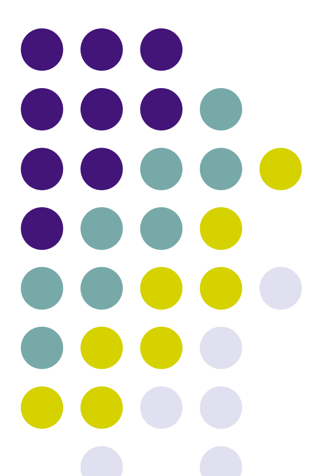
Øvingsforelesning 6

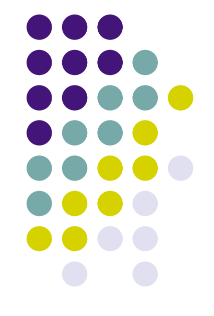
Prioritetskøer. Sortering. Median.

Martin Gammelsæter
martigam@stud.ntnu.no

Foiler i all hovedsak laget av
Kristian Veøy

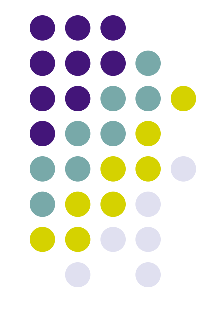


I dag



- Neste ukes øving
- Sortering
- Prioritetskø
- Median
- Denne ukens øving

Neste ukes øving



- Veibygging i Ogligogo 2
  - Tar inn en liste over kanter og deres vekting.
  - Finn MST (Kruskal eller Prim) og finn den dyreste kanten.

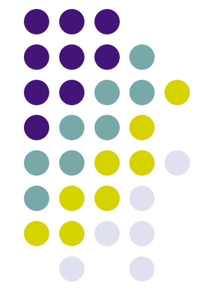
Hva er sortering?

- Vi må finne en permutasjon av elementene slik at:
- X1 <= X2 <= X3 <= ... <= Xn-1 <= Xn

6	2	4	1	1	3	7	2

1	1	2	2	3	4	6	7

Sorteringsalgoritmer:



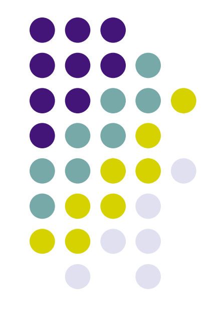
Sammenligning:

- $O(n^2)$
  - Bubble sort
  - Insertion sort
  - Selection sort
- $O(n \lg n)$
  - Heapsort
  - Merge sort
  - Quicksort

Uten sammenligning:

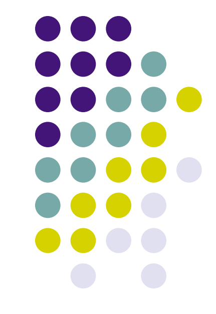
- $O(n)$
  - Counting sort
  - Radix sort
  - Bucket sort

Bubblesort



- ”Sjekker” definisjonen av sortert:
- X1 <= X2 <= X3 <= ... <= Xn-1 <= Xn
- Hvis et usortert par oppdages: Xi > Xi+1
  - Bytt om på de to verdiene.
- Gjenta helt til ingen usorterte par oppdages.

Eksempel:



