

Oppgave 1

Under følger en liste med kjøretidsklasser – ordne dem i stigende rekkefølge:

konstant, eksponensiell, lineær, kubisk, kvadratisk, faktoriell, logaritmisk, n-log-n

Svar:

konstant, logaritmisk, lineær, n-log-n, kvadratisk, kubisk, eksponensiell, faktoriell.



Oppgave 2

Hva gjør følgende program, og hva blir kjøretiden?

```
Algorithm Azathoth( $A[1..n]$ )  
 $i \leftarrow 1$   
 $j \leftarrow n$   
while  $i < j$   
    if  $A[i] > A[j]$   
         $i \leftarrow i + 1$   
    else  
         $j \leftarrow j - 1$   
return  $A[i]$ 
```

Svar:

Finner det minste elementet i A på $\Theta(n)$ tid.



Sortering i lineær tid
 $\Theta(n)$

Å javel ja, når ble det mulig?

Hemmeligheten

- ▶ Sortering i det generelle tilfellet er $\Omega(n \log n)$ i worst case (bevisbart)
 - ▶ Ekvivalent: Kan ikke garantere at det alltid går fortere enn $n \log n$
 - ▶ Vi er som regel interessert i at det *alltid* går fortere enn x tid
- ▶ Men; hva hvis vi *vet* noe om inputen? Kan vi *anta* noe om den?
- ▶ Eksempel:
 - ▶ Den består kun av tall mellom 0 og 100?
 - ▶ Den består kun av tall med mindre enn 20 siffer?
 - ▶ Input er jevnt fordelt over et intervall?
- ▶ Ektremtilfelle:
Sortering av tall: Hva hvis input består kun av 0 og 1?



Counting Sort

- ▶ Antar at inputelementene er heltall mellom 0 og k , for et heltall k . Når $k = O(n)$, er kjøretiden $\Theta(n)$
- ▶ **Grunnidé:** Vi teller antall forekomster av hvert tall, og løper så gjennom alle tallene og skriver ut det antallet vi fant ut.
 - ▶ Altså: Vi teller opp antall 1-tall, 2-tall, 3-tall osv, og skriver til slutt ut så mange 1, 2 og 3-tall.
- ▶ Vi får problemer hvis vi har mer enn bare sorteringsnøkkelen som skal med ("satelittdata")
- ▶ **Bedre idé:** Vi finner ut for hvert element x , hvor mange elementer som er mindre enn x .
 - ▶ Vi vet da hvor x skal i output'en!



Counting Sort (forts)

Counting-Sort(A, B, k)

```
for i ← 0 to k  
do C[i] ← 0
```

Initialiser array,
sett alle antall til 0

```
for j ← 1 to length(A)  
do C[A[j]] ← C[A[j]] + 1
```

Tell opp antall av hver verdi,
C[i] inneholder antall i'ere

```
for i ← 1 to k  
do C[i] ← C[i] + C[i - 1]
```

Summer slik at C[i] inneholder
antall elementer før og inkl i.

```
for j ← length(A) downto 1  
do B[C[A[j]]] ← A[j]  
C[A[j]] ← C[A[j]] - 1
```

Sett elementene inn på riktig
plass i output. Dekrementer
for neste like verdi



Stabilitet

- ▶ Sorteringsalgoritmer kan være stabile, eller ikke stabile
- ▶ Forteller om *like* elementer blir værende i samme rekkefølge etter sorteringen som de var før
- ▶ Stabil:
 $3, 1, 2a, 2b \rightarrow 1, 2a, 2b, 3$
- ▶ Ikke stabil:
 $3, 1, 2a, 2b \rightarrow 1, 2b, 2a, 3$
- ▶ Viktig egenskap når vi nå skal se på Radix Sort
- ▶ Stabile: Counting, bubble, insertion, merge, radix, bucket
Ikke stabile: Heapsort, quicksort



Radix Sort

- ▶ Antar at tallene i input har d siffer
- ▶ Sorterer tall ved å bruke en annen sorteringsalgoritme til å sortere på et og et siffer av gangen
 - ▶ Counting sort er ypperlig, siden vi vet hvilke tall hvert siffer kan innta
- ▶ Starter med minst signifikante (til høyre) og fortsetter til mest signifikante siffer
- ▶ Må basere seg på en stabil sorteringsalgoritme!
- ▶ Kjøretid: $\Theta(d(n + k))$

Radix-Sort(A, d)

for $i \leftarrow$ **to** d

do use a stable sort to sort array A on digit i



Bucket Sort

- ▶ Antar at input er jevnt fordelt over et intervall
- ▶ Idé:
 - ▶ Vi deler opp intervallet i n like store bølter
 - ▶ Elementene som skal sorteres puttes i sine respektive bølter
 - ▶ Hver bølte sorteres med insertion sort
 - ▶ Bøttene konkatineres til det ferdige resultatet
- ▶ Kjører på $\Theta(n)$ dersom vi har uniform fordeling, men beviset er ikke spesielt pent.



Bucket sort (forts.)

Bucket-Sort (A)

$n \leftarrow \text{length}(A)$

```
for i ← 1 to n  
  do insert A[i] into list B[msb(A[i], k)]
```

```
for i ← 1 to n  
  do sort list B[i] with insertion sort
```

```
concatenate the lists B[0], B[1]...B[n-1] in  
order
```



Eksamensoppgave (oppg. 3 - H05, 15%)

Vi har gitt 3 vektorer $A[1\dots n]$, $B[1\dots n]$, $C[1\dots n]$, alle med verdier som er fødselsdatoer på formen "ddmmåå", f.eks.: $A[1342] = 180368$.

Vi har som mål å finne ut hvilke datoer som er med i både A , B og C .

Du skal lage en enkel og særdeles effektiv algoritme for å framskaffe en vektor D som inneholder disse felles fødselsdatoene. Tenk på konstantleddene i metoden også.

Beskriv en slik algoritme SNITTABC. Uttrykk algoritmens tidskompleksitet ved Θ -notasjonen.



Linker til demoene

- ▶ Mye forskjellig:

<http://www.cs.usfca.edu/~galles/visualization/>

- ▶ Countingsort

<http://users.cs.cf.ac.uk/C.L.Mumford/tristan/CountingSort.html>

- ▶ RadixSort

<http://www.cs.umass.edu/~immerman/cs311/applets/visual/RadixSort.html>

