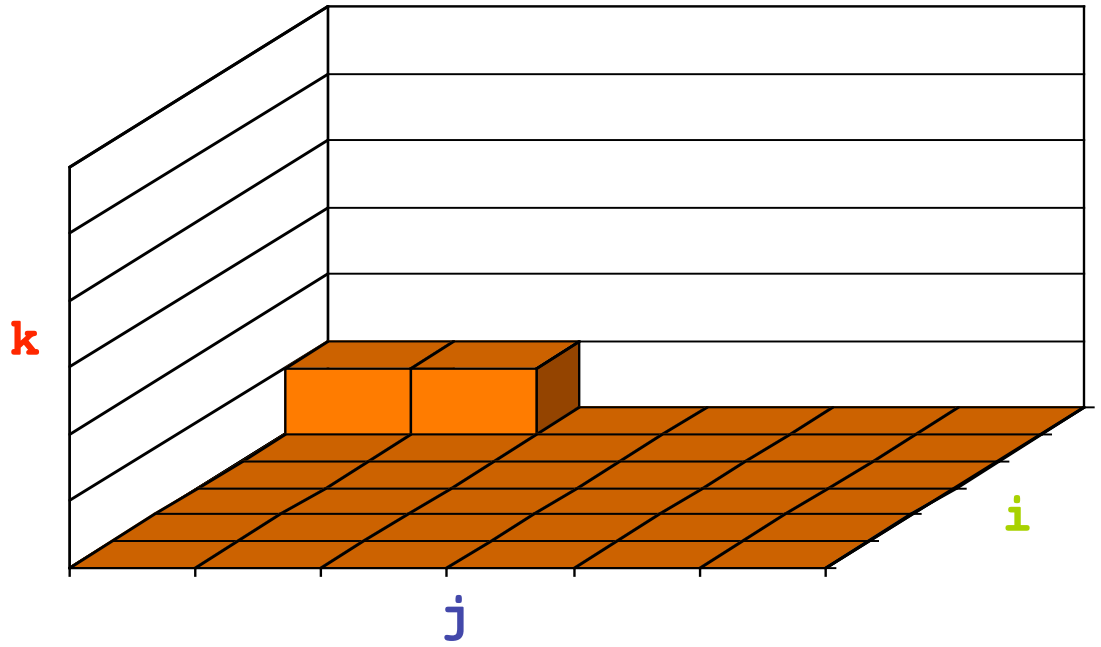
k	↓					
j	↓					
i	↓					
	–2	11	-4	13	–5	2



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;    }}
```

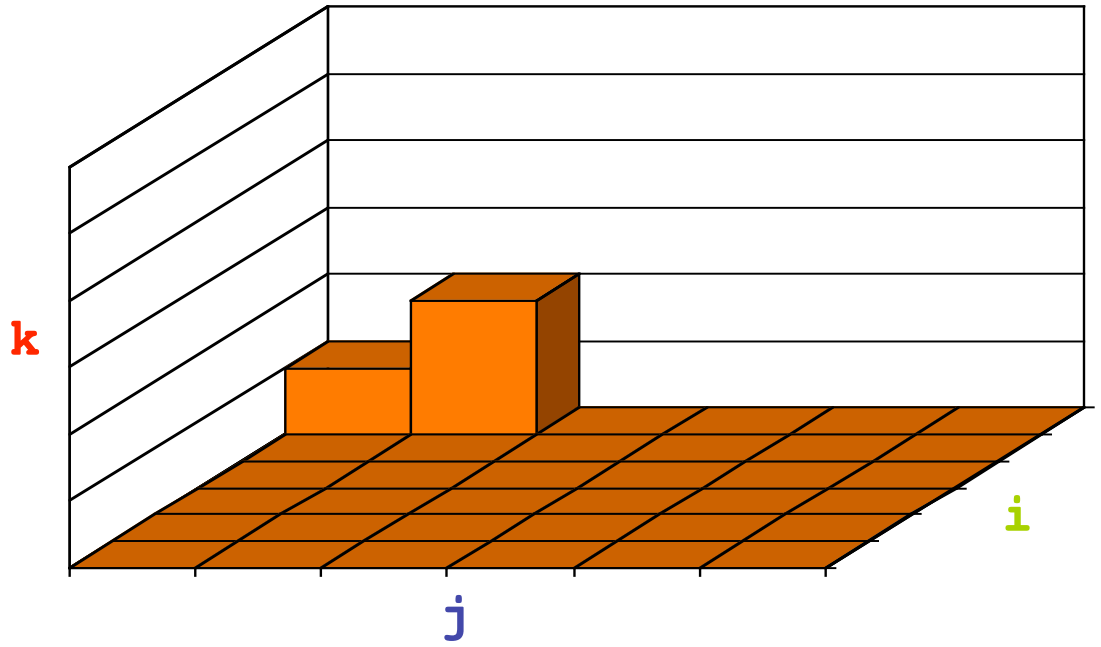
k	↓					
j		↓				
i	↓					
	–2	11	-4	13	–5	2



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;    }}
```

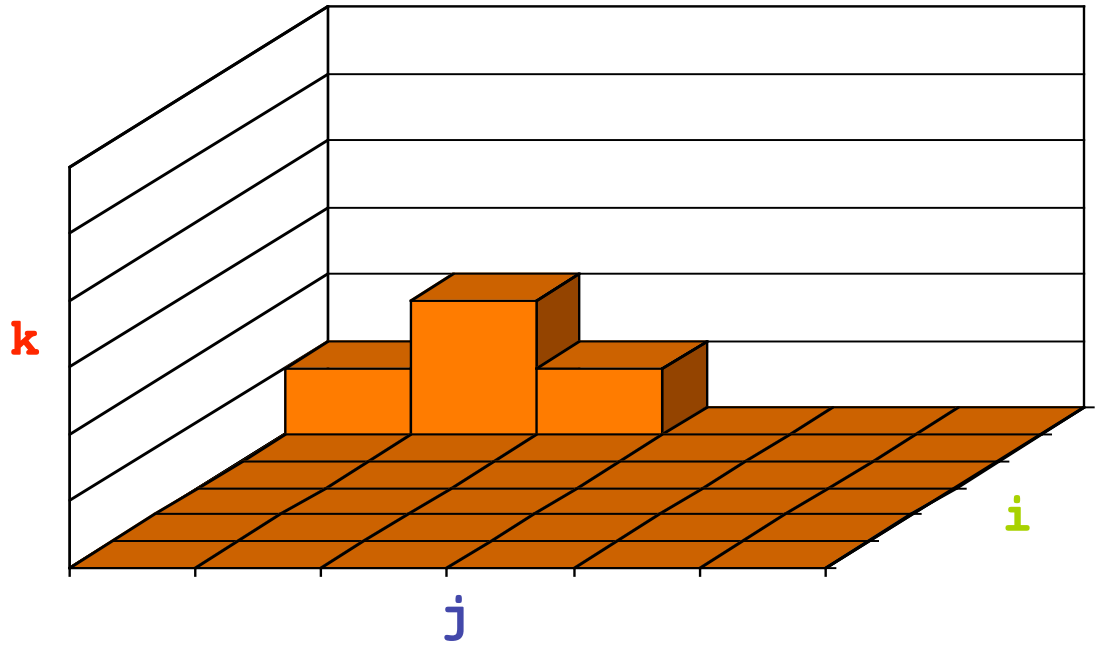
k		↓				
j		↓				
i	↓					
	–2	11	-4	13	–5	2



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;    }}
```

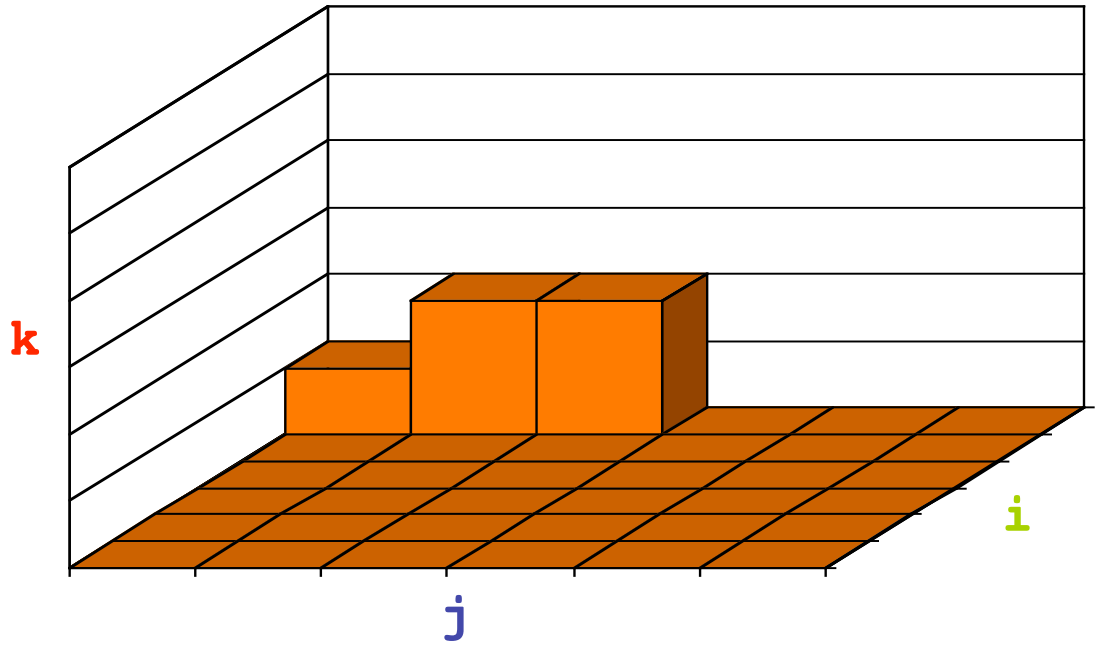
k	↓					
j			↓			
i	↓					
	–2	11	-4	13	–5	2



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;    }}
```

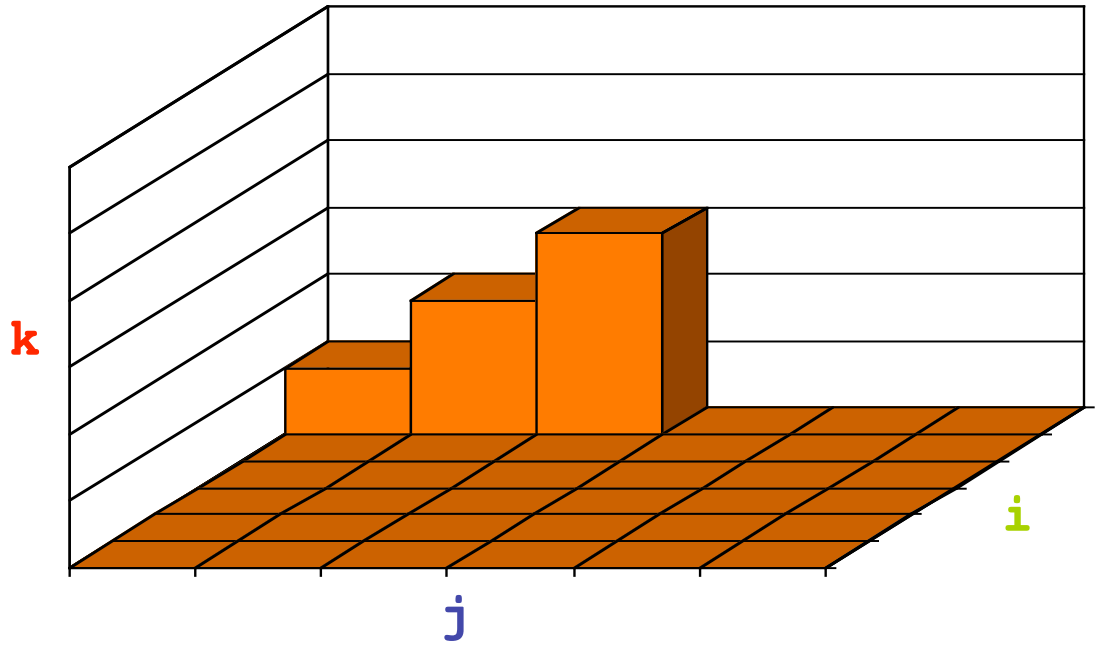
k		↓				
j			↓			
i	↓					
	–2	11	-4	13	–5	2



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;    }}
```

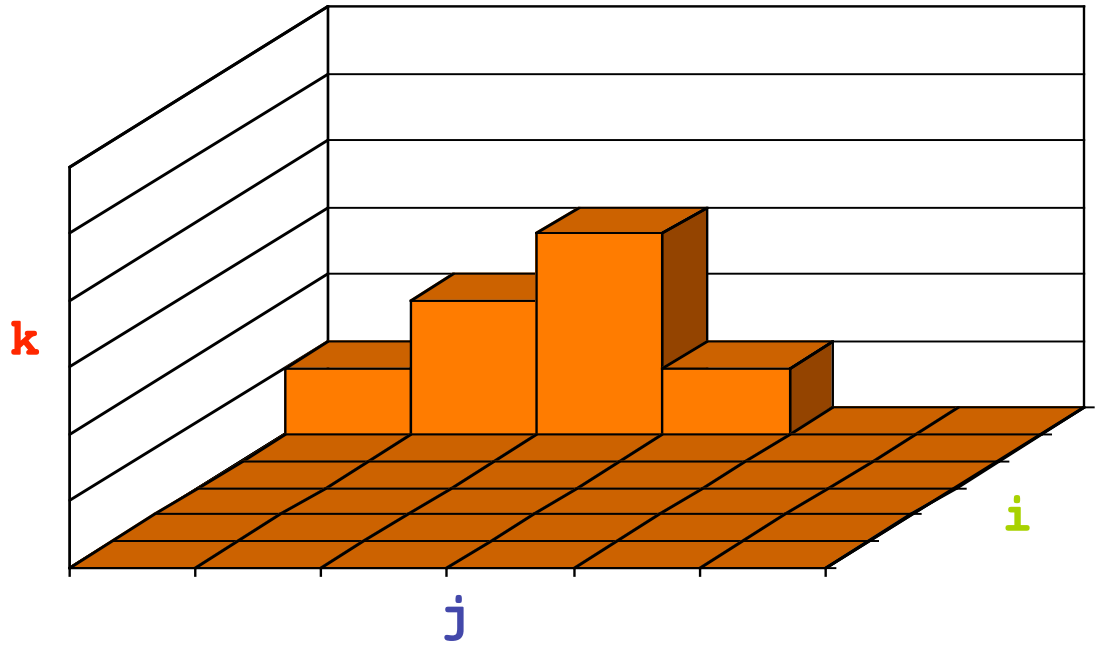
k			↓			
j			↓			
i	↓					
	–2	11	-4	13	–5	2



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;    }}
```

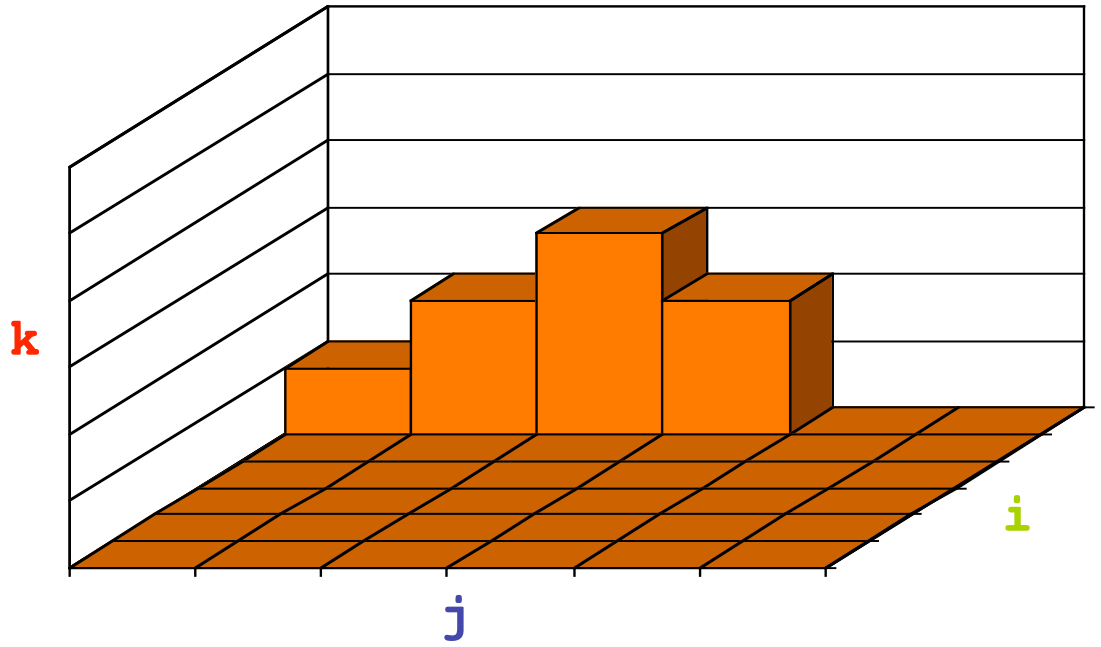
k	↓					
j				↓		
i	↓					
	–2	11	-4	13	–5	2



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;    }}
```

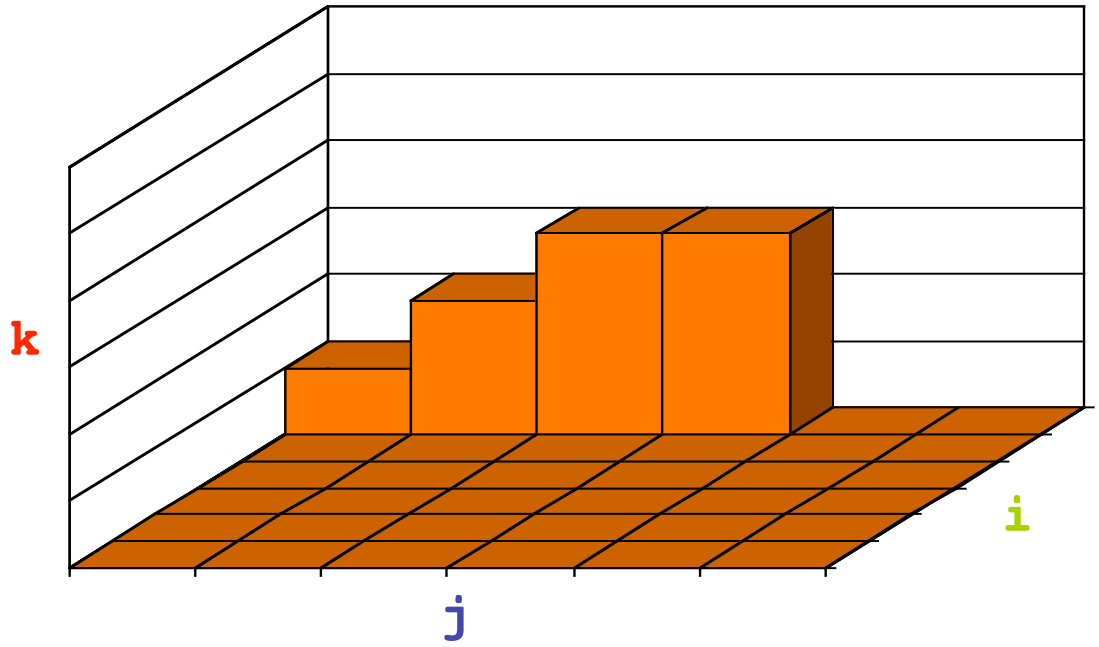
k		↓				
j				↓		
i	↓					
	–2	11	-4	13	–5	2



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;   }}
```

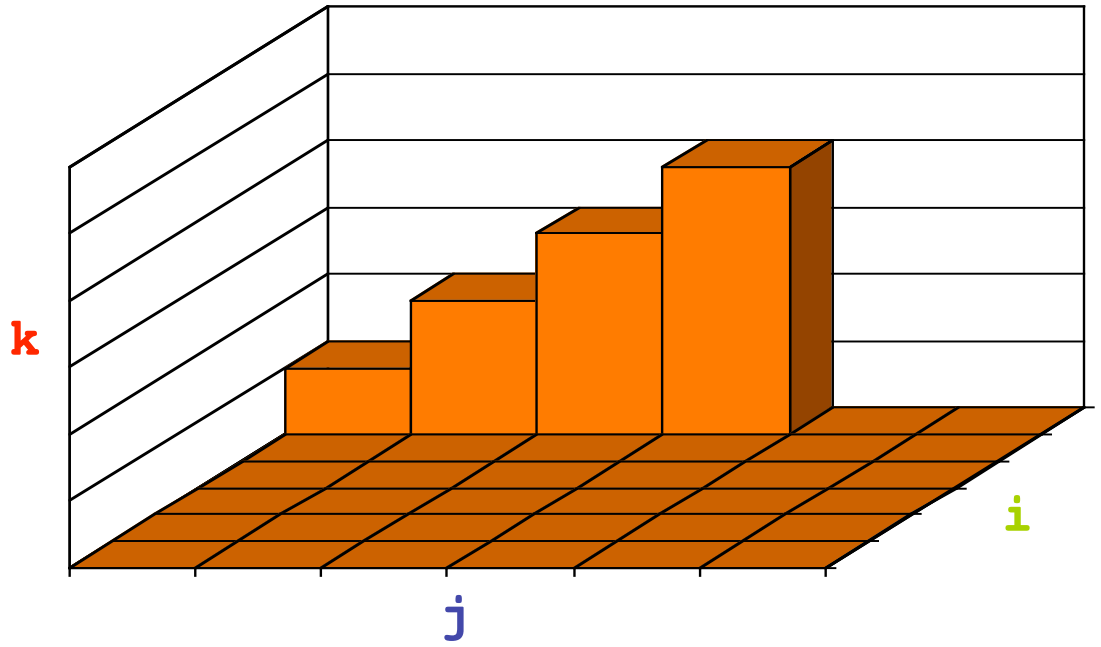
k			↓			
j				↓		
i	↓					
	–2	11	-4	13	–5	2



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;    }}
```

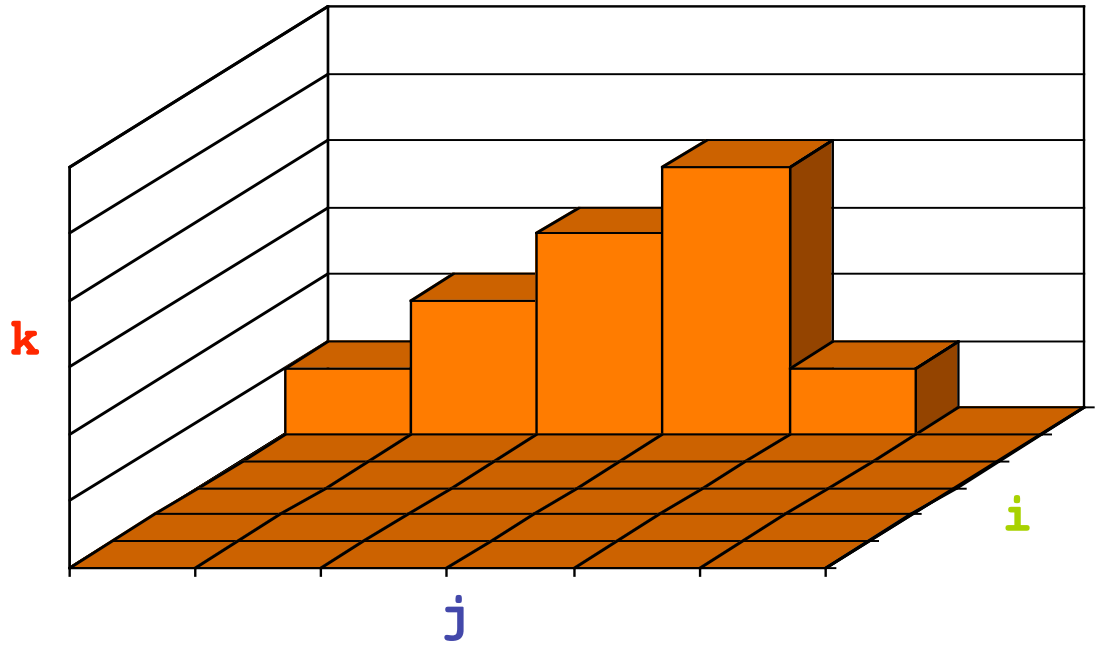
k				↓		
j				↓		
i	↓					
	–2	11	-4	13	–5	2



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;    }}
```