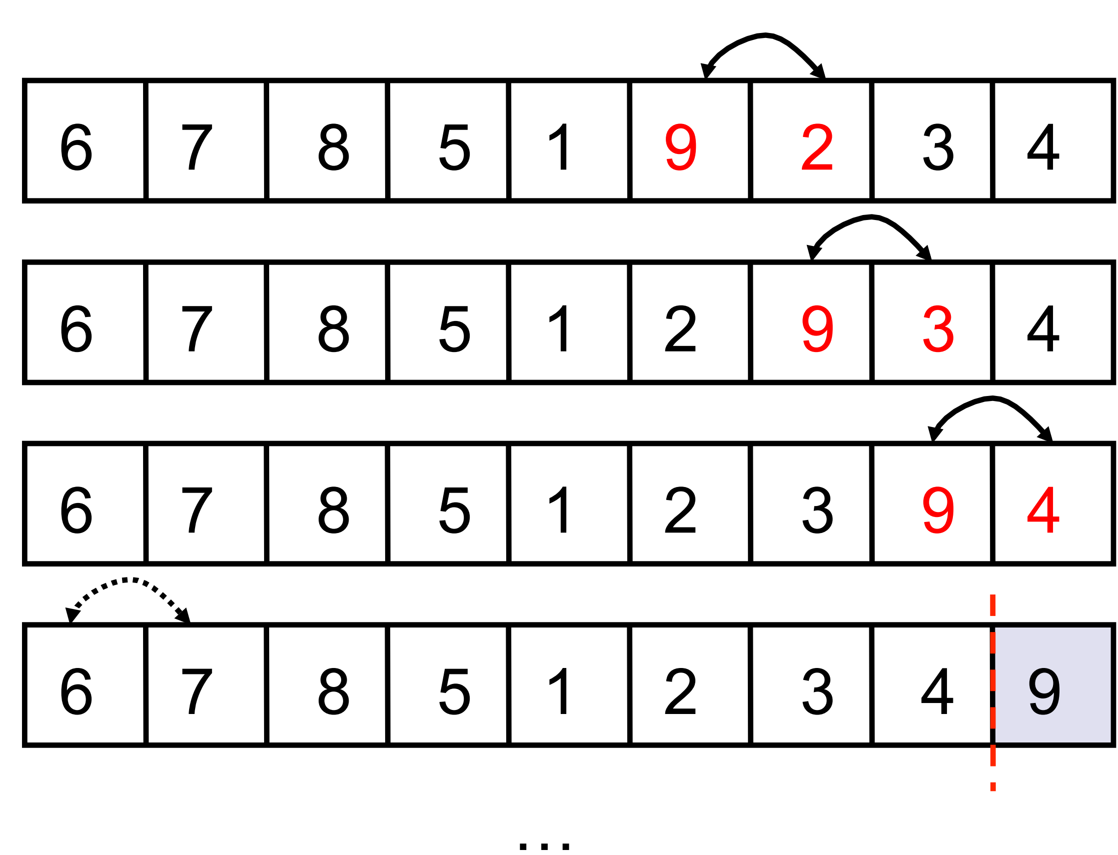
6	7	8	9	5	1	2	3	4

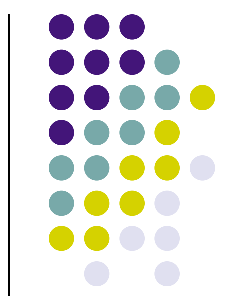
6	7	8	9	5	1	2	3	4

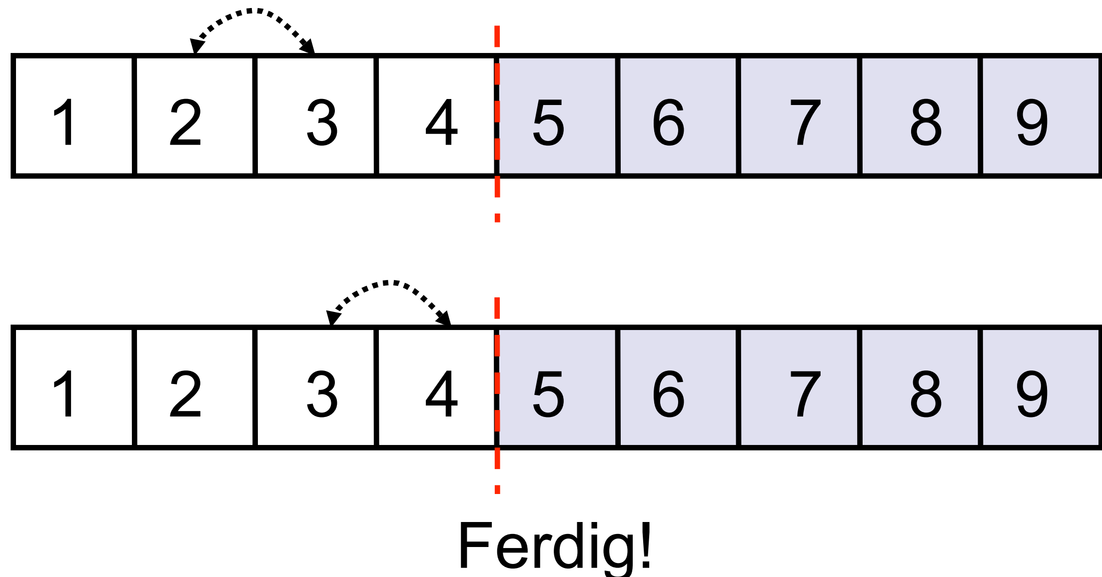
6	7	8	9	5	1	2	3	4

6	7	8	9	5	1	2	3	4

6	7	8	5	9	1	2	3	4

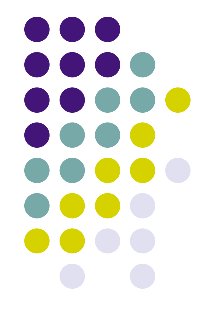

6 7 8 5 1 9 2 3 4
6 7 8 5 1 2 9 3 4
6 7 8 5 1 2 3 9 4
6 7 8 5 1 2 3 4 9
…
1 2 3 4 5 6 7 8 9





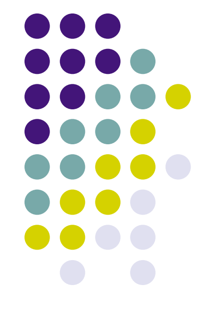

```
def sjekkOgBytt(A, siste):
    konflikt = False
    for i in range(0,siste):
        if A[i] > A[i+1]:
            A[i], A[i+1] = A[i+1], A[i]
            konflikt = True
    return konflikt

def bubbleSort(A):
    for i in reversed(range(1, len(A))):
        if not sjekkOgBytt(A, i):
            break
    return A
```

Kjøretid:



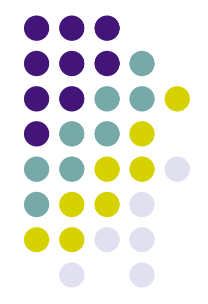
- I beste tilfelle er input allerede sortert, da oppdages det ingen konflikter og algoritmen bruker $\Theta(n)$ tid.