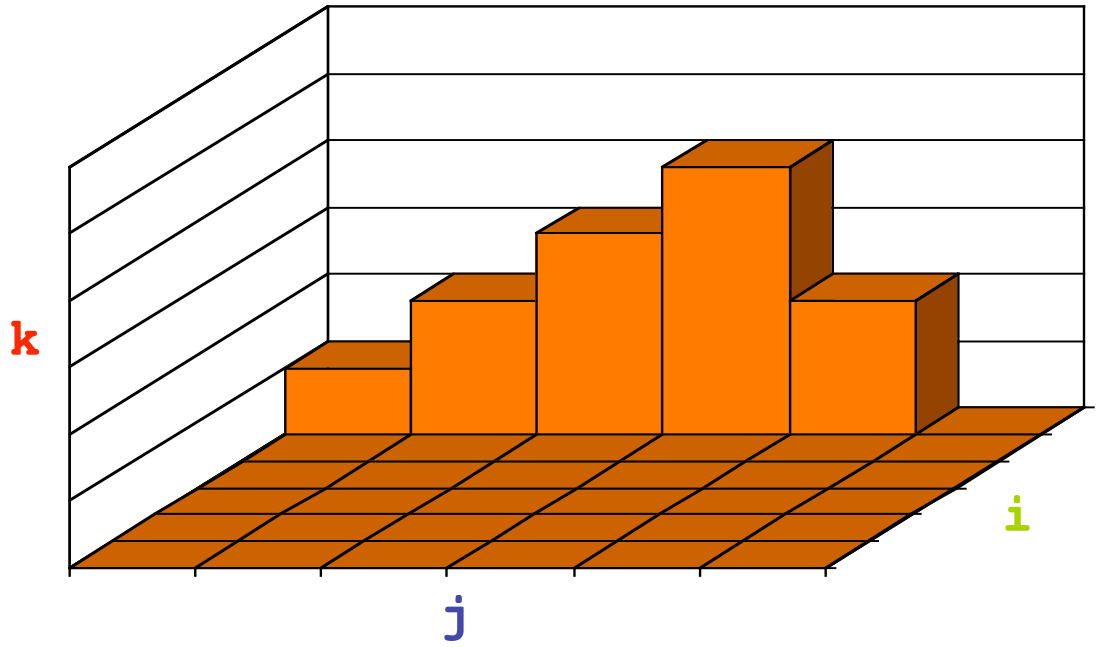
k	↓					
j					↓	
i	↓					
	–2	11	-4	13	–5	2



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;    }}
```

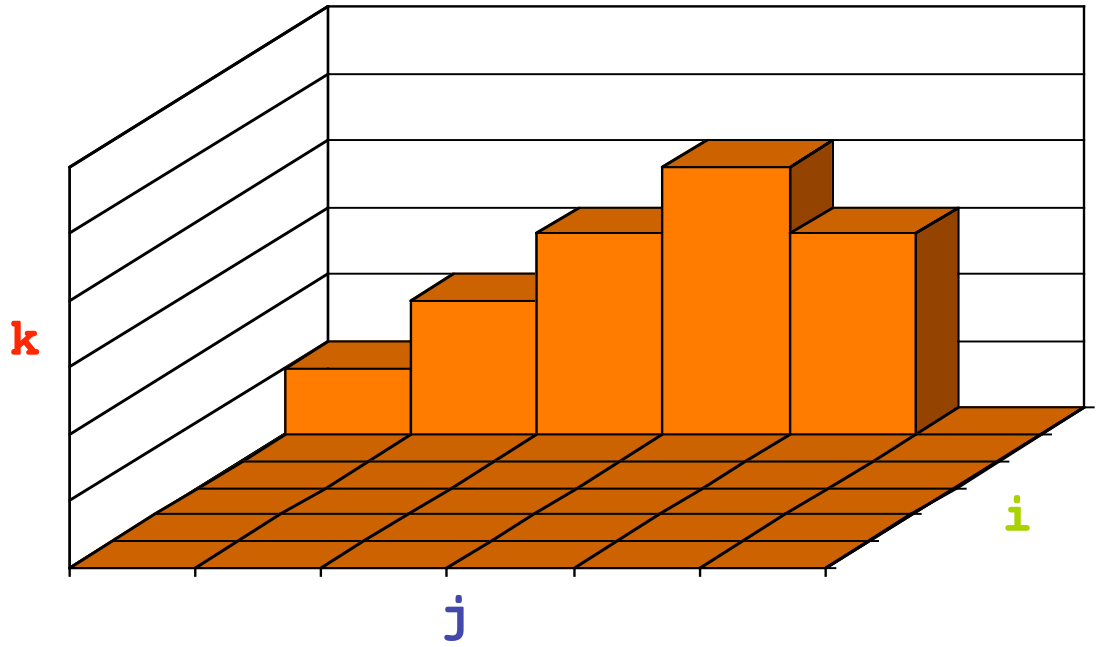
k		↓				
j					↓	
i	↓					
	–2	11	-4	13	–5	2



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;    }}
```

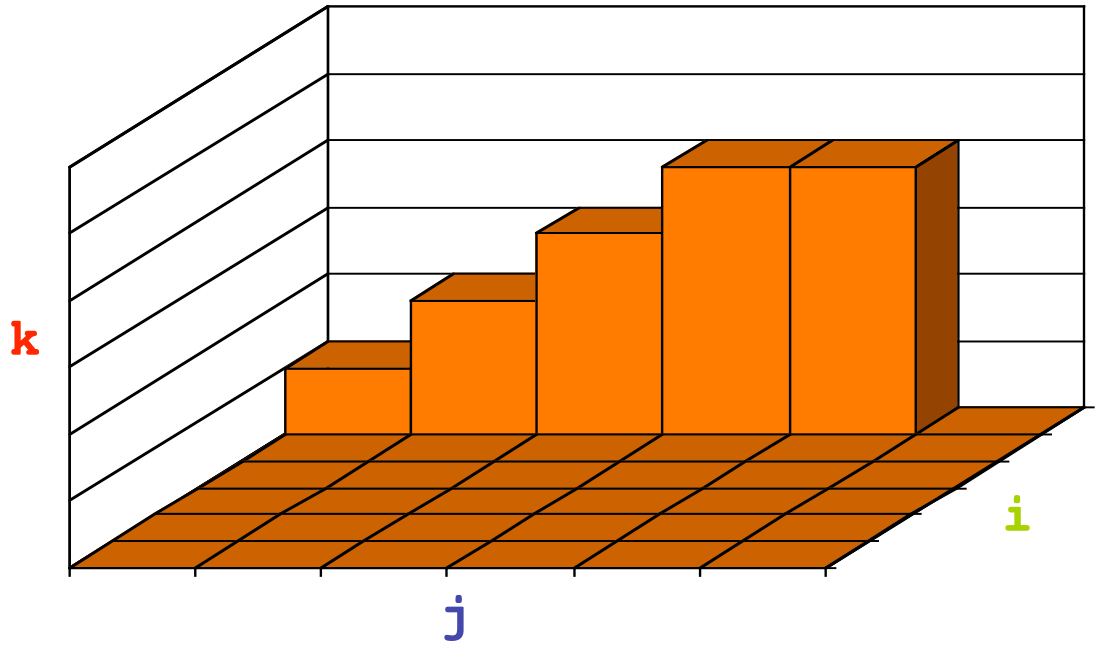
k			↓			
j					↓	
i	↓					
	–2	11	-4	13	–5	2



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;    }}
```

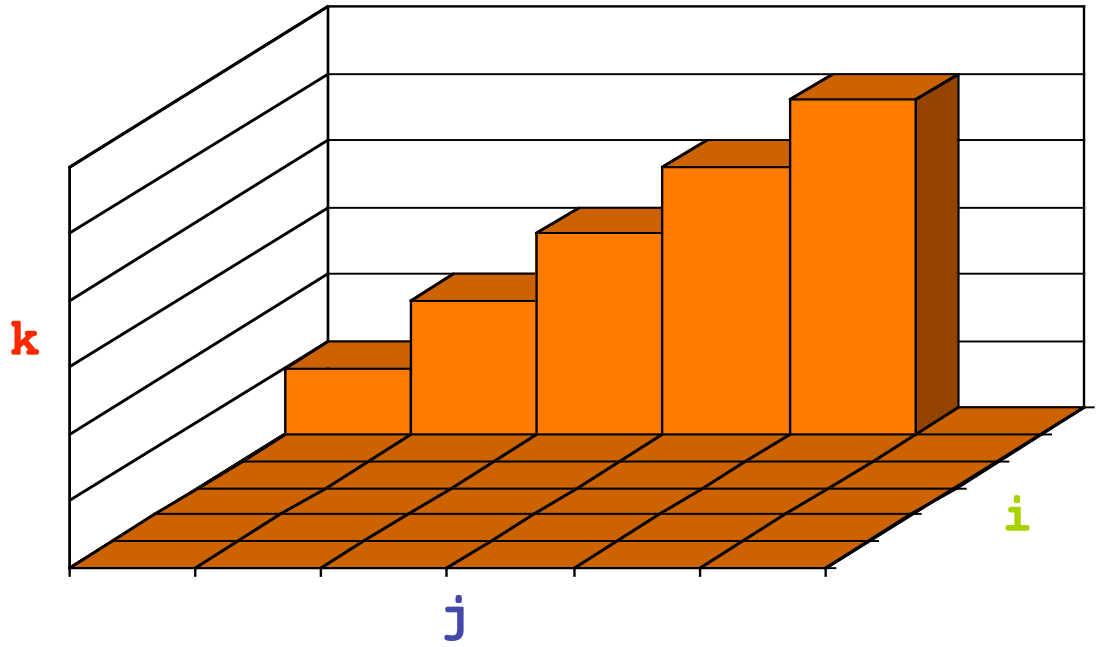
k				↓		
j					↓	
i	↓					
	–2	11	-4	13	–5	2



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;    }}
```

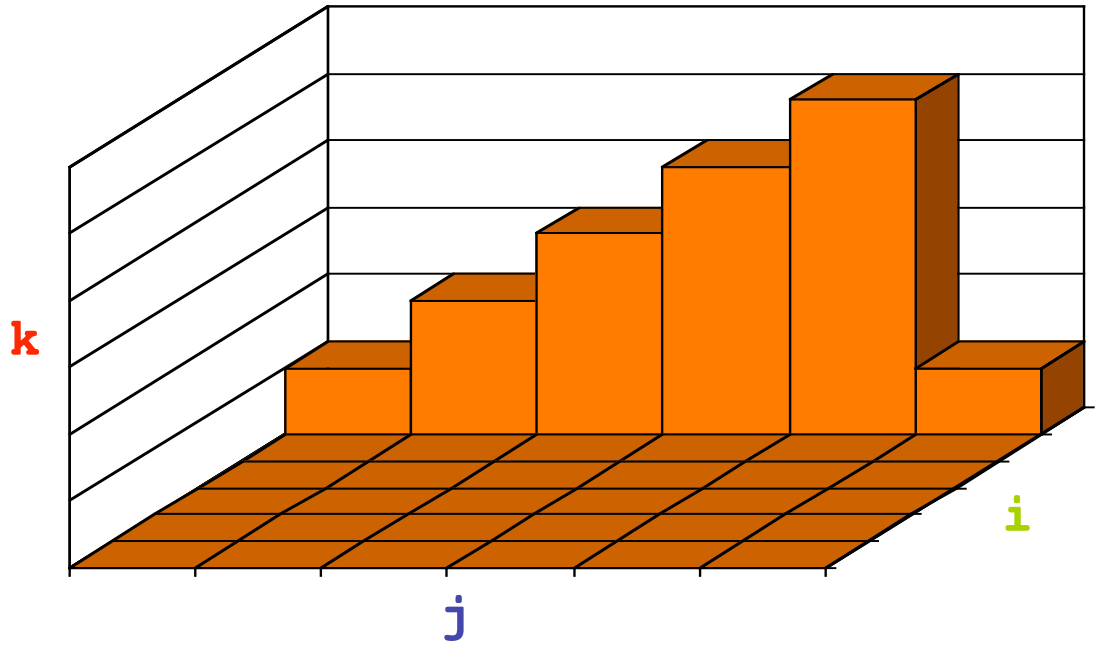
k					↓	
j					↓	
i	↓					
	–2	11	-4	13	–5	2



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;    }}
```

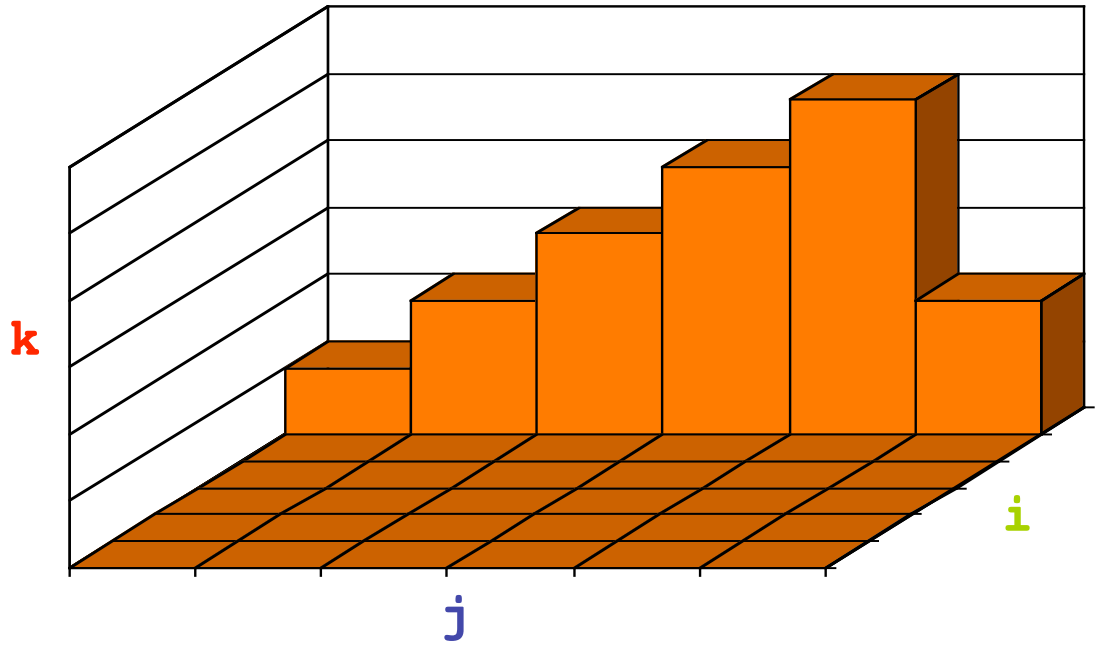
k	↓					
j						↓
i	↓					
	–2	11	-4	13	–5	2



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;    }}
```

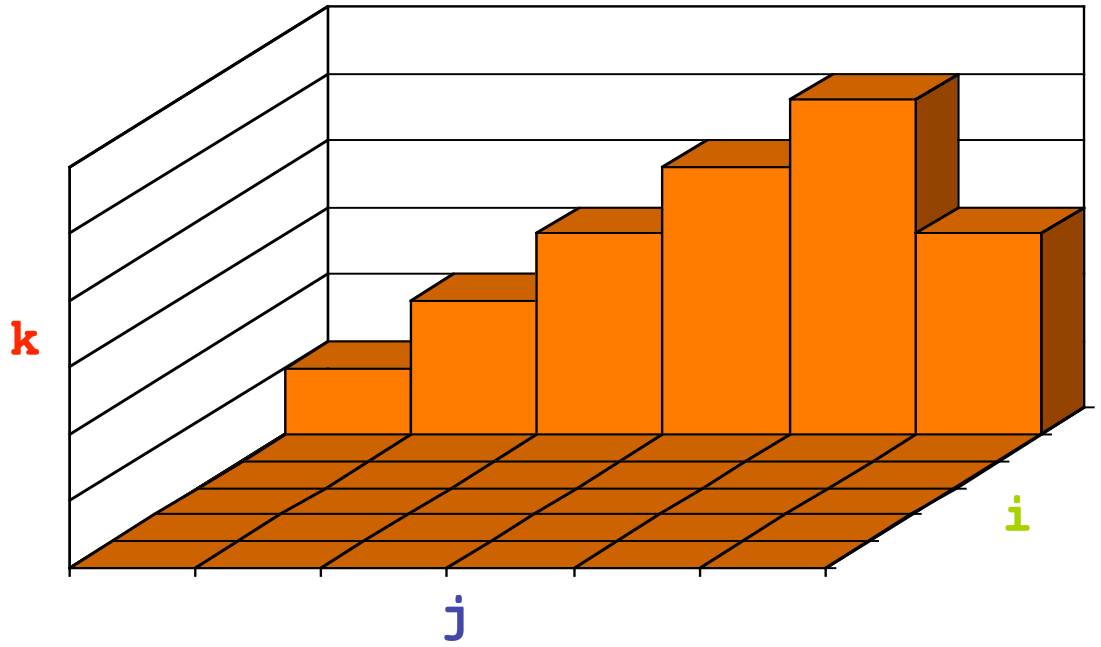
k		↓				
j						↓
i	↓					
	–2	11	-4	13	–5	2



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;    }}
```

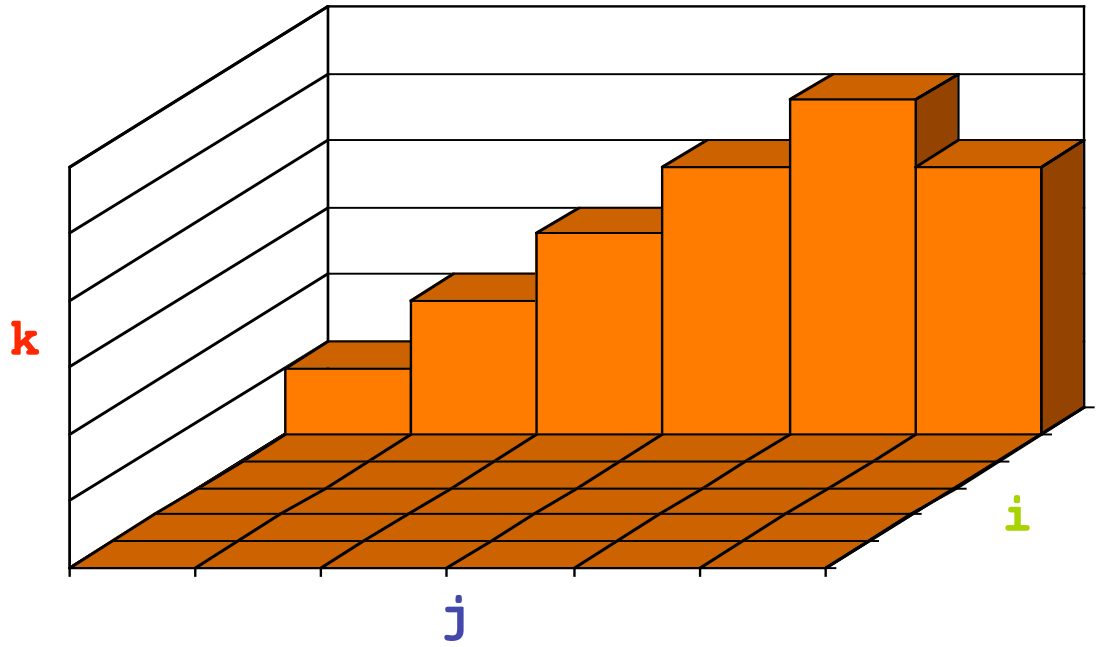
k			↓			
j						↓
i	↓					
	–2	11	-4	13	–5	2



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;    }}
```

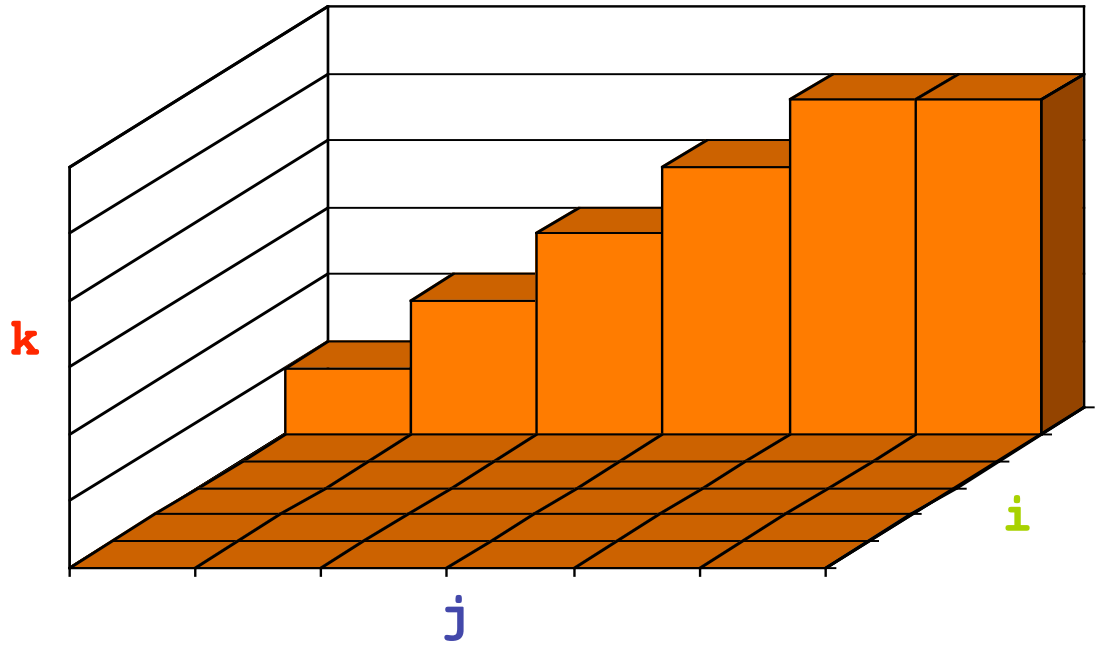
k				↓		
j						↓
i	↓					
	–2	11	-4	13	–5	2



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;    }}
```

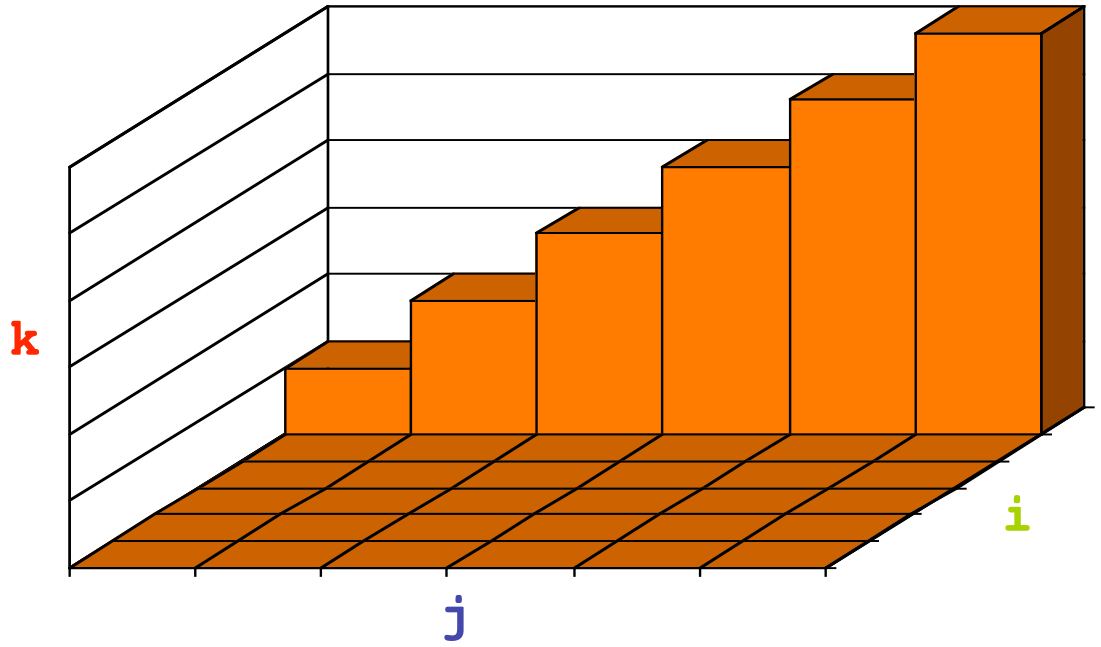
k					↓	
j						↓
i	↓					
	–2	11	-4	13	–5	2



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;    }}
```