- I verste tilfelle er input sortert i motsatt rekkefølge
  - Element 1 må da flyttes n - 1 plasser, element 2: n - 2 plasser, osv... $\Theta(n^2)$.

Bubblesort:



- Fordeler:
  - $\Theta(n)$ på sorterte lister.
  - Stabil, in-place.

- Ulemper:
  - Generelt treg.
  - Alltid tregere(evt like rask) som insertion sort.

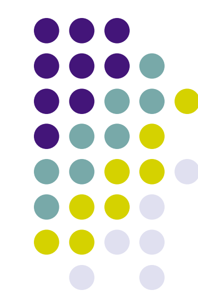


Kvaliteter ved sorteringsalgoritmer

Stabil sortering

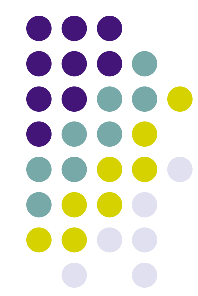
- Tar vare på like elementers plassering i forhold til hverandre og bruker denne til å sortere elementer som har lik verdi.
- Kun nyttig når :
  - Man har flere måter å sortere lista på og vil benytte flere samtidig. Telefonkatalogen i Vestfold er for eksempel sortert etter følgende: By, Etternavn, Fornavn
  - Det finnes like verdier
    (Hvis alle nordmenn bodde i forskjellige byer, ville ikke stabil sortering hatt noe å si)