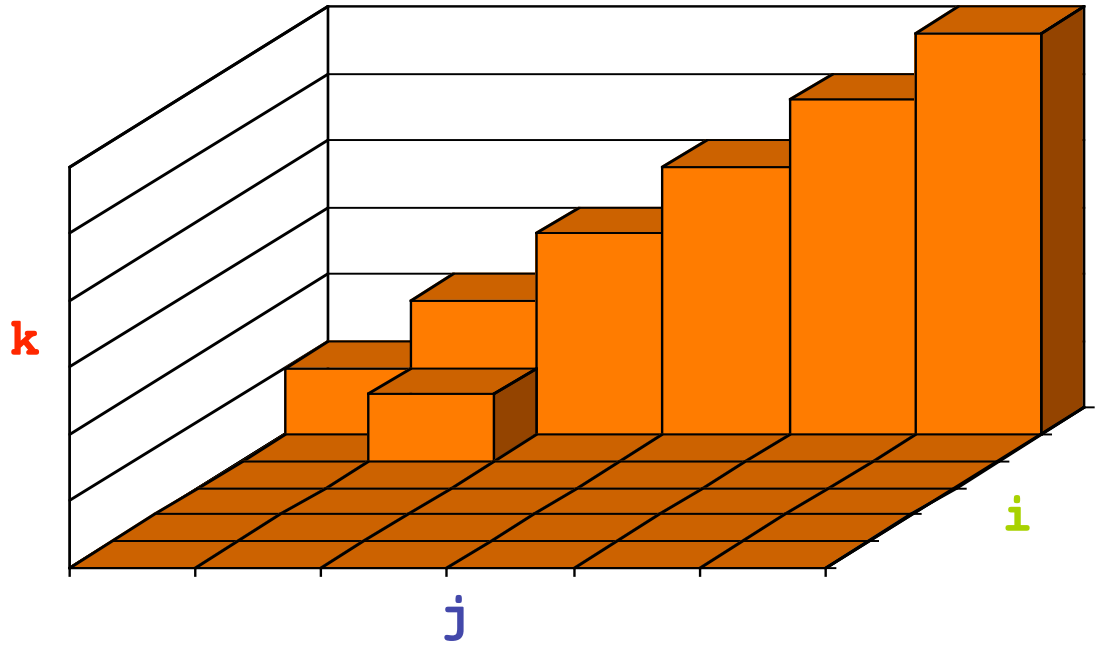
k						↓
j						↓
i	↓					
	–2	11	-4	13	–5	2



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;   }}
```

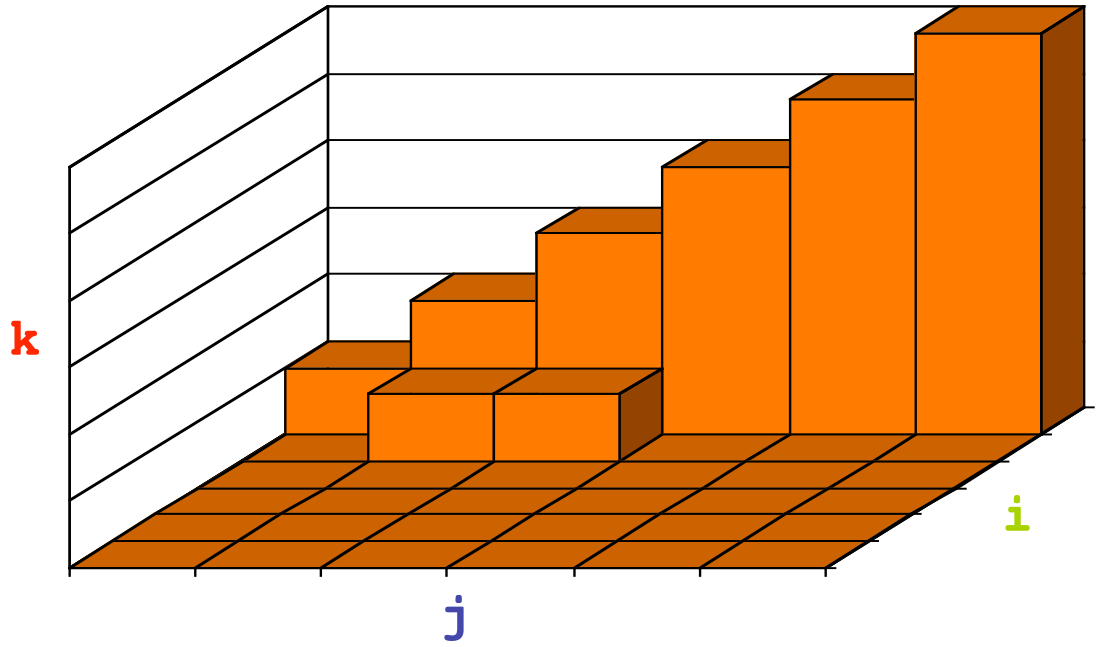
k		↓				
j		↓				
i		↓				
	–2	11	-4	13	–5	2



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;    }}
```

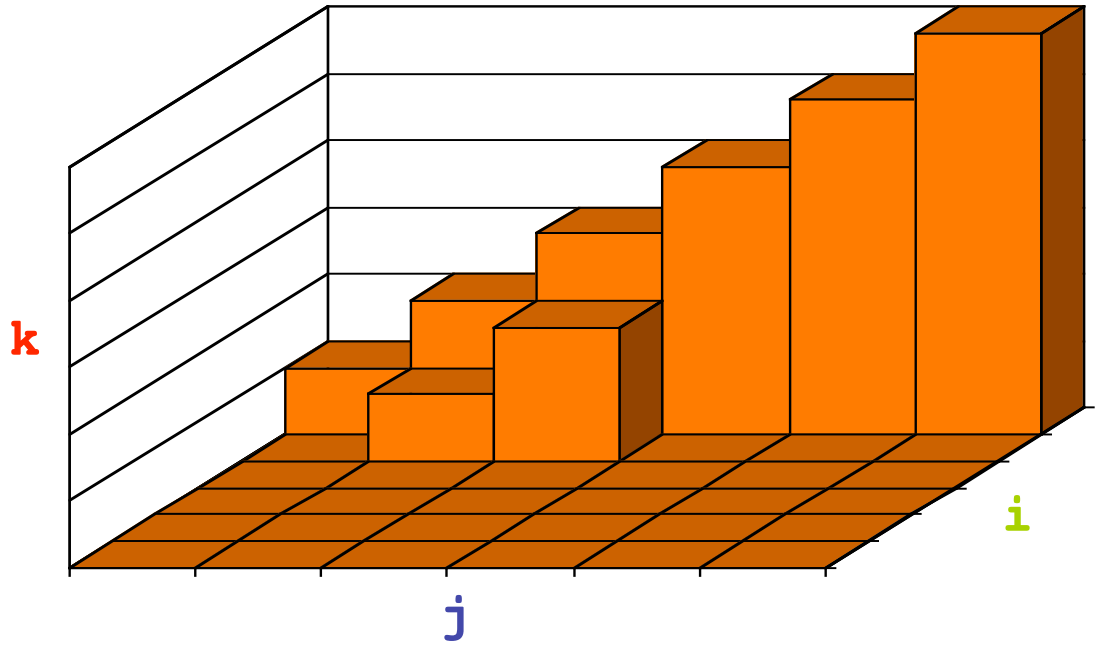
k		↓				
j			↓			
i		↓				
	–2	11	-4	13	–5	2



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;    }}
```

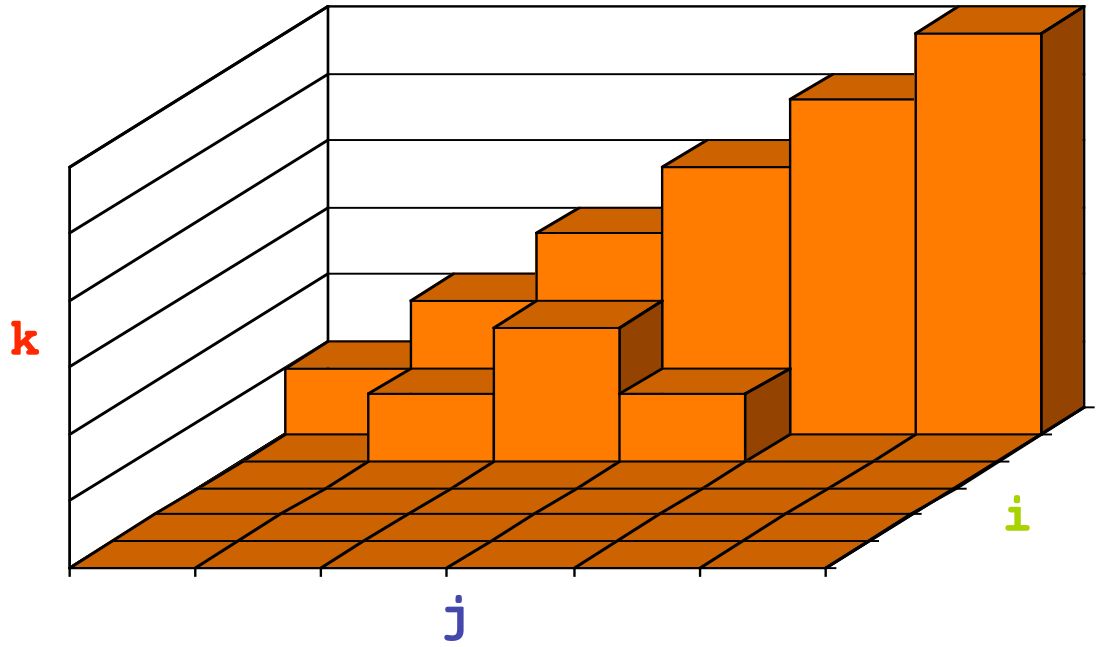
k			↓			
j			↓			
i		↓				
	–2	11	-4	13	–5	2



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;    }}
```

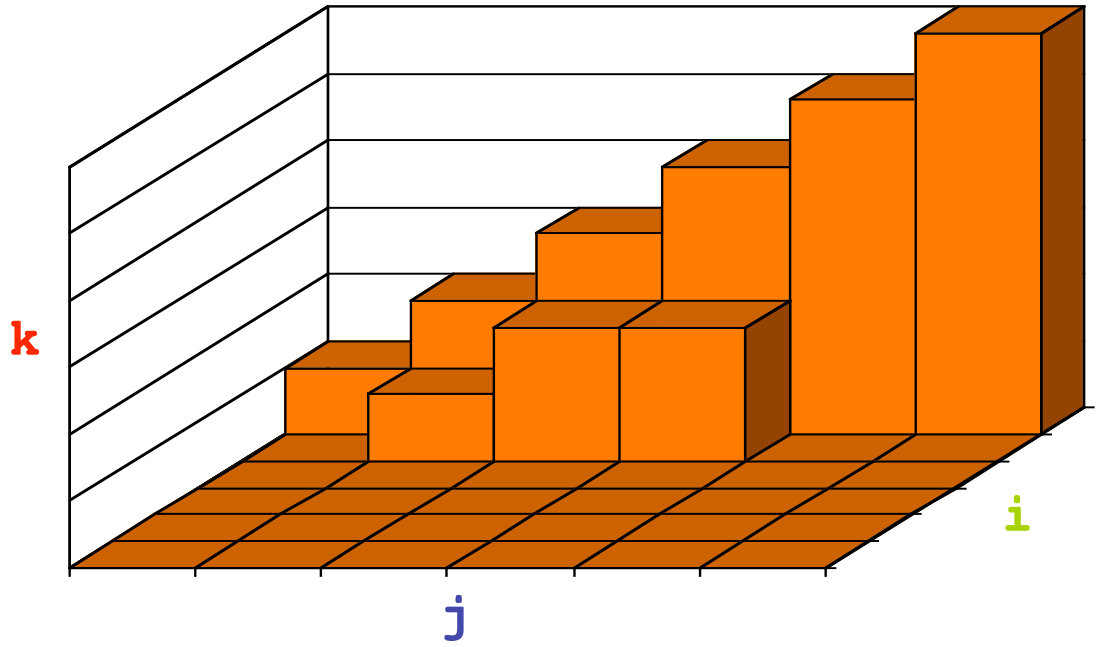
k		↓				
j				↓		
i		↓				
	–2	11	-4	13	–5	2



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;    }}
```

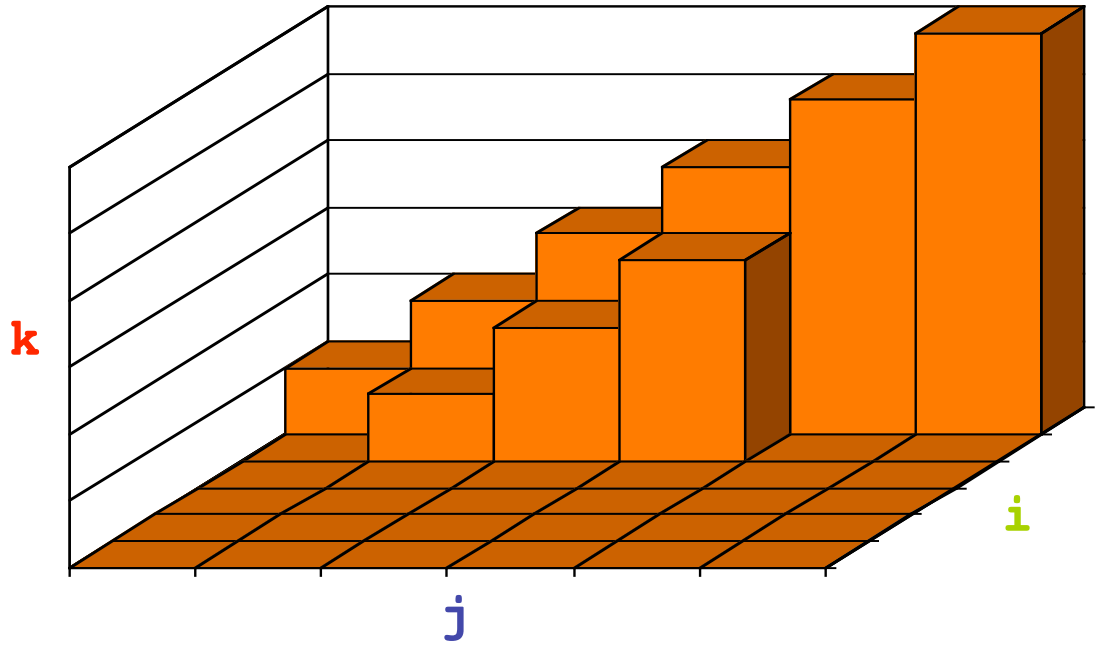
k			↓			
j				↓		
i		↓				
	–2	11	-4	13	–5	2



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;    }}
```

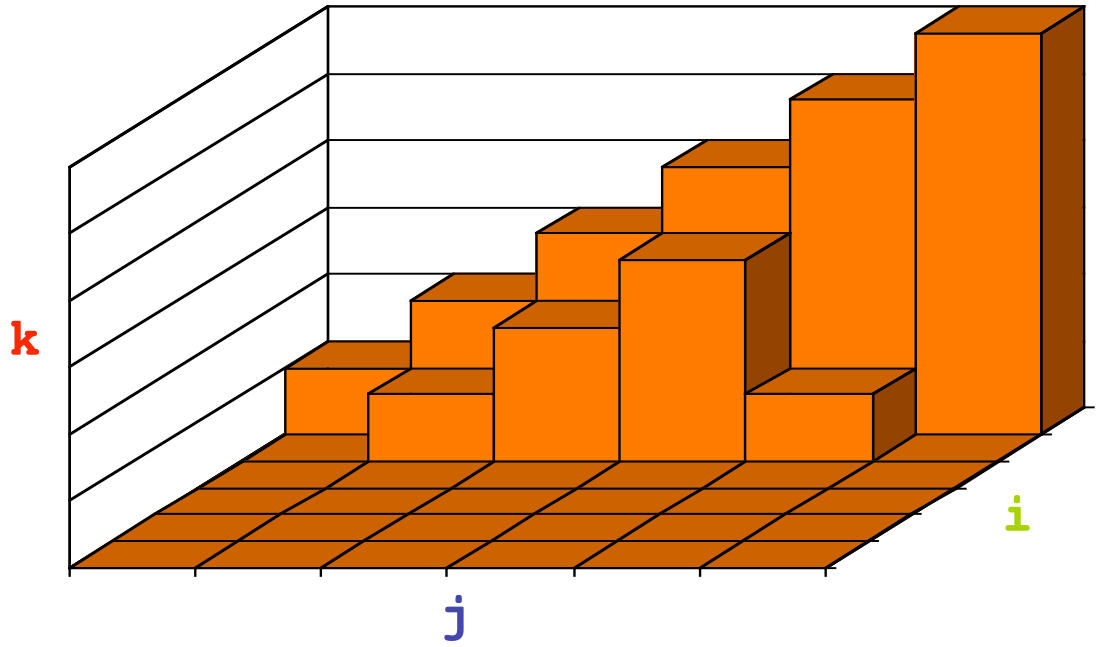
k				↓		
j				↓		
i		↓				
	–2	11	-4	13	–5	2



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;    }}
```

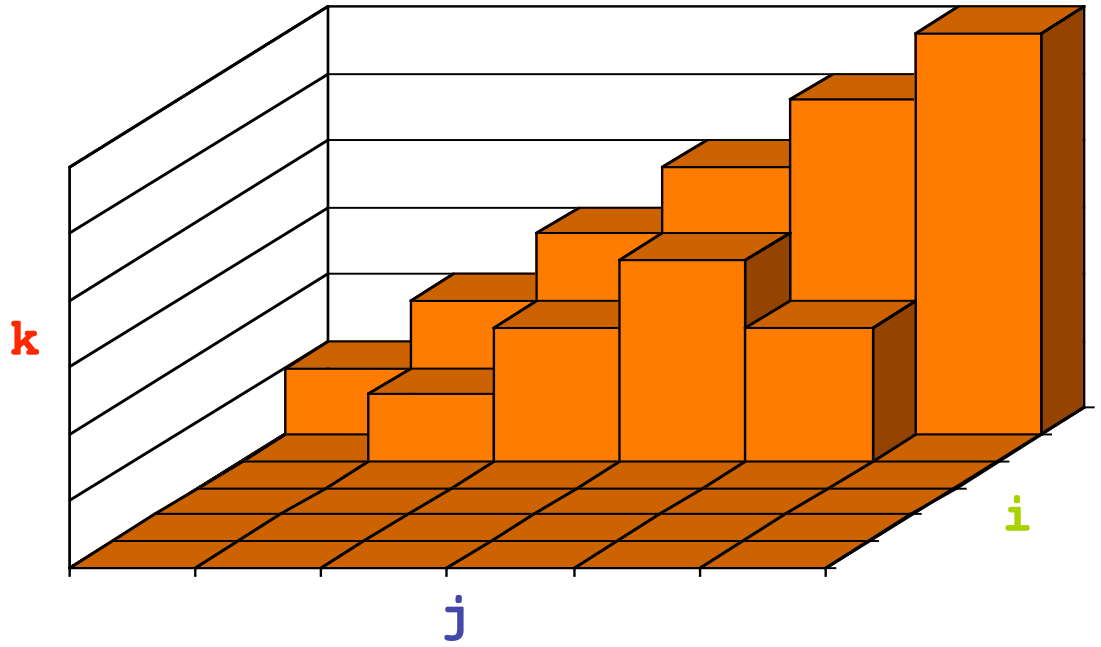
k		↓				
j					↓	
i		↓				
	–2	11	-4	13	–5	2



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;    }}
```

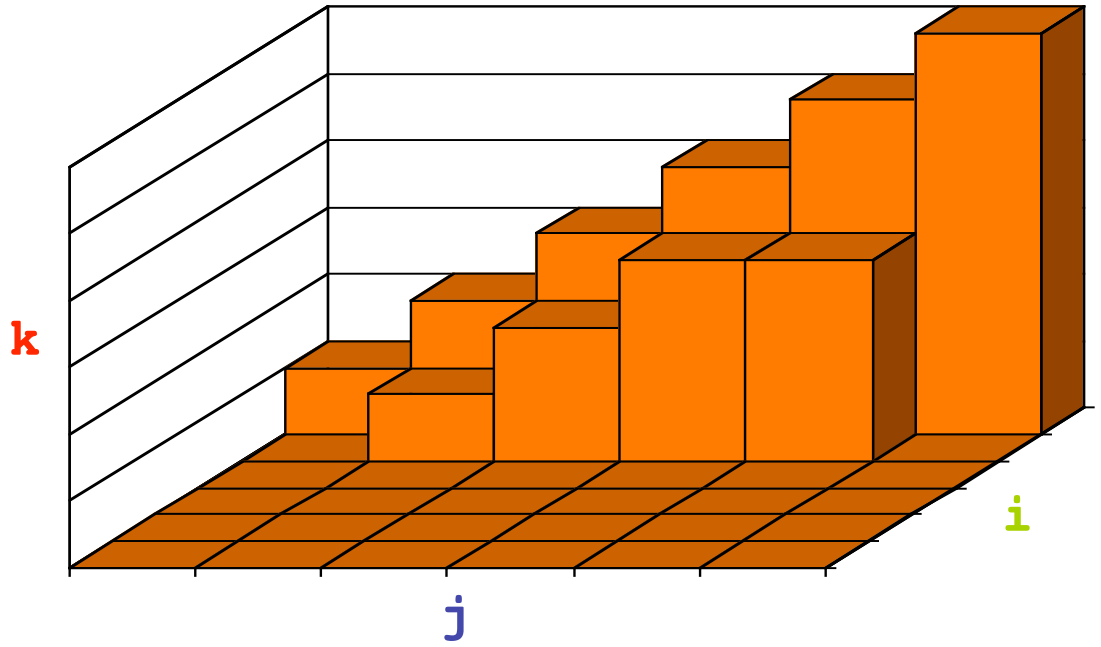
k			↓			
j					↓	
i		↓				
	–2	11	-4	13	–5	2



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;    }}
```

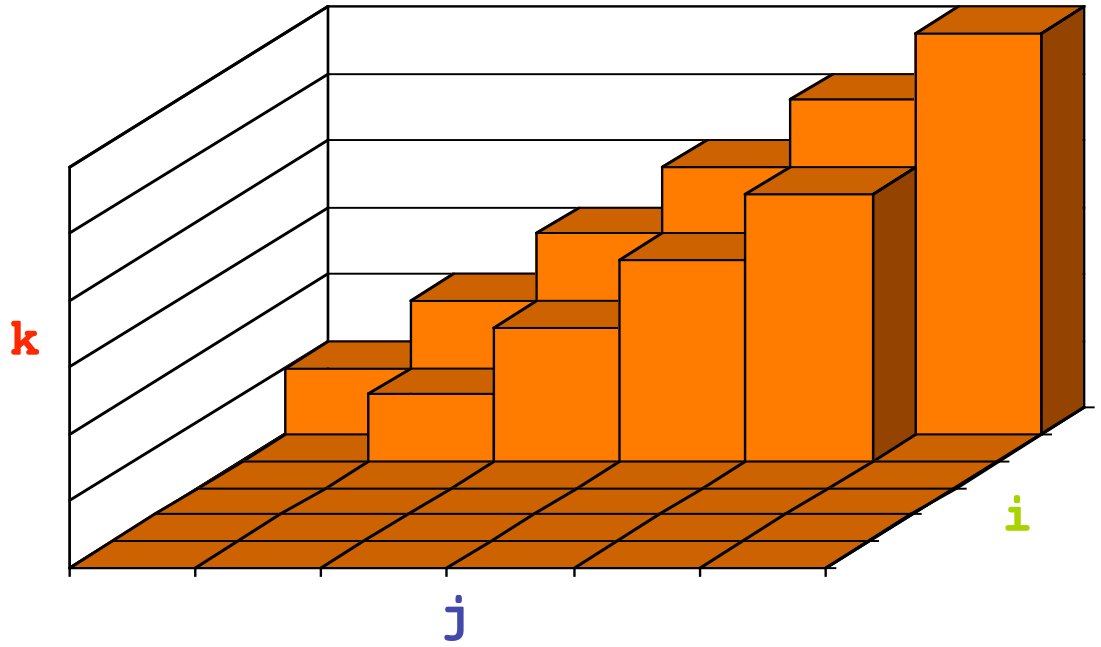
k				↓		
j					↓	
i		↓				
	–2	11	-4	13	–5	2



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;    }}
```

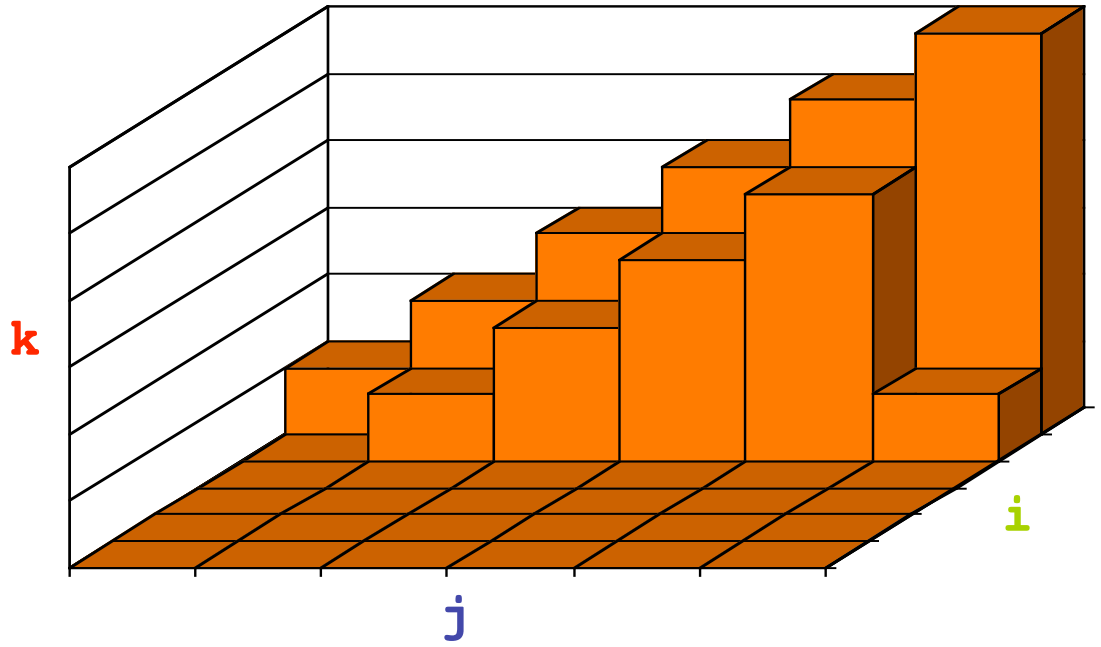
k					↓	
j					↓	
i		↓				
	–2	11	-4	13	–5	2



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;    }}
```

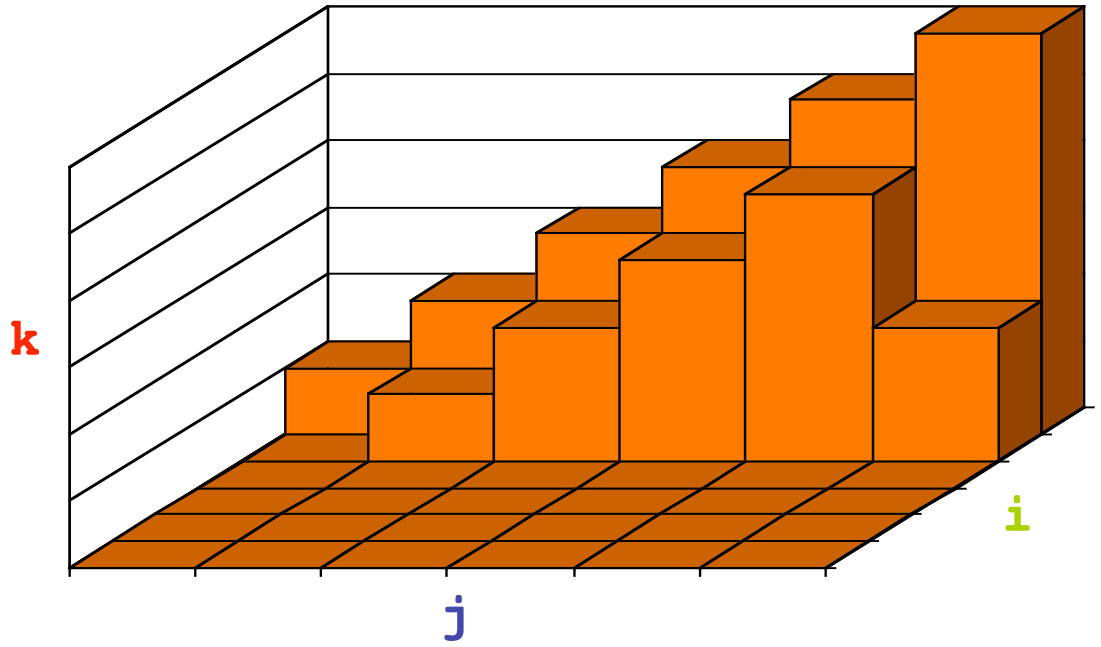
k		↓				
j						↓
i		↓				
	–2	11	-4	13	–5	2



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;    }}
```

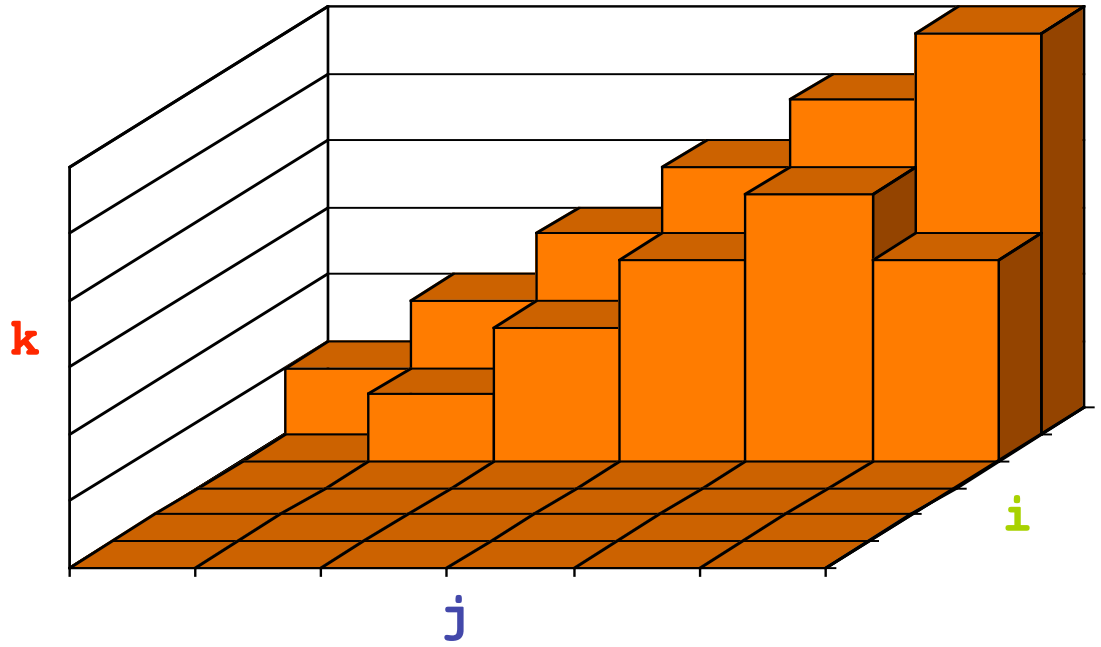
k			↓			
j						↓
i		↓				
	–2	11	-4	13	–5	2



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;    }}
```

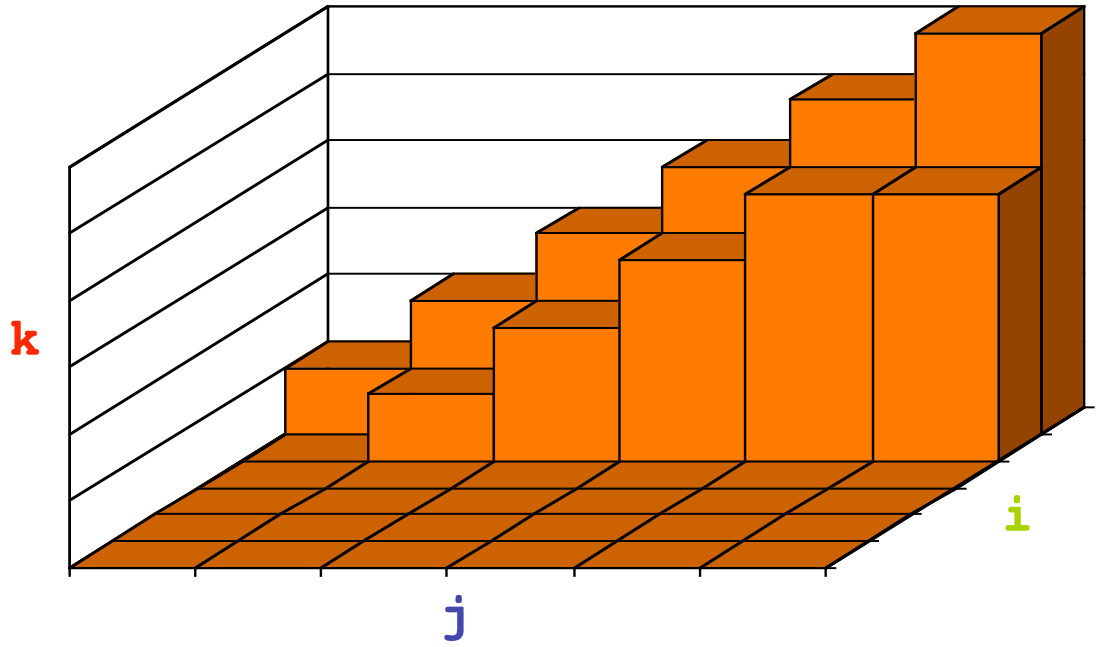
k				↓		
j						↓
i		↓				
	–2	11	-4	13	–5	2



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;   }}
```

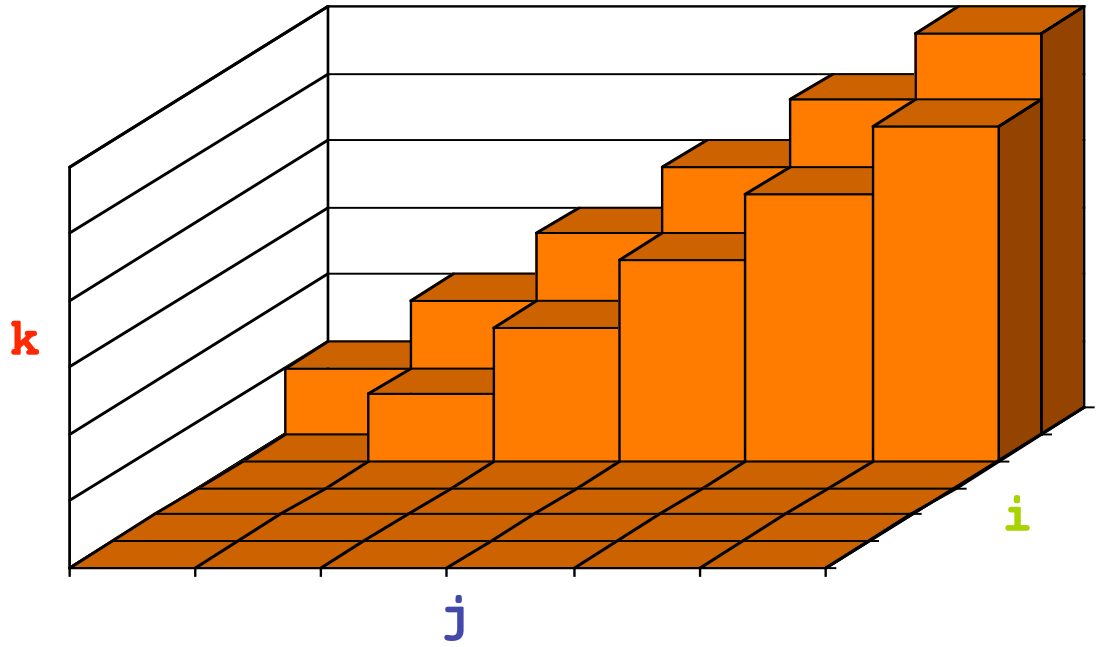
k					↓	
j						↓
i		↓				
	–2	11	-4	13	–5	2



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;    }}
```

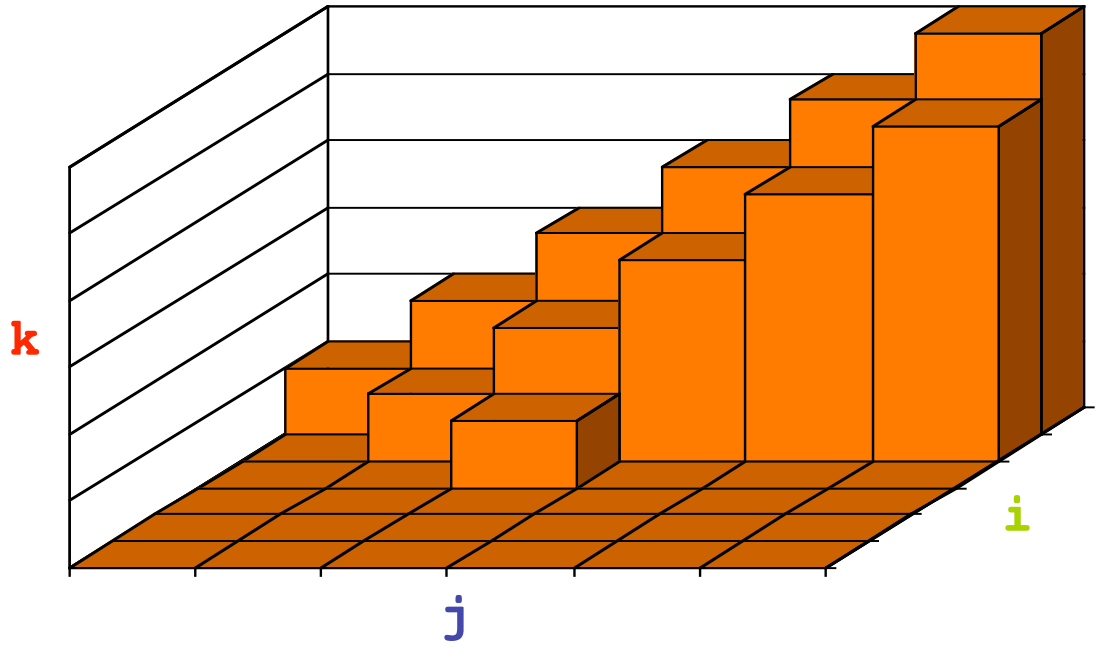
k						↓
j						↓
i		↓				
	–2	11	-4	13	–5	2



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;    }}
```

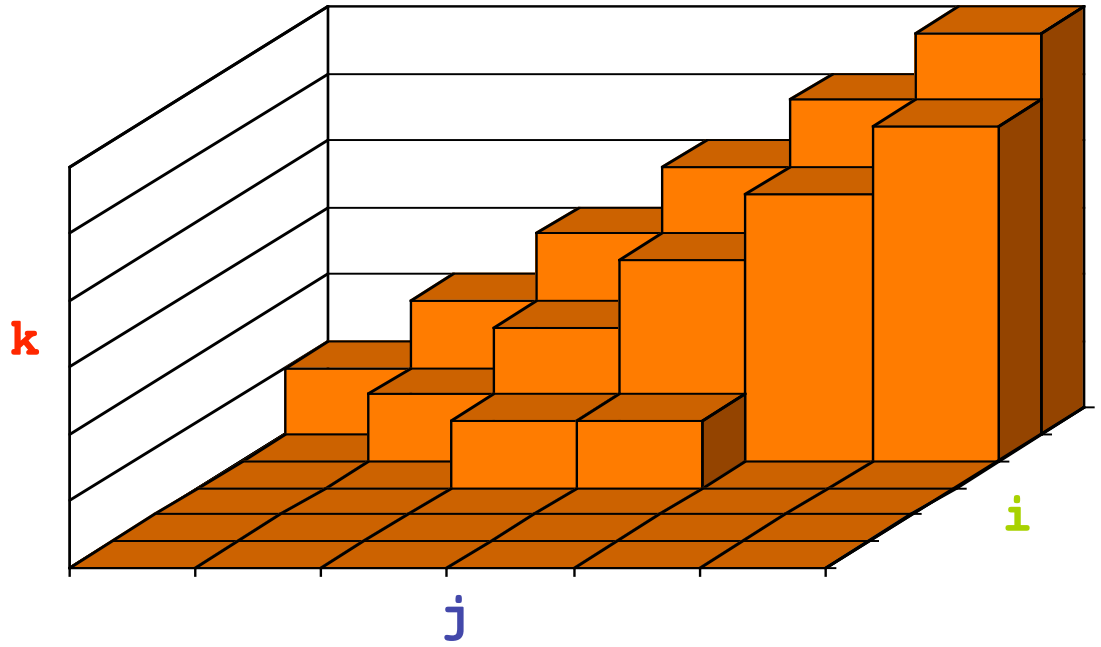
k			↓			
j			↓			
i			↓			
	–2	11	-4	13	–5	2



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;    }}
```

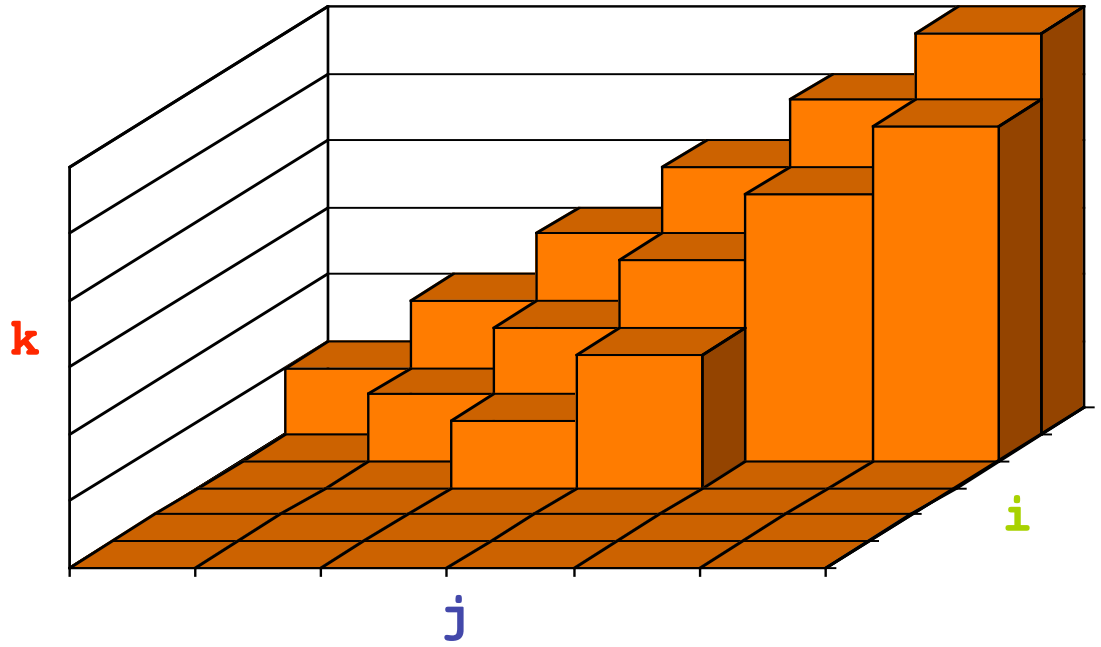
k			↓			
j				↓		
i			↓			
	–2	11	-4	13	–5	2



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;    }}
```

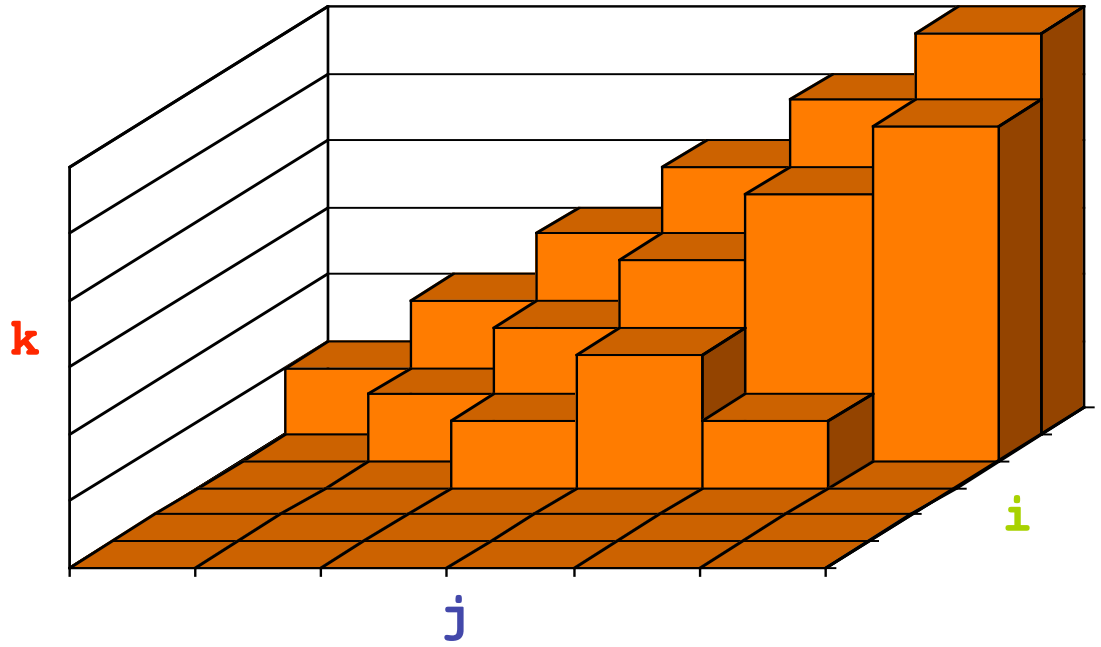
k				↓		
j				↓		
i			↓			
	–2	11	-4	13	–5	2



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;    }}
```

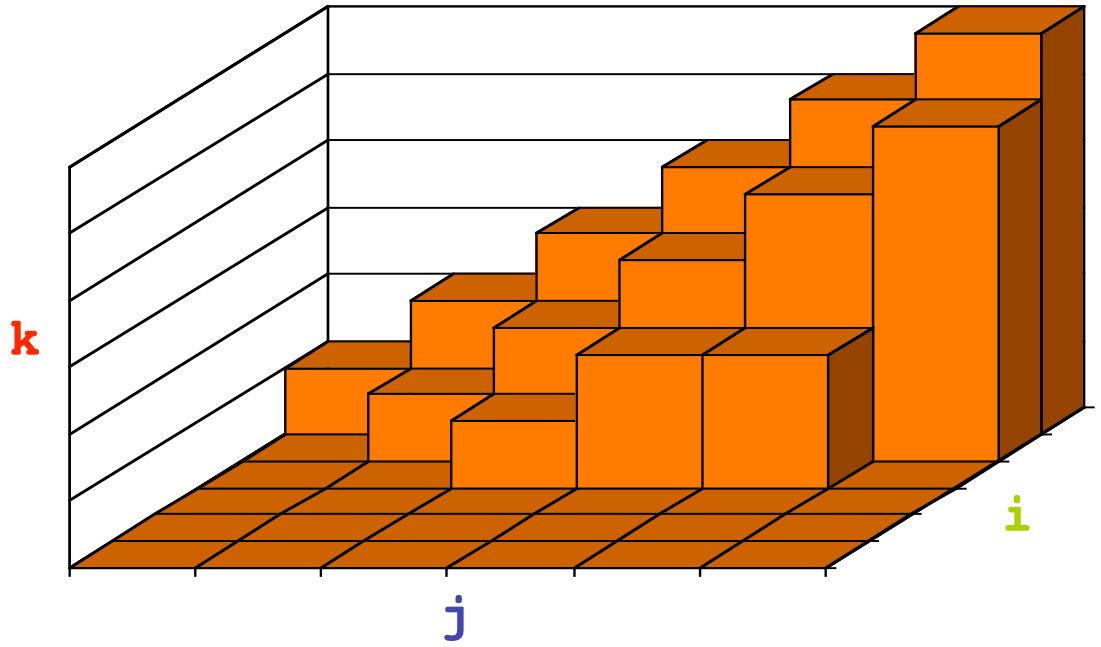
k			↓			
j					↓	
i			↓			
	–2	11	-4	13	–5	2



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;    }}
```