Første sortering



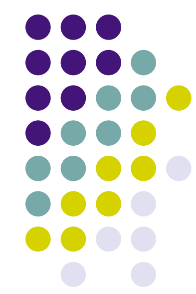
Artist	Tittel	Album
Raga Rockers	Aldri mer	Ubermensch
Queen	Another One Bites The Dust	Greatest Hits I II & III
Briskeby	Berlin	Jeans For Onassis
Briskeby	Between You and Me	Jumping On Cars
Queen	Bicycle Race	Greatest Hits I II & III
Briskeby	Bobby, Come Home	Jumping On Cars
Briskeby	Cellophane Eyes	Jeans For Onassis
Briskeby	Dancing	Tonight, Captain
Queen	Don't Stop Me Now	Greatest Hits I II & III
Briskeby	Electro Boy	Jeans For Onassis
Briskeby	End of summer	Tonight, Captain
Briskeby	Envy	Jeans For Onassis
Raga Rockers	Falsk	Ubermensch
Queen	Fat Bottomed Girls	Greatest Hits I II & III
Queen	Flash	Greatest Hits I II & III
Briskeby	Get lost	Tonight, Captain
Briskeby	Hallelujah	Tonight, Captain
Briskeby	Hey baby	Tonight, Captain
Briskeby	Hey Harvey	Jeans For Onassis
Briskeby	Joe Dallesandro	Jumping On Cars
Raga Rockers	Jungelens lov	Ubermensch
Briskeby	Keep It To Yourself	Jeans For Onassis
Briskeby	Light My Way	Jumping On Cars
Briskeby	Loveenterprise	Jeans For Onassis
Briskeby	Miss You Like Crazy	Jumping On Cars
Queen	Now I'm Here	Greatest Hits I II & III
Raga Rockers	Nærmere deg	Ubermensch
Briskeby	Ocean Drive	Jumping On Cars
Briskeby	Out of Town	Jumping On Cars
Queen	Play The Game	Greatest Hits I II & III
Raga Rockers	Positiv	Ubermensch
Briskeby	Propaganda	Jeans For Onassis
Queen	Queen - Bohemian Rhapsody	Greatest Hits I II & III
Queen	Queen - Crazy Little Thing Called Love	Greatest Hits I II & III
Queen	Queen - Good Old-fashioned Lover Boy	Greatest Hits I II & III

Ustabil, andre sortering

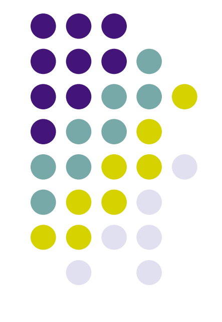


Artist ▼	Tittel	Album
Briskeby	The mess we are in	Tonight, Captain
Briskeby	Light My Way	Jumping On Cars
Briskeby	Keep It To Yourself	Jeans For Onassis
Briskeby	Miss You Like Crazy	Jumping On Cars
Briskeby	Loveenterprise	Jeans For Onassis
Briskeby	Hey Harvey	Jeans For Onassis
Briskeby	Electro Boy	Jeans For Onassis
Briskeby	There is a Wave	Jumping On Cars
Briskeby	Joe Dallesandro	Jumping On Cars
Briskeby	Substitute for Love	Jumping On Cars
Briskeby	Propaganda	Jeans For Onassis
Briskeby	The Asphalt Beach	Jeans For Onassis
Briskeby	Sweet sensation	Tonight, Captain
Briskeby	Ten stories high	Tonight, Captain
Briskeby	Ocean Drive	Jumping On Cars
Briskeby	The electric field	Tonight, Captain
Briskeby	The Asphalt Beach (reprise)	Jeans For Onassis
Briskeby	Out of Town	Jumping On Cars
Briskeby	Cellophane Eyes	Jeans For Onassis
Briskeby	Bobby, Come Home	Jumping On Cars
Briskeby	Shaking Like a Tambourine	Jumping On Cars
Briskeby	Dancing	Tonight, Captain
Briskeby	Berlin	Jeans For Onassis
Briskeby	Wide Awake	Jeans For Onassis
Briskeby	Rebel against your own rules	Tonight, Captain
Briskeby	Hey baby	Tonight, Captain
Briskeby	Stars In Their Eyes	Jeans For Onassis
Briskeby	Trying to lose you	Tonight, Captain
Briskeby	Hallelujah	Tonight, Captain
Briskeby	Envy	Jeans For Onassis
Briskeby	End of summer	Tonight, Captain
Briskeby	Get lost	Tonight, Captain
Briskeby	Between You and Me	Jumping On Cars
Briskeby	Valentine	Tonight, Captain
Queen	Seven Seas Of Rhye	Greatest Hits I II & III