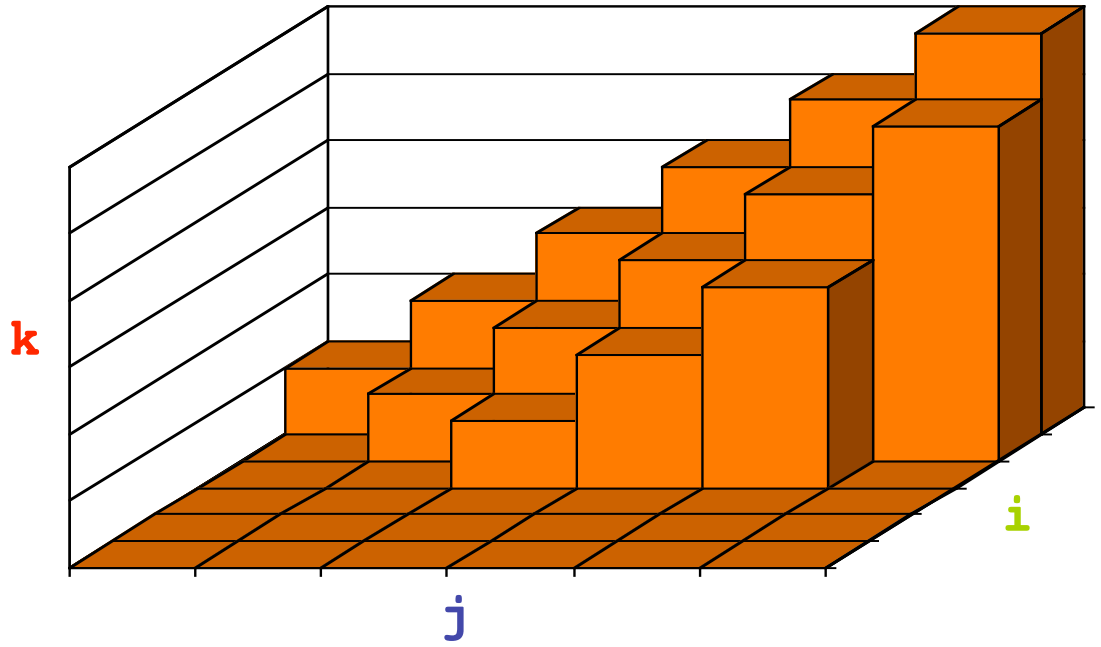
k				↓		
j					↓	
i			↓			
	–2	11	-4	13	–5	2



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;    }}
```

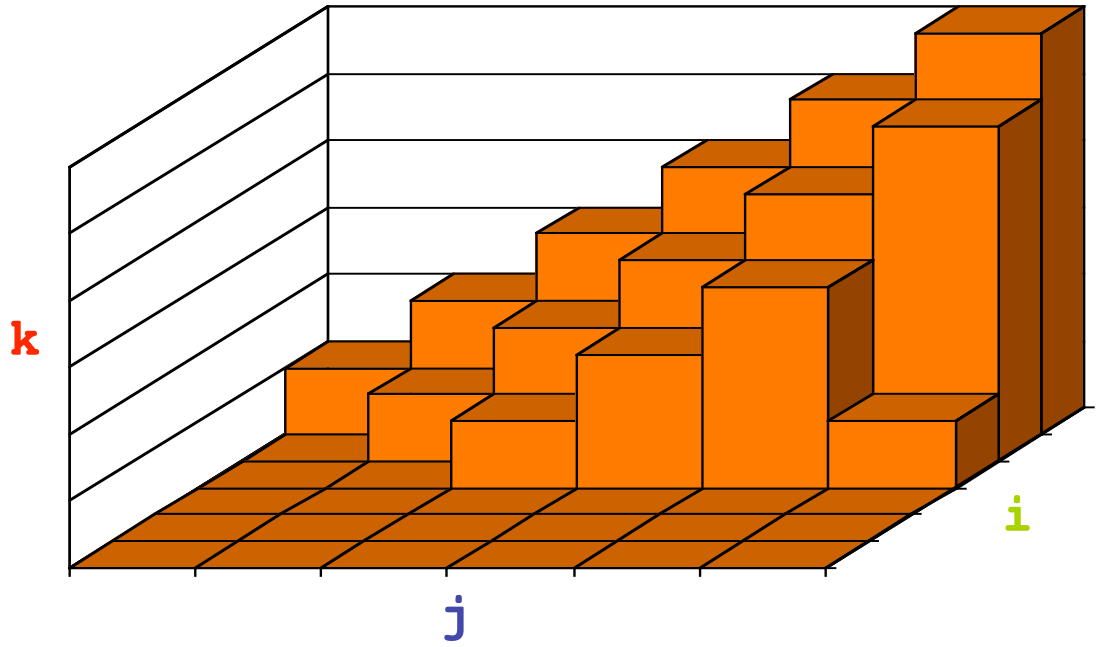
k					↓	
j					↓	
i			↓			
	–2	11	-4	13	–5	2



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;    }}
```

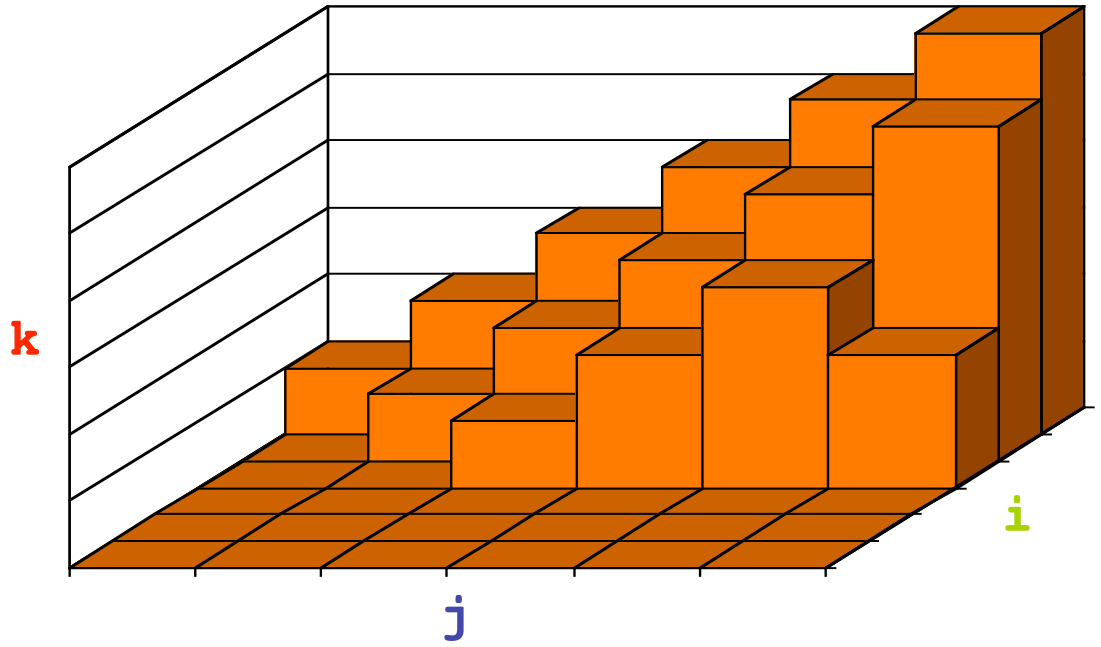
k			↓			
j						↓
i			↓			
	–2	11	-4	13	–5	2



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;    }}
```

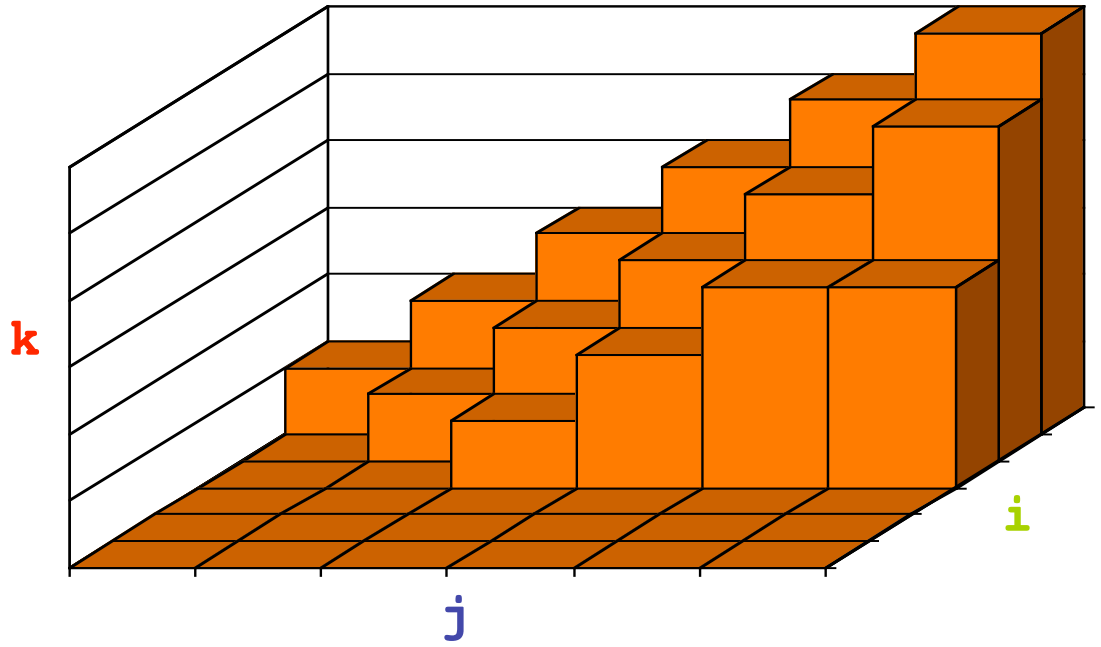
k				↓		
j						↓
i			↓			
	–2	11	-4	13	–5	2



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;    }}
```

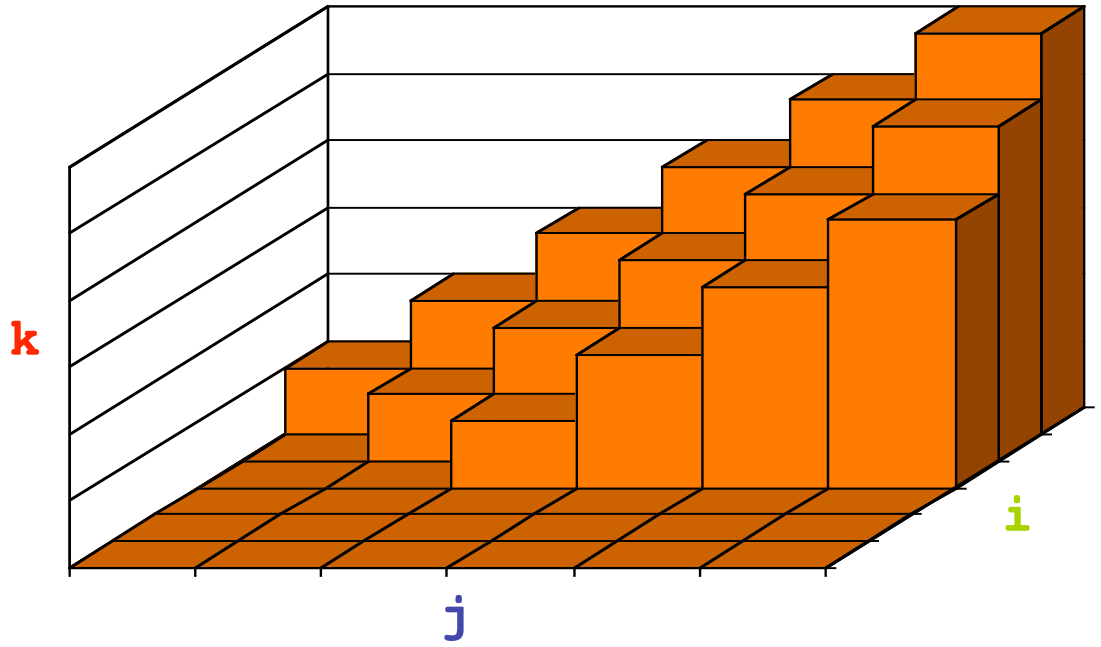
k					↓	
j						↓
i			↓			
	–2	11	-4	13	–5	2



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;    }}
```

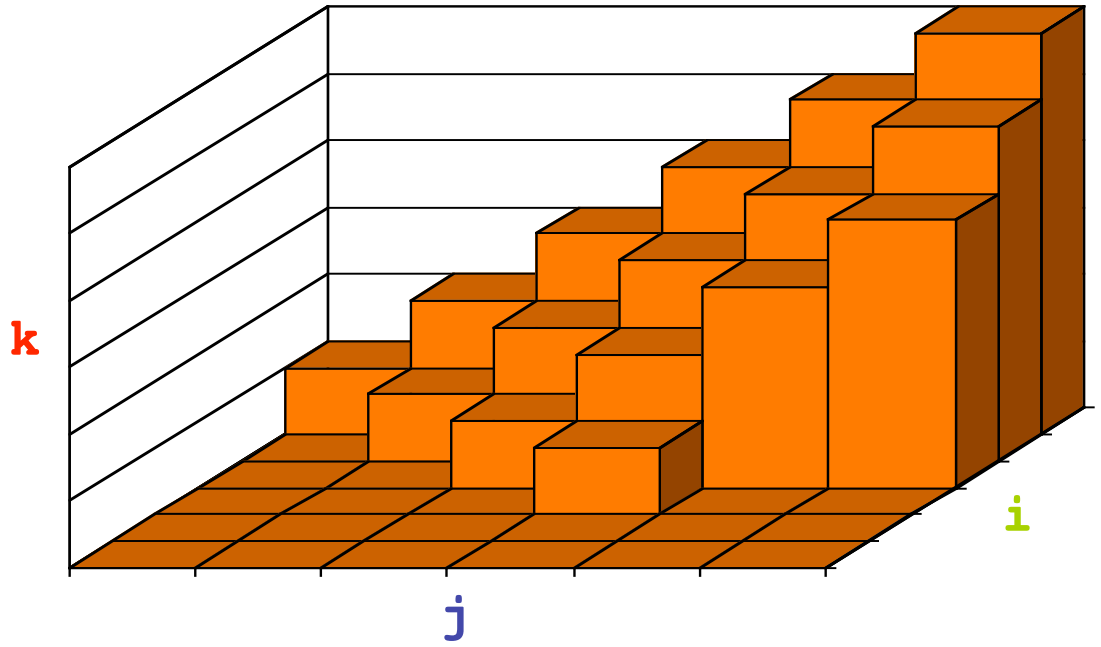
k						↓
j						↓
i			↓			
	–2	11	-4	13	–5	2



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;    }}
```

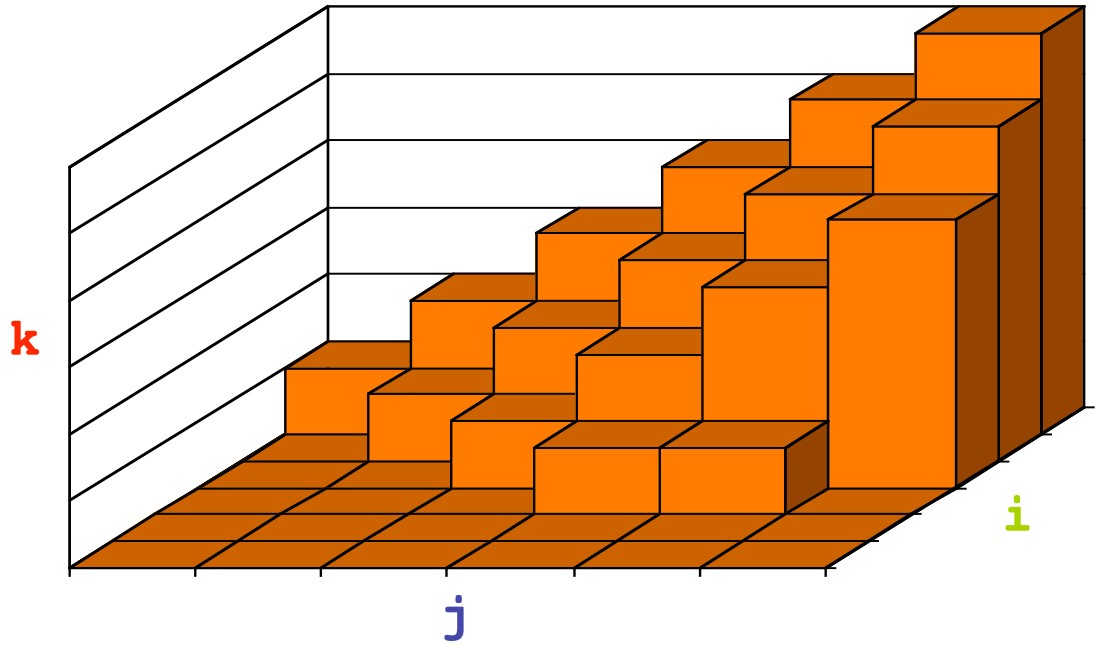
k				↓		
j				↓		
i				↓		
	–2	11	-4	13	–5	2



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;    }}
```

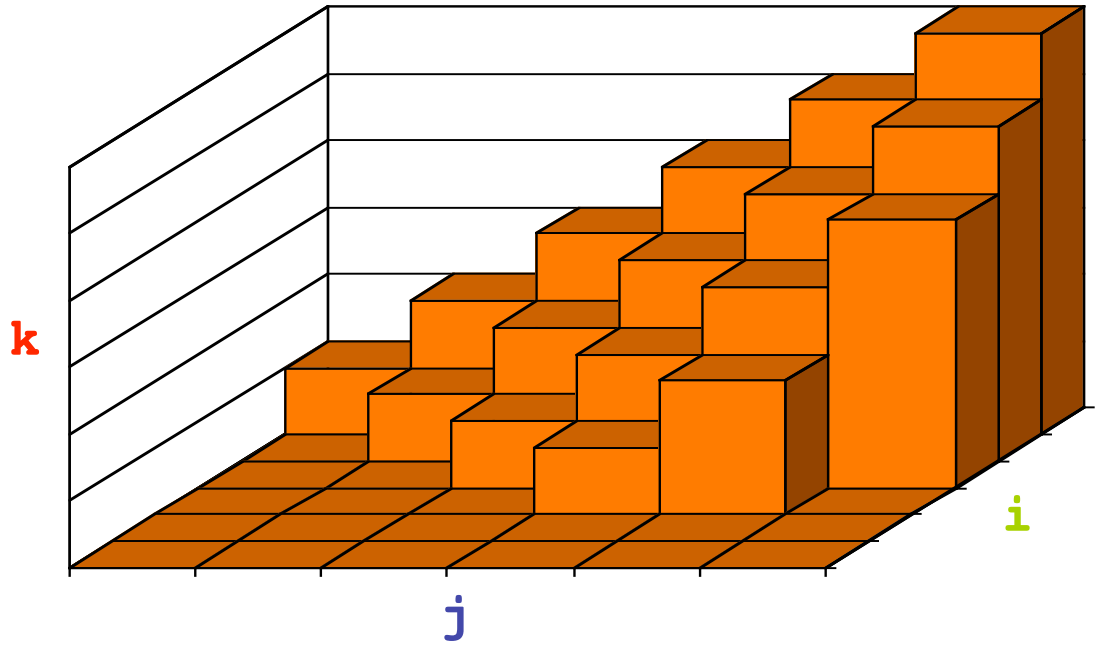
k				↓		
j					↓	
i				↓		
	–2	11	-4	13	–5	2



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;    }}
```

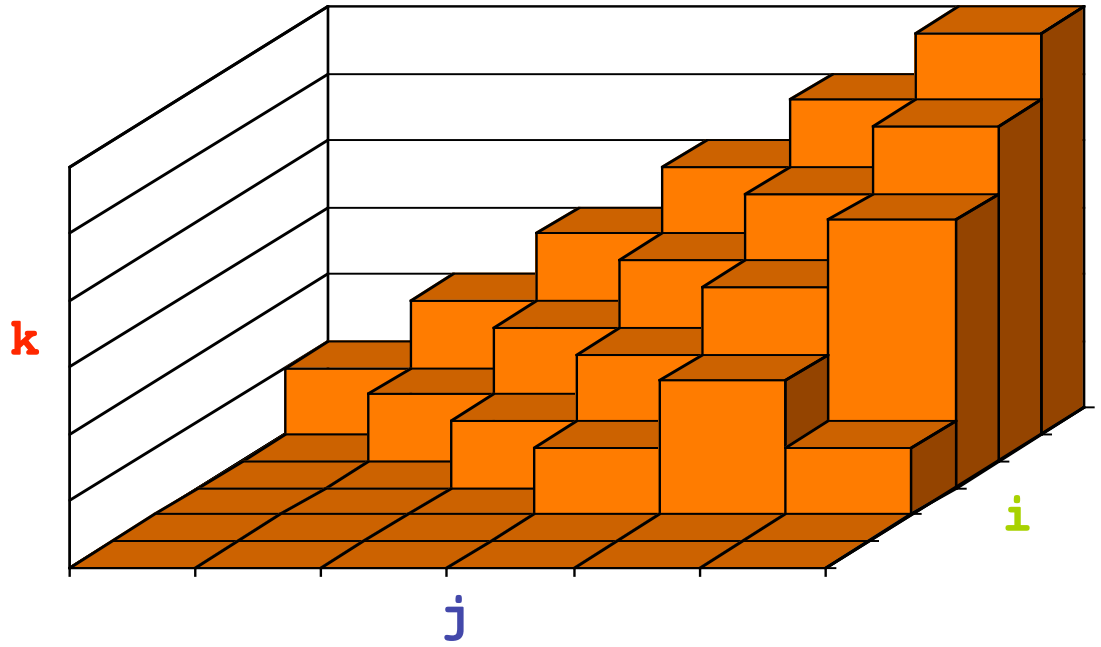
k					↓	
j					↓	
i				↓		
	–2	11	-4	13	–5	2



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;    }}
```

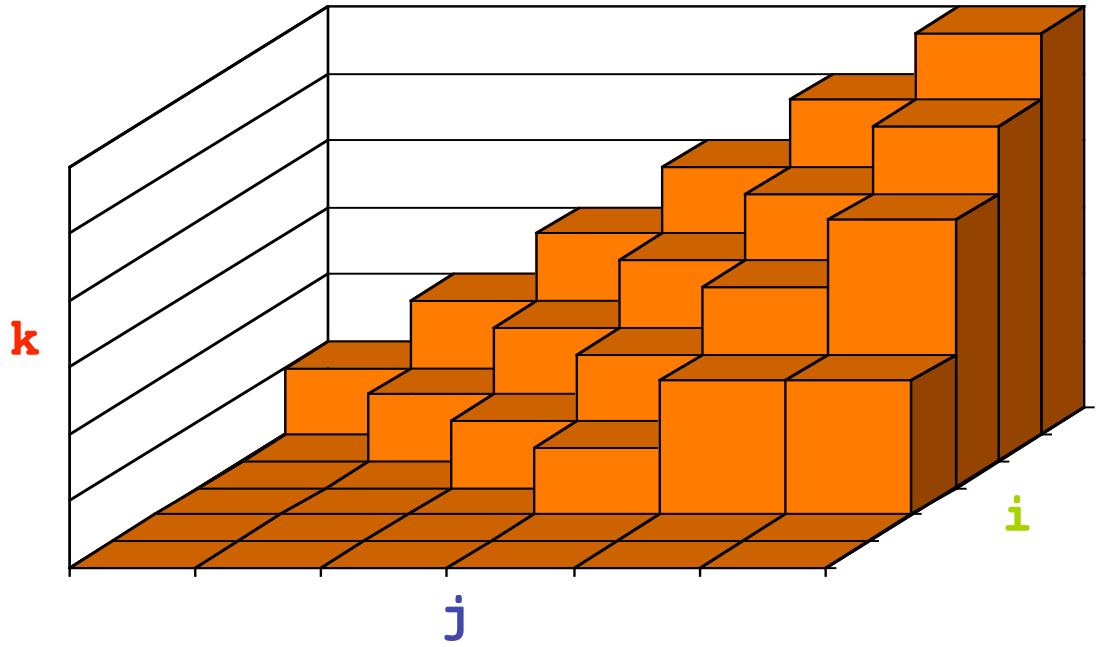
k				↓		
j						↓
i				↓		
	–2	11	-4	13	–5	2



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;    }}
```

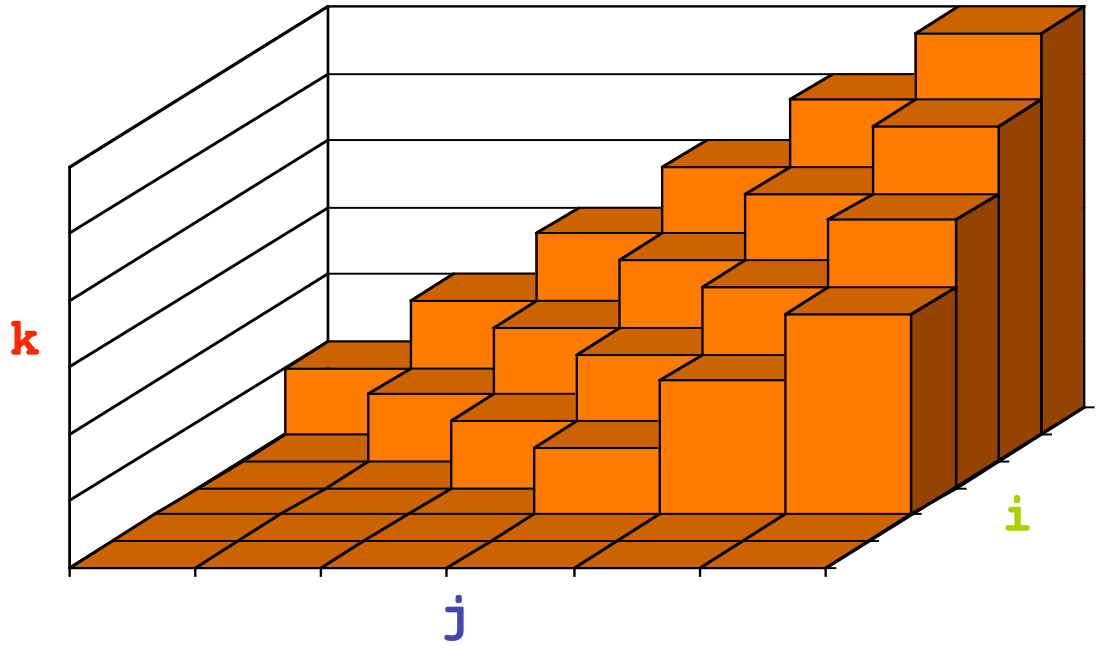
k					↓	
j						↓
i				↓		
	–2	11	-4	13	–5	2



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;    }}
```

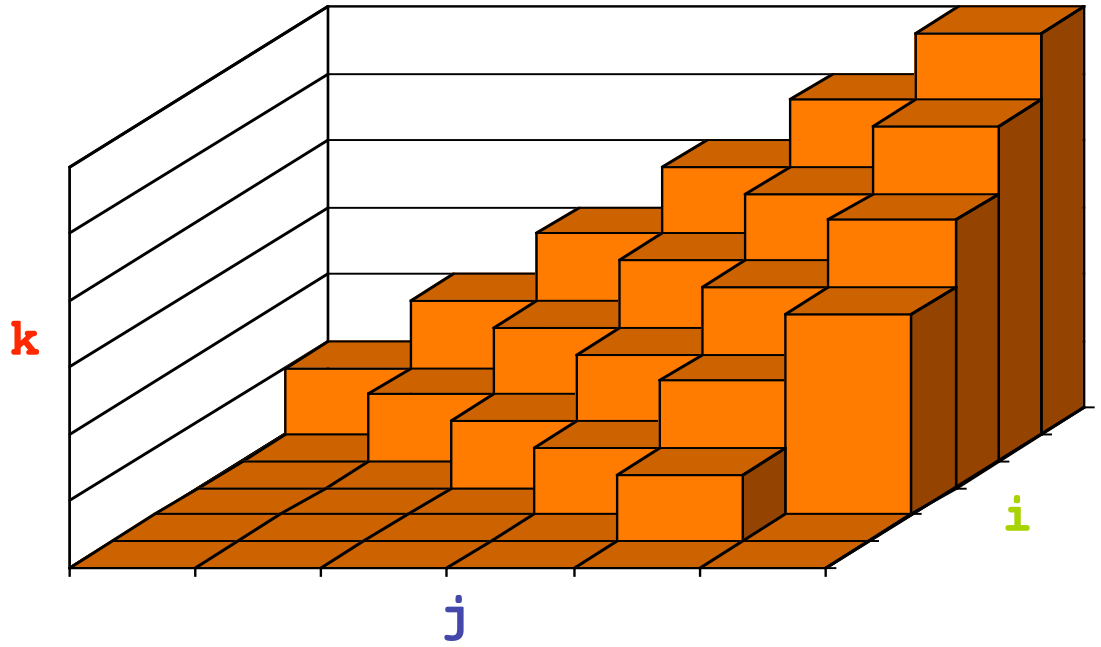
k						↓
j						↓
i				↓		
	–2	11	-4	13	–5	2



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;    }}
```

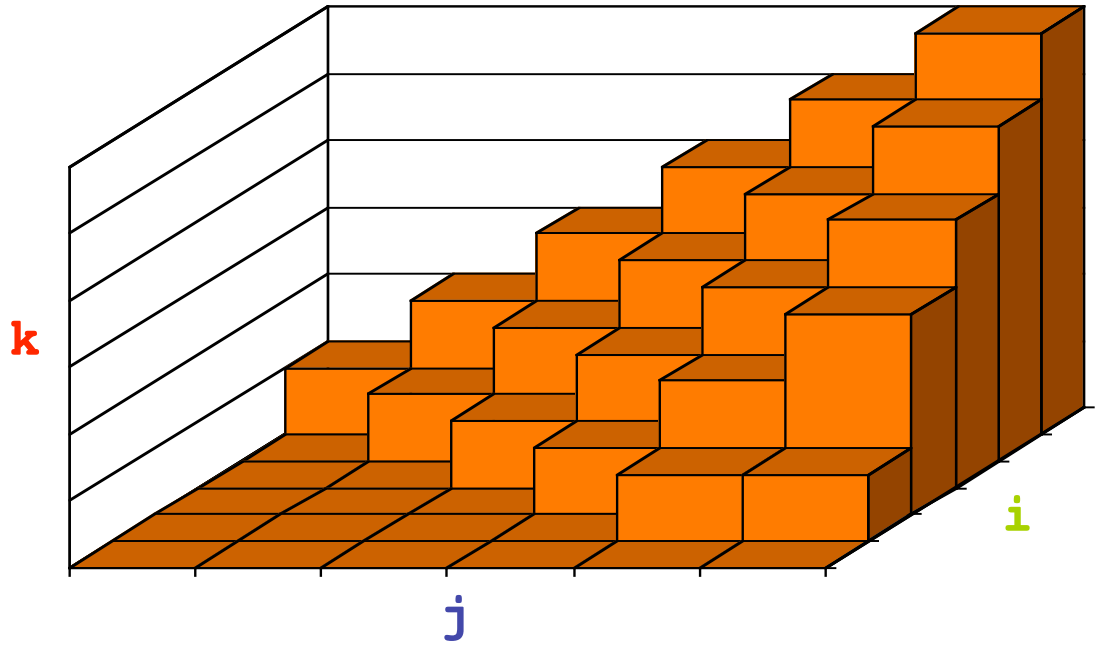
k					↓	
j					↓	
i					↓	
	–2	11	-4	13	–5	2



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;    }}
```

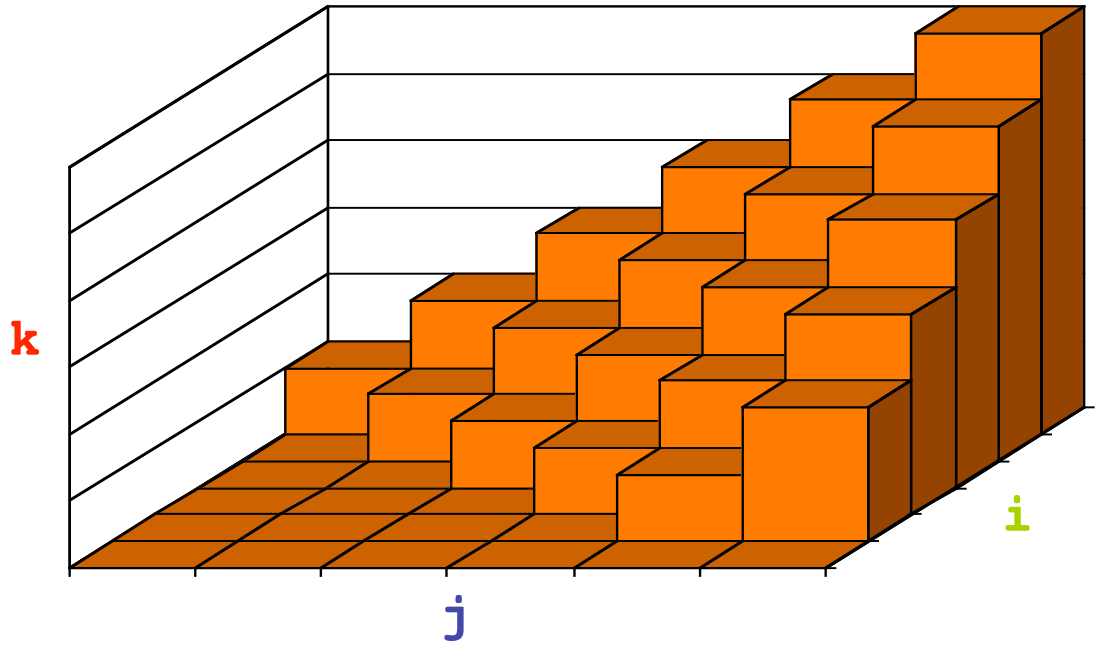
k					↓	
j						↓
i					↓	
	–2	11	-4	13	–5	2



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;    }}
```

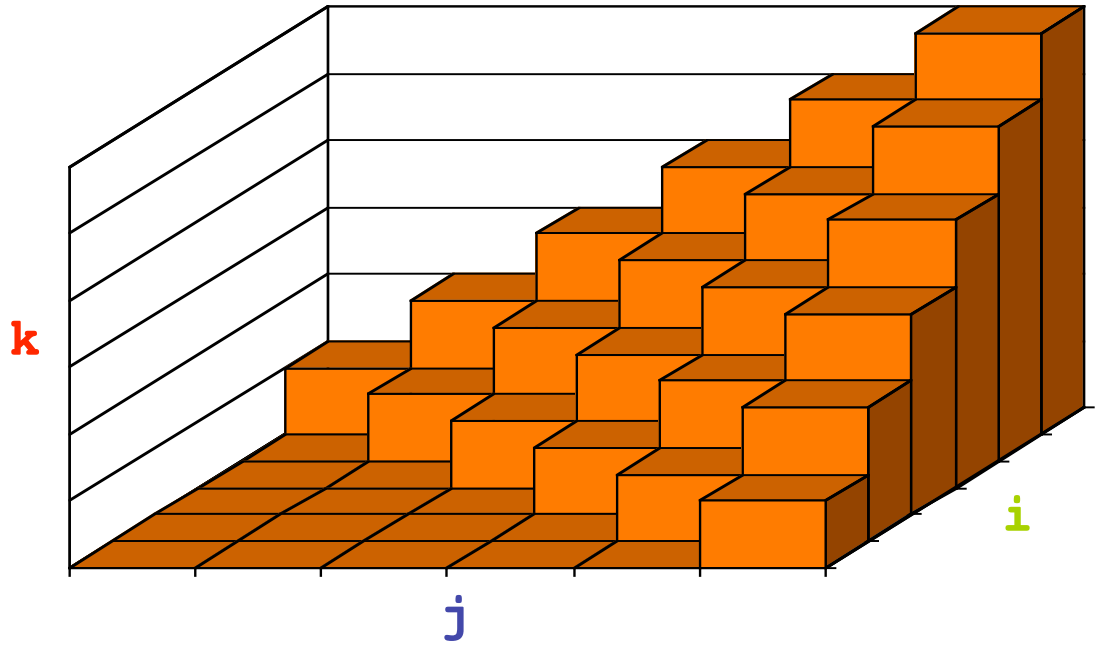
k						↓
j						↓
i					↓	
	–2	11	-4	13	–5	2



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



1 + (1+2) + (1+2+3) + ...

+ (1+2+...+n)

= n * (n+1) * (n+2) / 6

dvs. O(n^3)

Kvadratisk algoritme for max-sum-delsekvens

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

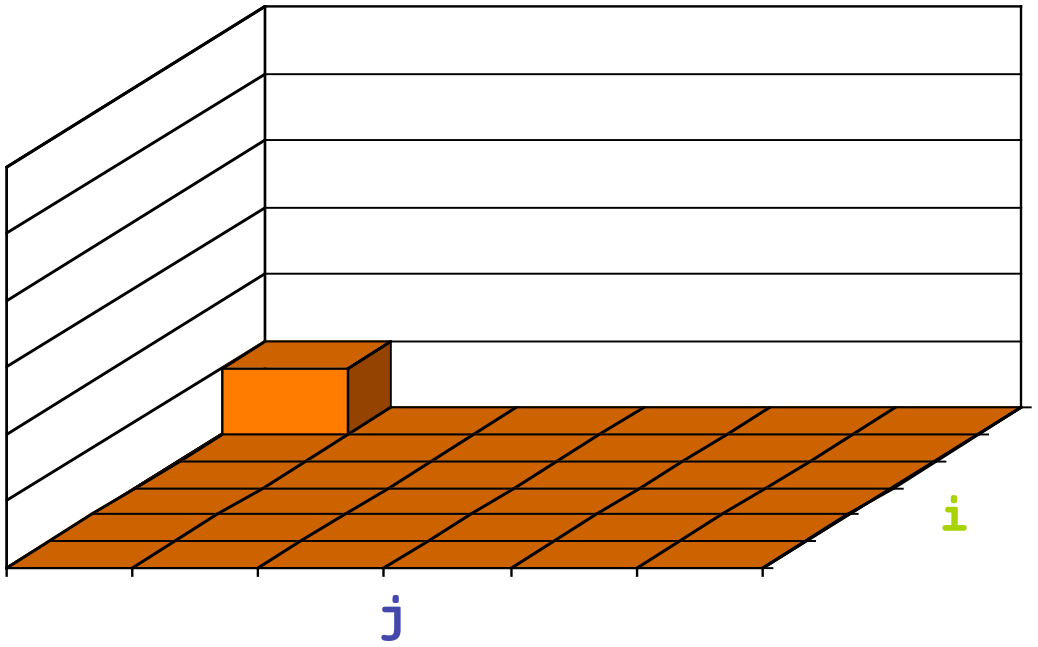
Dvs. $Sum_{i,j+1} = Sum_{i,j} + a[j+1]$

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

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;   }}
```

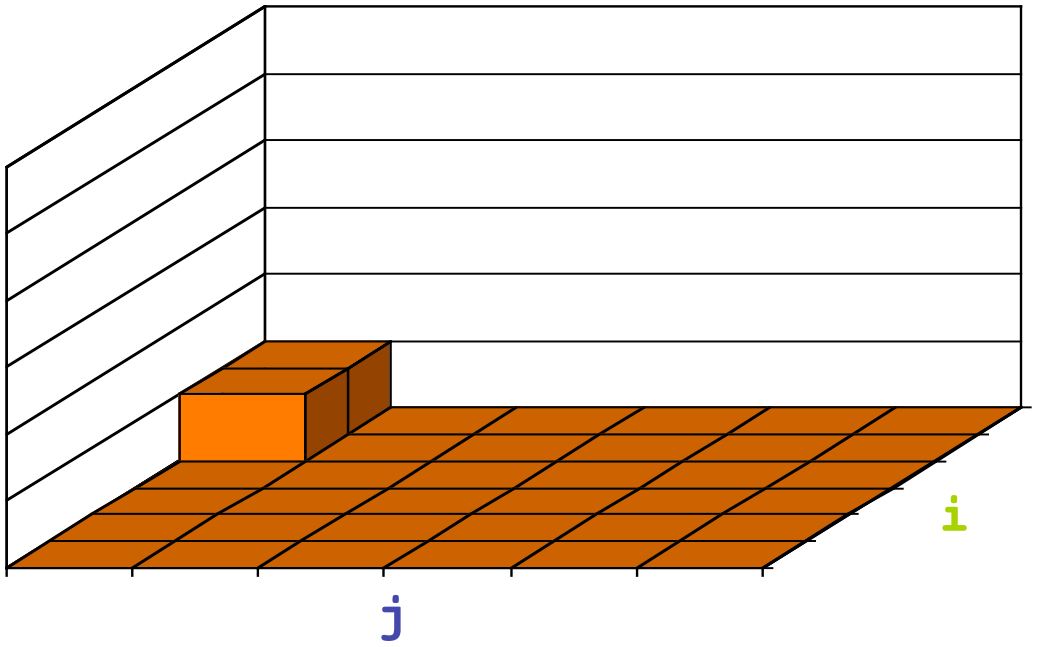
j	↓					
i	↓					
	–2	11	-4	13	–5	2



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;   }}
```

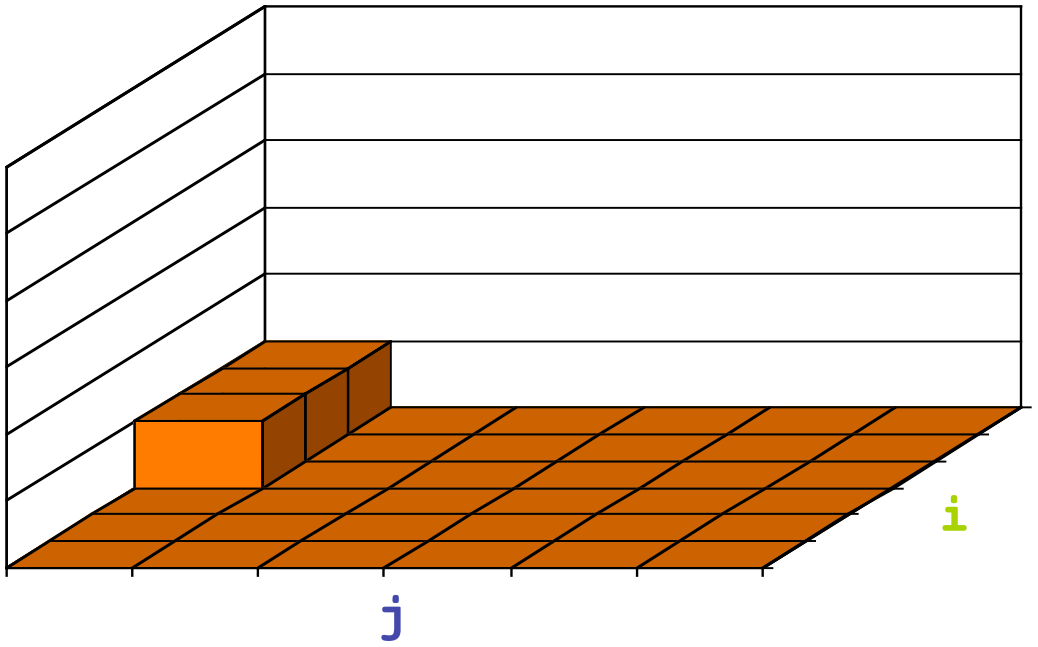
j		↓				
i	↓					
	–2	11	-4	13	–5	2



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;   }}
```

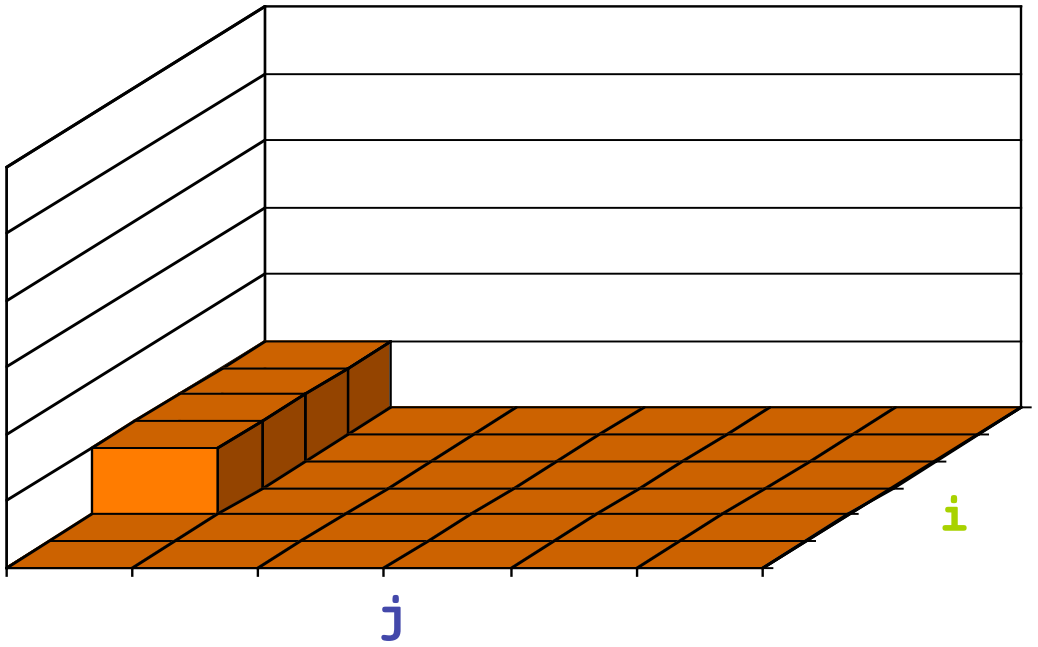
j			↓			
i	↓					
	–2	11	-4	13	–5	2



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;   }}
```