Stabil, andre sortering

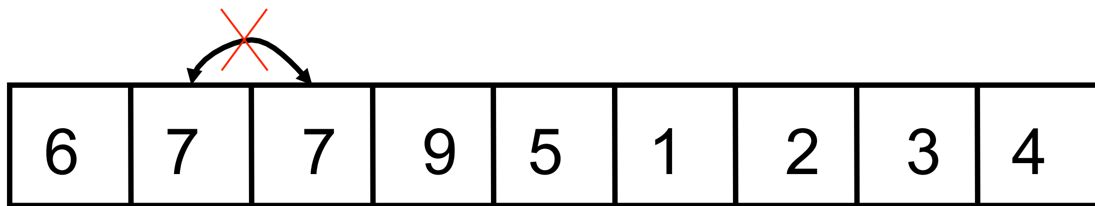


Artist	Tittel	Album
Briskeby	Berlin	Jeans For Onassis
Briskeby	Between You and Me	Jumping On Cars
Briskeby	Bobby, Come Home	Jumping On Cars
Briskeby	Cellophane Eyes	Jeans For Onassis
Briskeby	Dancing	Tonight, Captain
Briskeby	Electro Boy	Jeans For Onassis
Briskeby	End of summer	Tonight, Captain
Briskeby	Envy	Jeans For Onassis
Briskeby	Get lost	Tonight, Captain
Briskeby	Hallelujah	Tonight, Captain
Briskeby	Hey baby	Tonight, Captain
Briskeby	Hey Harvey	Jeans For Onassis
Briskeby	Joe Dallesandro	Jumping On Cars
Briskeby	Keep It To Yourself	Jeans For Onassis
Briskeby	Light My Way	Jumping On Cars
Briskeby	Loveenterprise	Jeans For Onassis
Briskeby	Miss You Like Crazy	Jumping On Cars
Briskeby	Ocean Drive	Jumping On Cars
Briskeby	Out of Town	Jumping On Cars
Briskeby	Propaganda	Jeans For Onassis
Briskeby	Rebel against your own rules	Tonight, Captain
Briskeby	Shaking Like a Tambourine	Jumping On Cars
Briskeby	Stars In Their Eyes	Jeans For Onassis
Briskeby	Substitute for Love	Jumping On Cars
Briskeby	Sweet sensation	Tonight, Captain
Briskeby	Ten stories high	Tonight, Captain
Briskeby	The Asphalt Beach	Jeans For Onassis
Briskeby	The Asphalt Beach (reprise)	Jeans For Onassis
Briskeby	The electric field	Tonight, Captain
Briskeby	The mess we are in	Tonight, Captain
Briskeby	There is a Wave	Jumping On Cars
Briskeby	Trying to lose you	Tonight, Captain
Briskeby	Valentine	Tonight, Captain
Briskeby	Wide Awake	Jeans For Onassis
Queen	Another One Bites The Dust	Greatest Hits I II & III

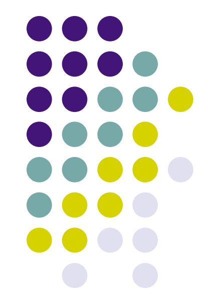


Bubblesort stabilitet:

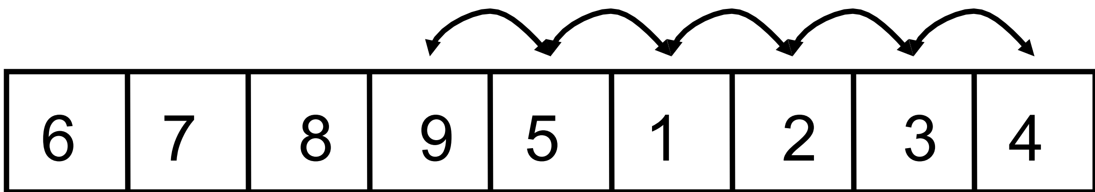
- Bubblesort vil kun bytte to elementer dersom et av dem er større enn det andre.
- Dvs. to like elementer kan ikke komme forbi hverandre.



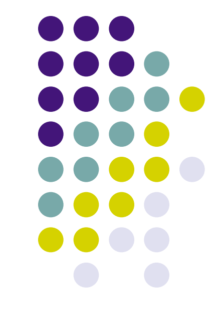
In-place sortering:



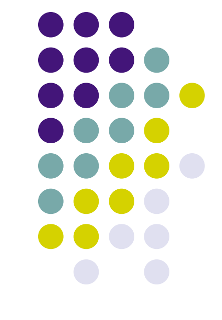
- Betyr vanligvis at algoritmen kun bytter på to tall av gangen (Alle algoritmene i pensum).
- Bruker relativt lite minne (O(1) i tillegg til selve lista).
- Bubblesort:
  - Bytter alltid på to og to elementer:



Insertion sort:



- Del input i to deler:
  - En sortert del. (Først tom)
  - En usortert. (Alle elementene ligger her i starten)
- Så lenge det finnes usorterte elementer:
  - Sett et usortert element på riktig plass i den sorterte delen.