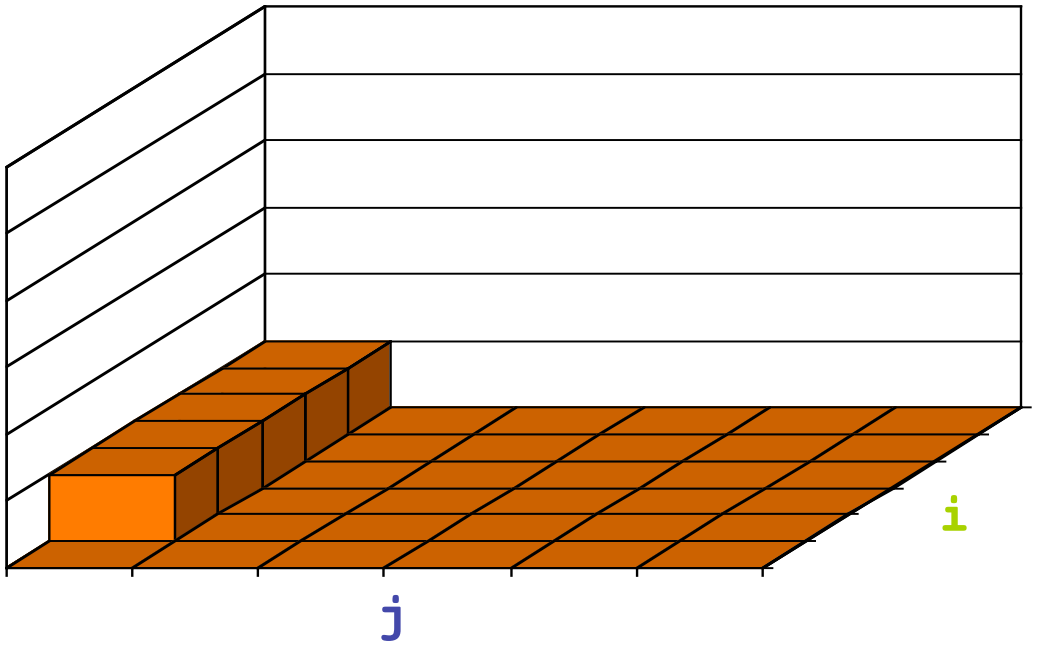
j				↓		
i	↓					
	–2	11	-4	13	–5	2



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;   }}
```

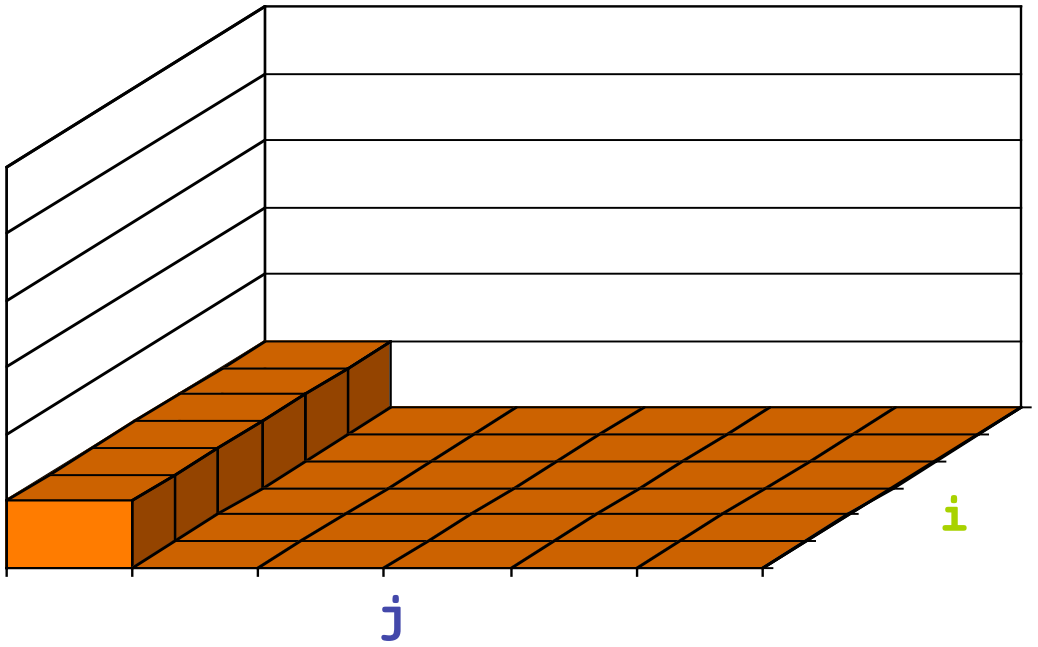
j					↓	
i	↓					
	–2	11	-4	13	–5	2



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;   }}
```

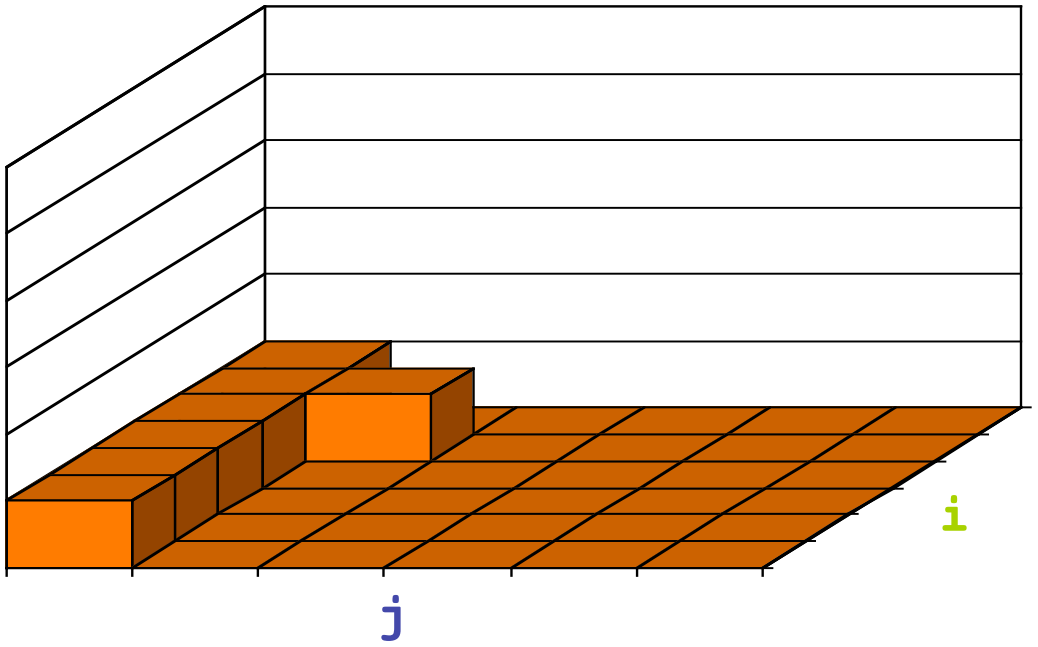
j						↓
i	↓					
	–2	11	-4	13	–5	2



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;   }}
```

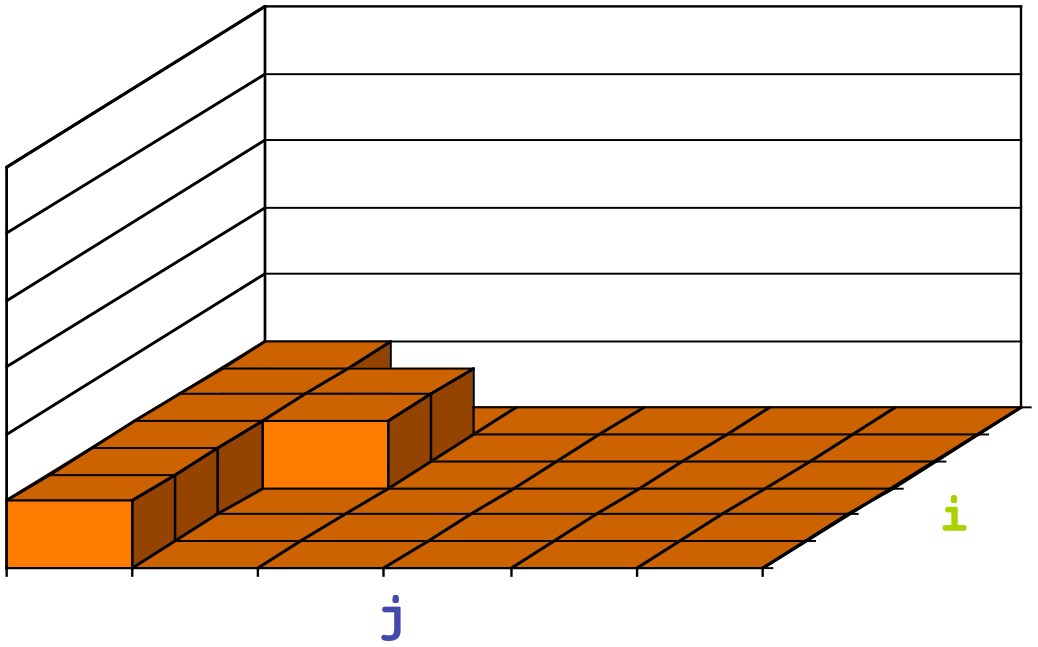
j		↓				
i		↓				
	–2	11	-4	13	–5	2



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;   }}
```

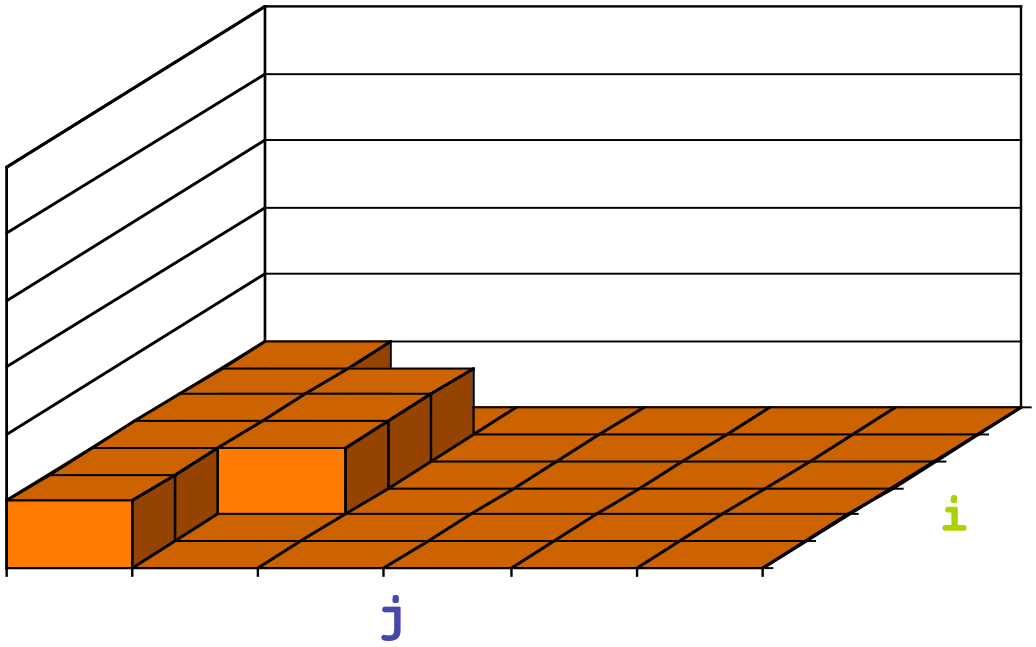
j			↓			
i		↓				
	–2	11	-4	13	–5	2



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;   }}
```

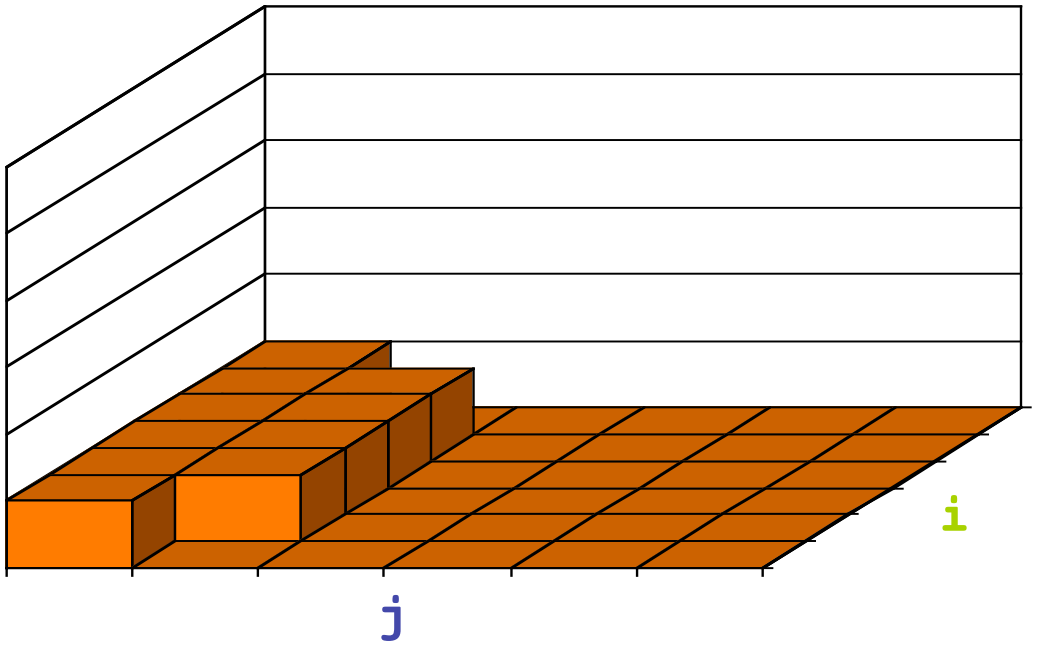
j				↓		
i		↓				
	–2	11	-4	13	–5	2



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;   }}
```

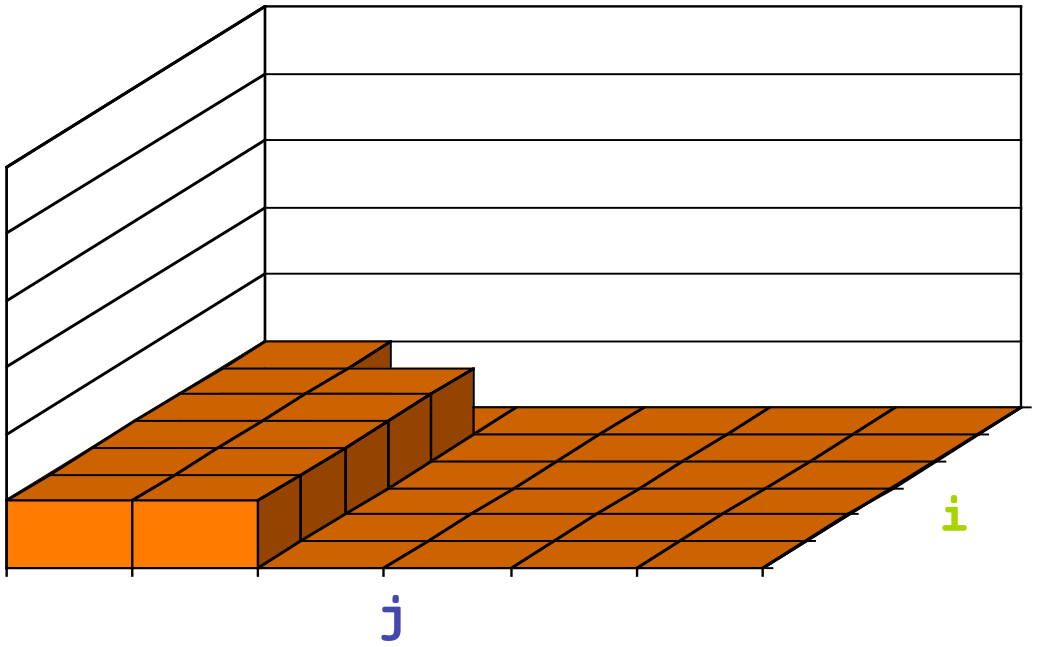
j					↓	
i		↓				
	–2	11	-4	13	–5	2



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;   }}
```

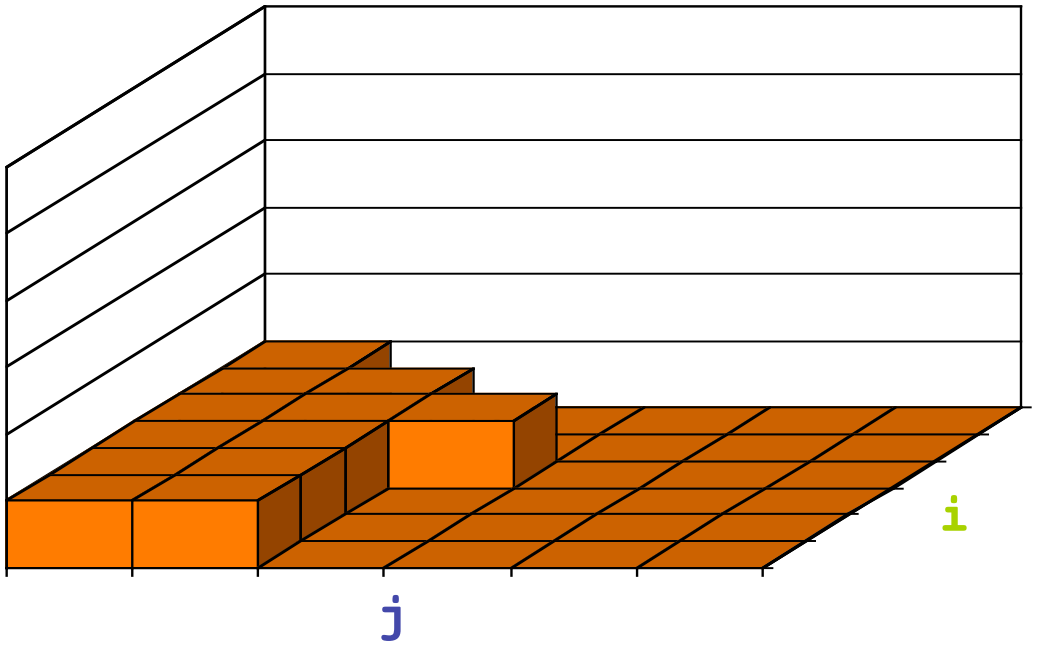
j						↓
i		↓				
	–2	11	-4	13	–5	2



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;   }}
```

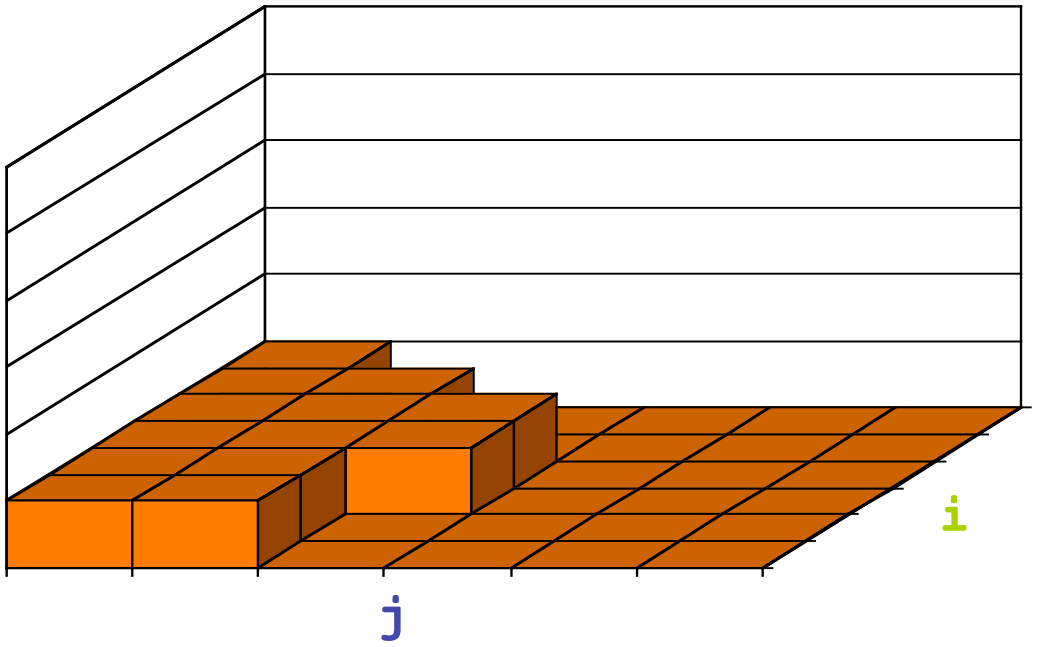
j			↓			
i			↓			
	–2	11	-4	13	–5	2



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;   }}
```

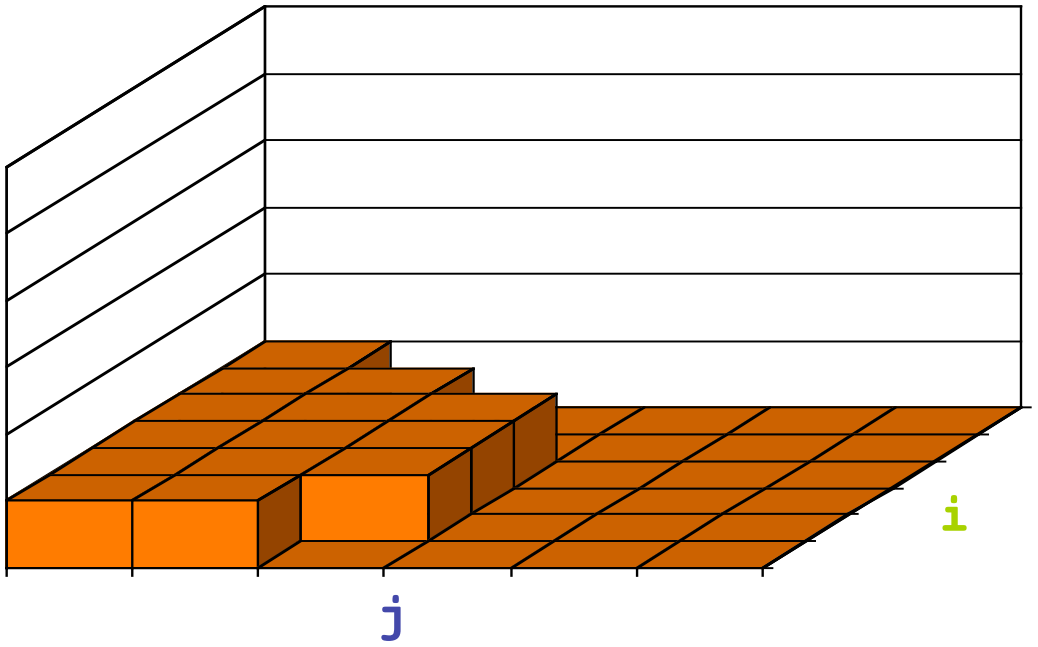
j				↓		
i			↓			
	–2	11	-4	13	–5	2



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;   }}
```

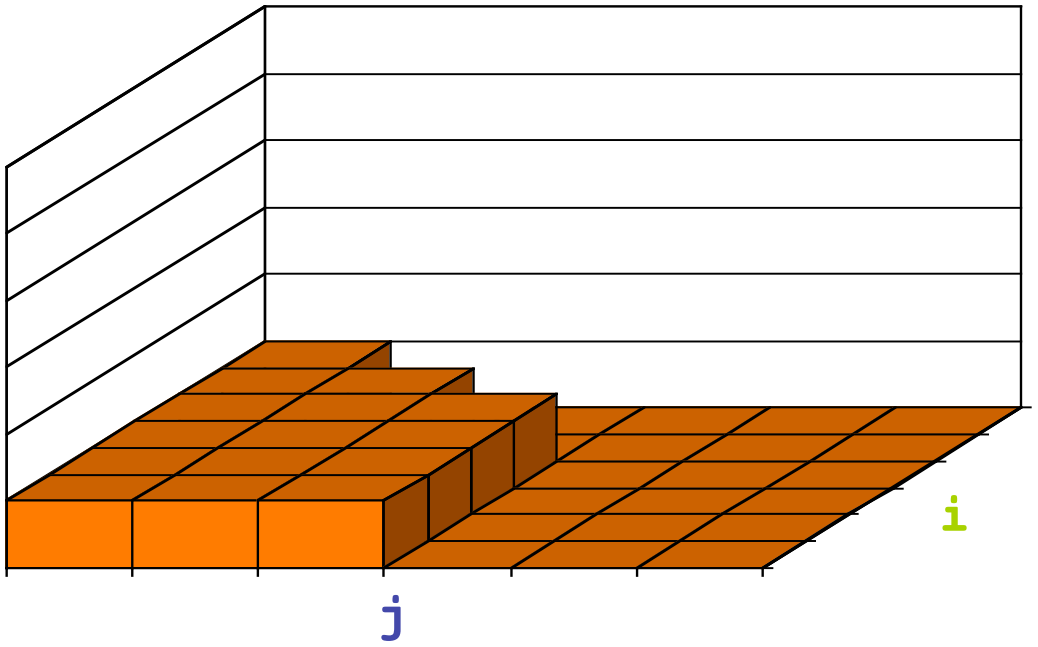
j					↓	
i			↓			
	–2	11	-4	13	–5	2



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;   }}
```

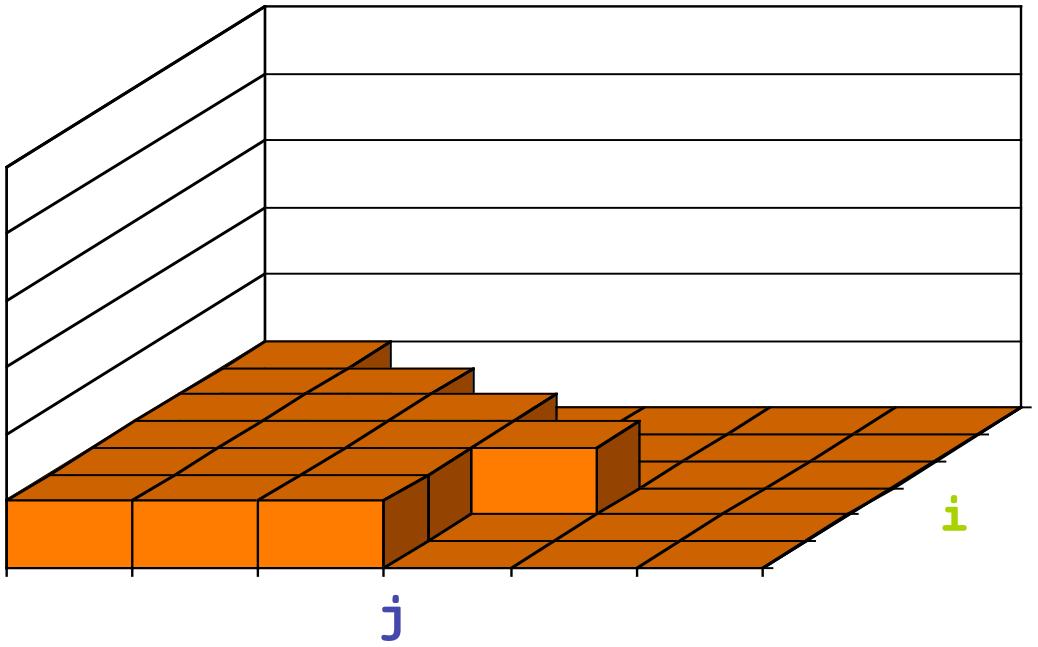
j						↓
i			↓			
	–2	11	-4	13	–5	2



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;   }}
```

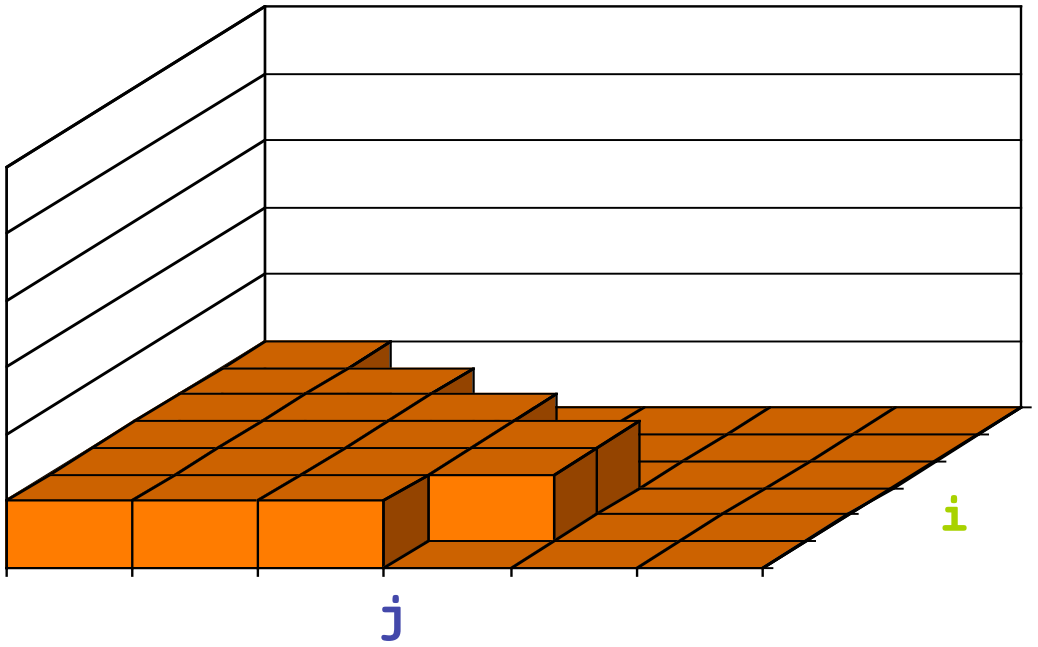
j				↓		
i				↓		
	–2	11	-4	13	–5	2



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;   }}
```

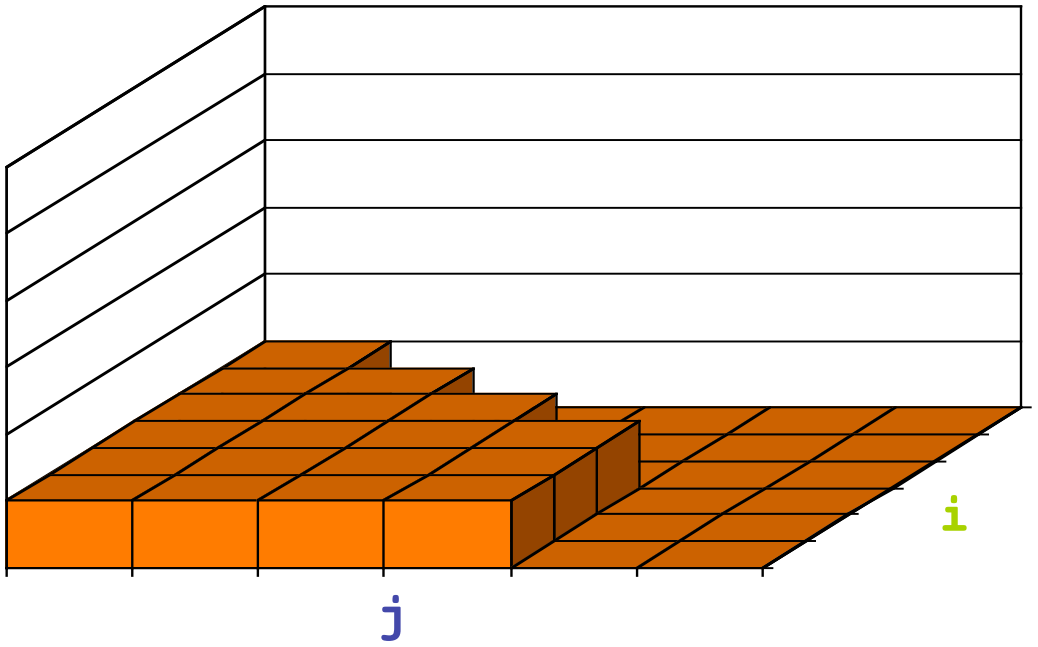
j					↓	
i				↓		
	–2	11	-4	13	–5	2



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;   }}
```

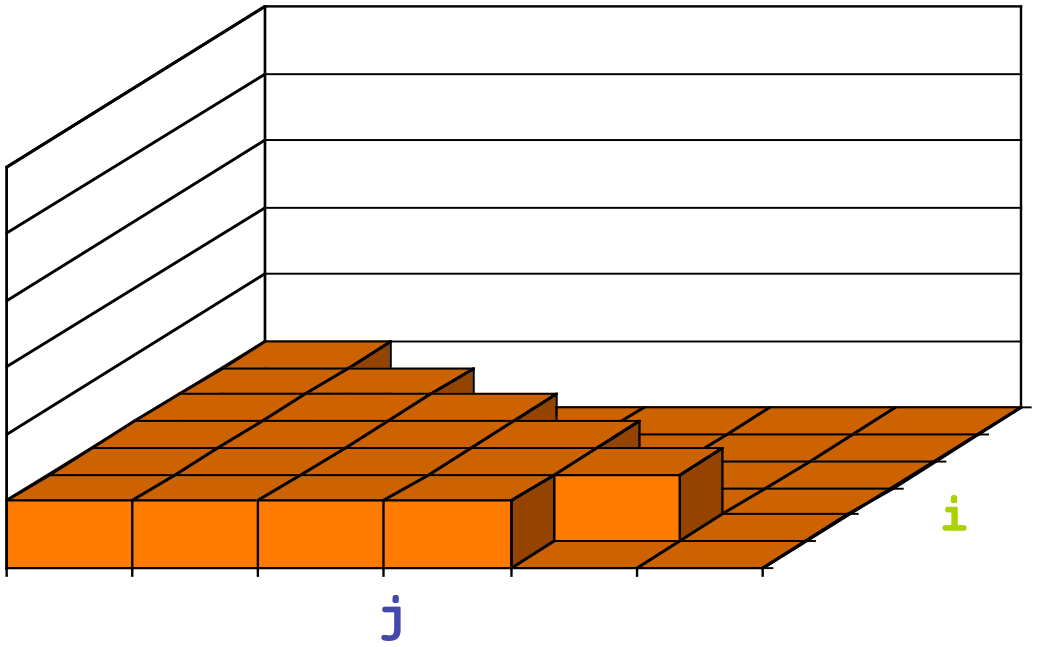
j						↓
i				↓		
	–2	11	-4	13	–5	2



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;   }}
```

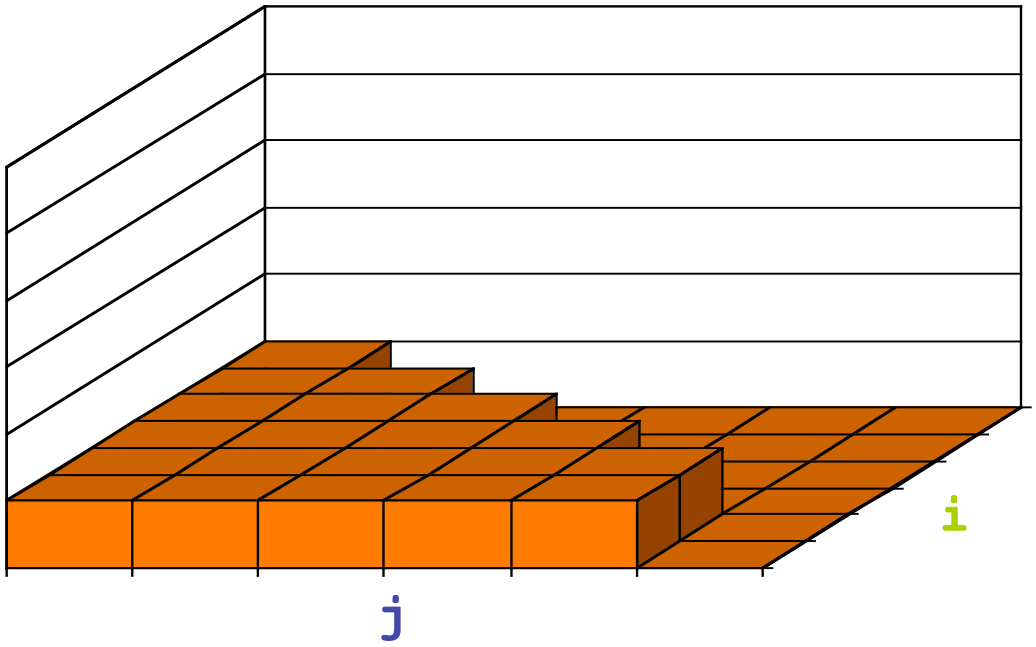
j					↓	
i					↓	
	–2	11	-4	13	–5	2



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

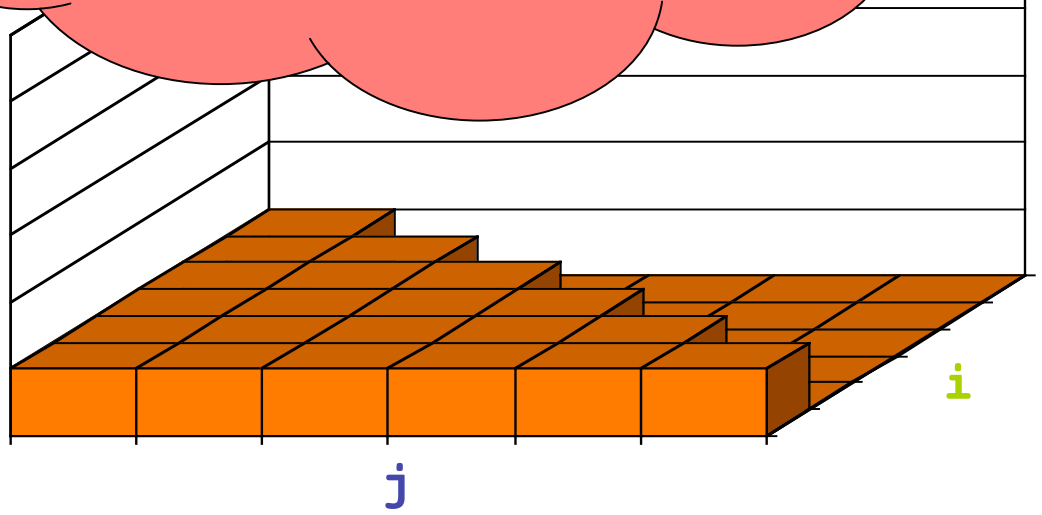


Kvadratisk algoritme for max-sum-delsekvens

```
maxSum = 0;
for(i=1; i<=n; i++) {
  sum = 0;
  for(j=i; j<=n; j++) {
    sum
```

j						↓
i						↓
	2	11	-4	13	–5	2

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



$1 + 2 + ... + n$

$= n * (n-1) / 2$

altså $O(n^2)$

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 <= 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"

$\approx \log_2 n$

Lidt om logaritmefunktionen

Defineret som omvendt til eksponentialfunktionen:

hvis $2^x = m$ så $\log_2 m = x$

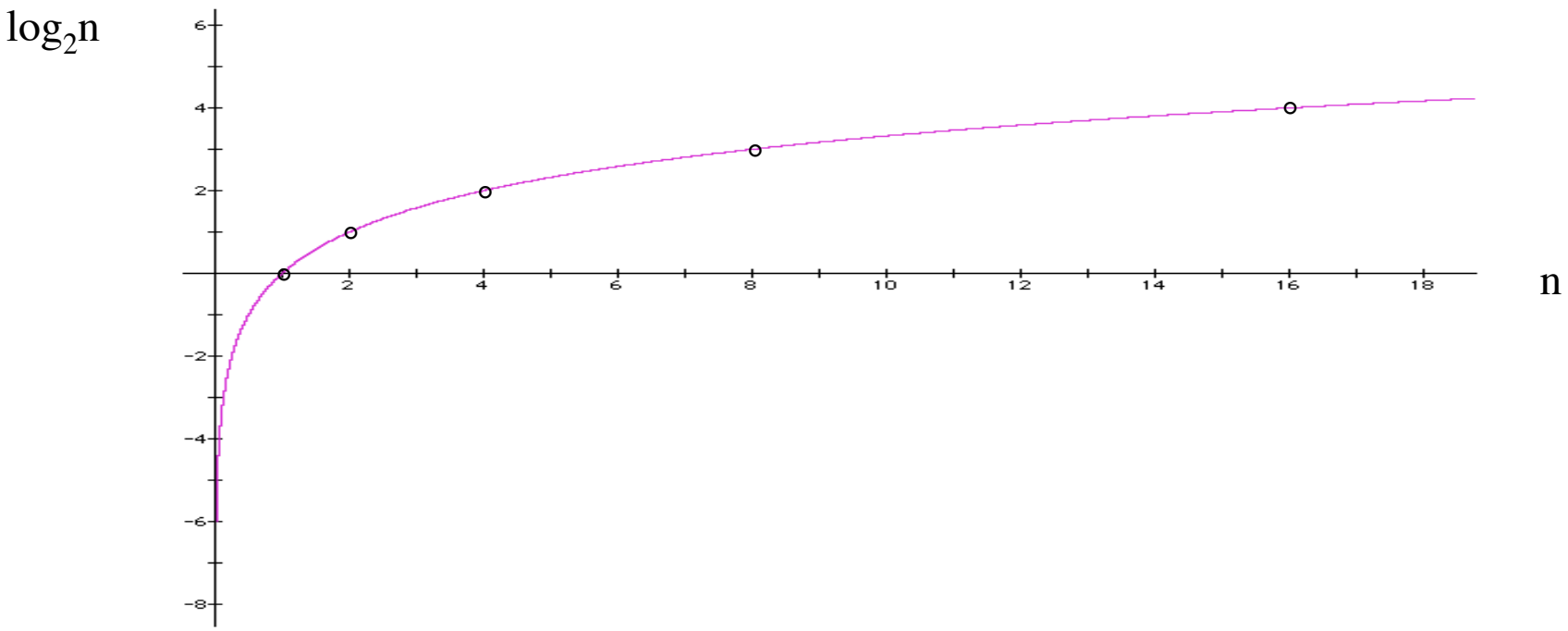
Eksempler:

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 ≈ antal tegn i n's binære repræsentation
- $\log_2$ n ≈ antal gange n skal halveres for at blive 1

Funktionskurven for $\log_2 n$



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

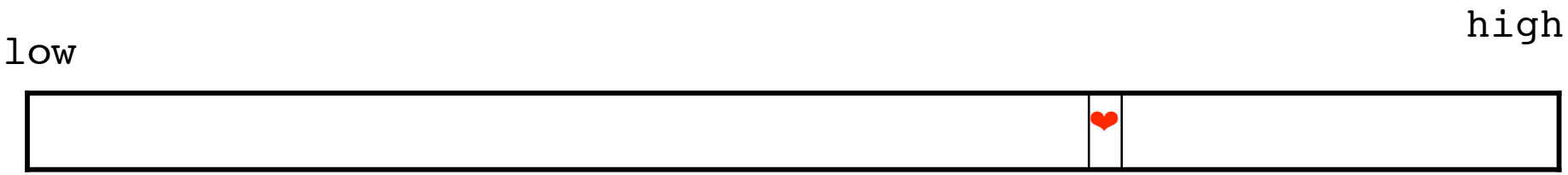
Kaldes: `binarySearch(0,a.length-1,a,x)`

```
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
```

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

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

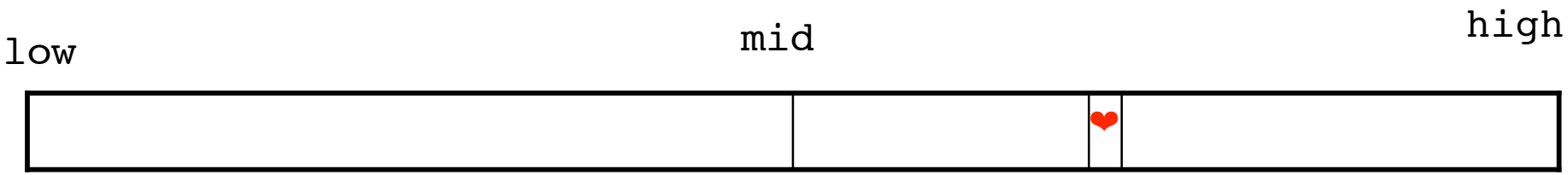
```
int binarySearch(int low, int high, int [] a, int x){
 if(low>high) return -1;
 int mid = (low+high)/2;
 if( a[mid] < x )
    binarySearch(mid+1,high,a,x)
 else if( a[mid] > x )
    binarySearch(low,mid-1,a,x);
else return mid; }} //a[mid]==x
```



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

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

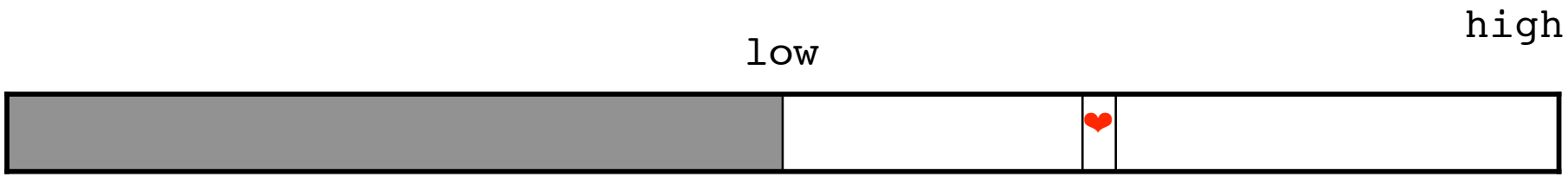
```
int binarySearch(int low, int high, int [] a, int x){
 if(low>high) return -1;
 int mid = (low+high)/2;
 if( a[mid] < x )
    binarySearch(mid+1,high,a,x)
 else if( a[mid] > x )
    binarySearch(low,mid-1,a,x);
else return mid; }} //a[mid]==x
```



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

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

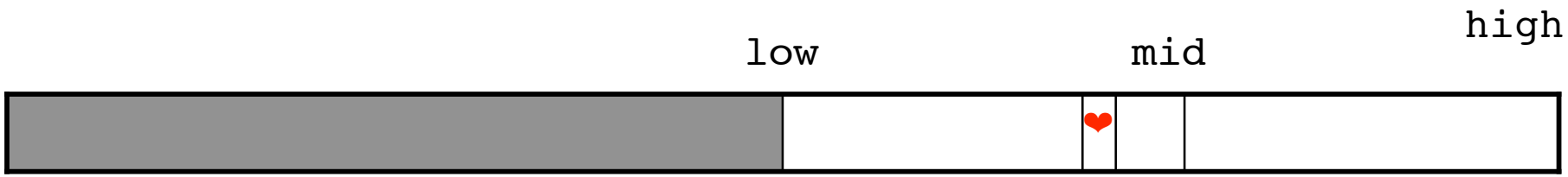
```
int binarySearch(int low, int high, int [] a, int x){
 if(low>high) return -1;
 int mid = (low+high)/2;
 if( a[mid] < x )
    binarySearch(mid+1,high,a,x)
 else if( a[mid] > x )
    binarySearch(low,mid-1,a,x);
else return mid; }} //a[mid]==x
```



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

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

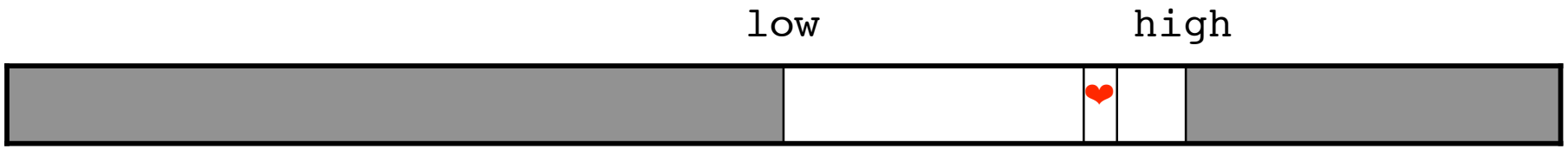
```
int binarySearch(int low, int high, int [] a, int x){
 if(low>high) return -1;
 int mid = (low+high)/2;
 if( a[mid] < x )
    binarySearch(mid+1,high,a,x)
 else if( a[mid] > x )
    binarySearch(low,mid-1,a,x);
else return mid; }} //a[mid]==x
```



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

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

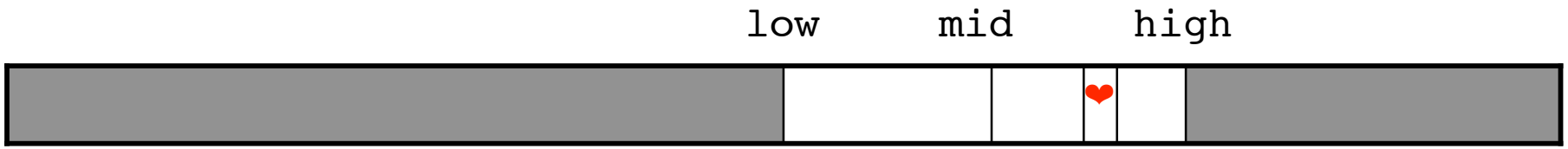
```
int binarySearch(int low, int high, int [] a, int x){
 if(low>high) return -1;
 int mid = (low+high)/2;
 if( a[mid] < x )
    binarySearch(mid+1,high,a,x)
 else if( a[mid] > x )
    binarySearch(low,mid-1,a,x);
else return mid; }} //a[mid]==x
```



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
```



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
```

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
```

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
```

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
```

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
```

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
```

Binær søgning i Java II: Rekursiv, generisk

```
package java.lang;
public interface Comparable {int 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)
    binarySearch1(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);
```

Binær søgning i Java III: Iterativ, generisk

Lærebogens version:

```
int binarySearch(Comparable[]a
 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
      high = mid-1;
    else return mid; }
  return -1;  }
```

Iterativ vs. rekursiv:
En smagssag, men rekursive alg. ofte simplere 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!

Afsluttende om O-notation

- udvide med flere parametre, f.eks. $O(m*2^n)$
- 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 karakteristikker:
  - "T(n) er Ω(F(n))" angiver undergrænse
  - "T(n) er Θ(F(n))" angiver eksakt karakteristik
  - "T(n) er o(F(n))" angiver karakteristik som er klart for høj

Opgaver til øvelserne

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

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 12. oktober