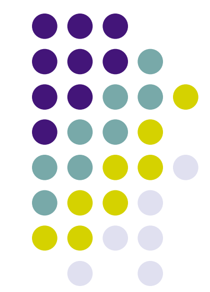
Eksempel:

4	3	2	1	5	9	8	7	6

3	4	2	1	5	9	8	7	6

2	3	4	1	5	9	8	7	6

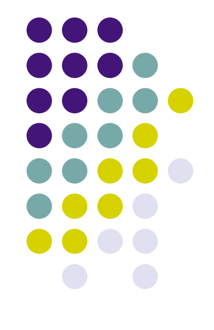
1	2	3	4	5	9	8	7	6



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

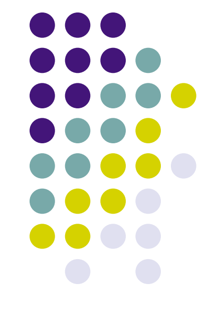
Ferdig!

Kjøretid:



- I beste tilfelle er listen allerede sortert
  - Da står neste element alltid på riktig plass, slik at det bare utføres totalt n-1 sammenligniger og ingen bytter
  - kjøretiden blir $\Theta(n)$
- I verste tilfelle er input sortert i motsatt rekkefølge
  - Da må element 2: flyttes 1 plass, element 3: 2 plasser, ... , element n: n-1 plasser
  - Kjøretiden blir $\Theta(n^2)$

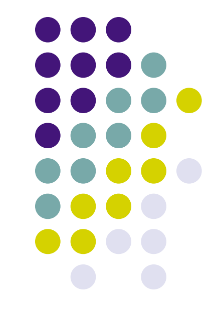
Insertion sort:



- Fordeler:
  - Optimal på sortert input.
  - Rask på nesten sortert input.
  - Stabil, in-place
  - Alltid raskere enn (evt. like rask som) Bubblesort.

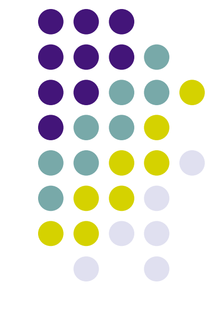
- Ulemper:
  - Treg på store input da den har worst case $\Theta(n^2)$ kjøretid.

Selection sort:



- Del input i to deler:
  - En sortert del
  - En usortert
- Så lenge det finnes usorterte elementer:
  - Finn største element i den usorterte delen, og flytt dette foran i den sorterte delen.

Eksempel:

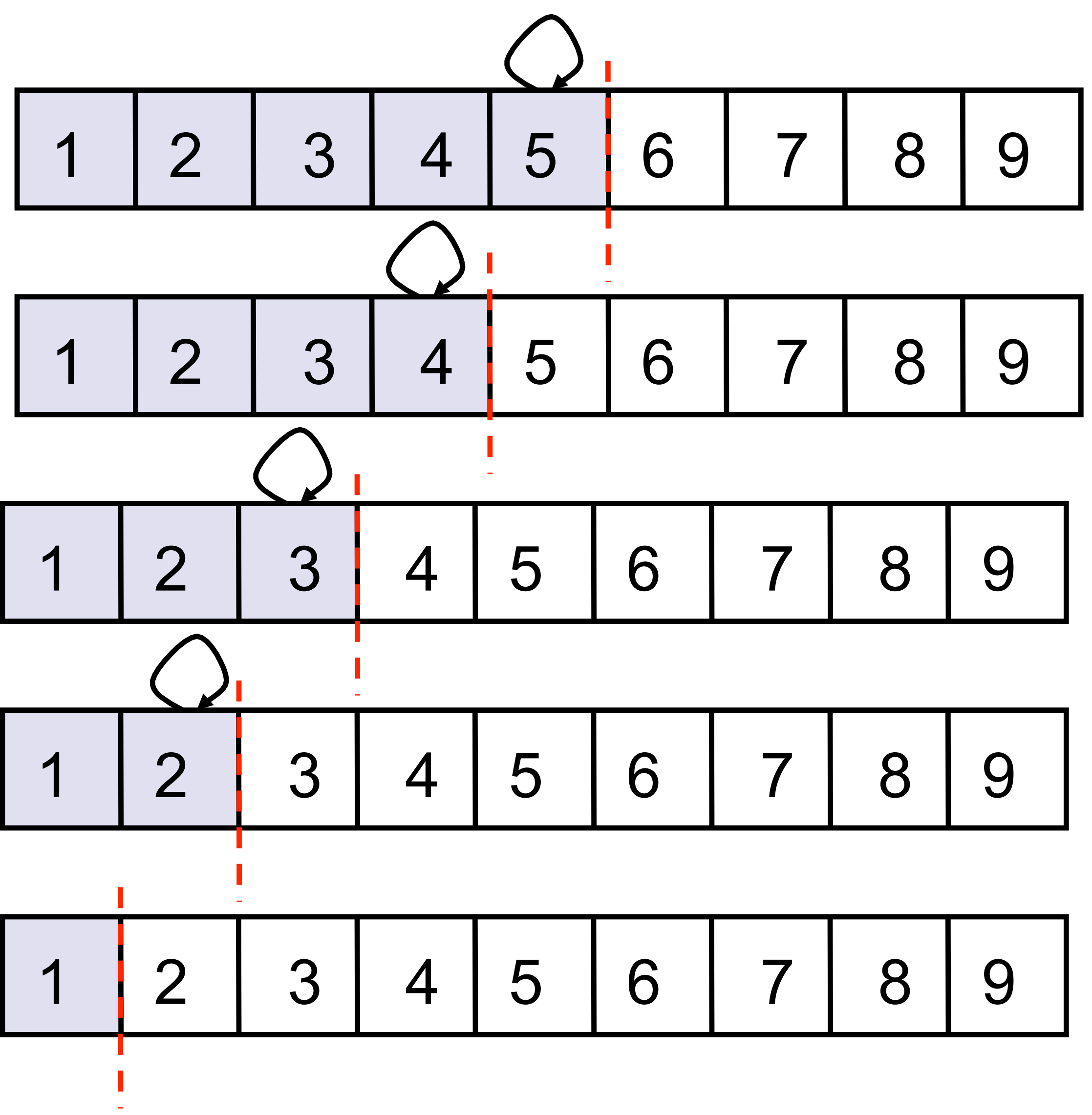


6	7	8	9	5	1	2	3	4

6	7	8	4	5	1	2	3	9

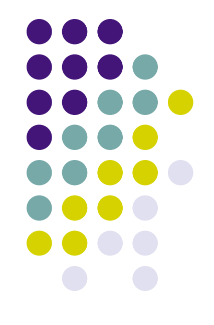
6	7	3	4	5	1	2	8	9

6	2	3	4	5	1	7	8	9



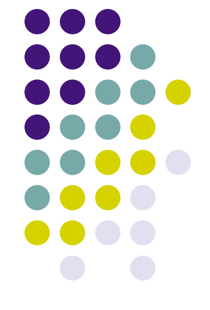
Ferdig!

Kjøretid:



- Selection sort vil alltid bruke $\Theta(n^2)$ tid:
  - Den må alltid gå igjennom alle de usorterte elementene for å finne det største. Deretter må dette flyttes. Dermed vil tiden alltid være lik.
- Samtidig vil det kun bli $O(n)$ flyttinger. På maskiner der dette er en dyr operasjon er dette en stor fordel.

Selection sort:



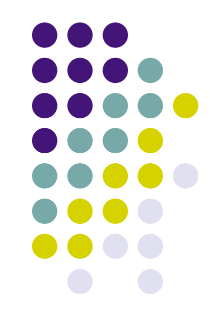
- Fordeler:
  - Alltid like rask.
  - Få skriveoperasjoner.
  - In-place.

- Ulemper:
  - Alltid like treg.
  - Ikke stabil (stabil for lenket liste).

Egenskaper:

	Bubblesort	Insertion Sort	Selection Sort
Stabil	✓	✓	✓ For lenket liste.
Parallelliserbar			
In-place	✓	✓	✓

Trestrukturer: Begreper Repetisjon



Rot

R

B og C's
Foreldre-node

Kanter

Noder

Subtre

A

D

H

Løvnoder

B

C

F

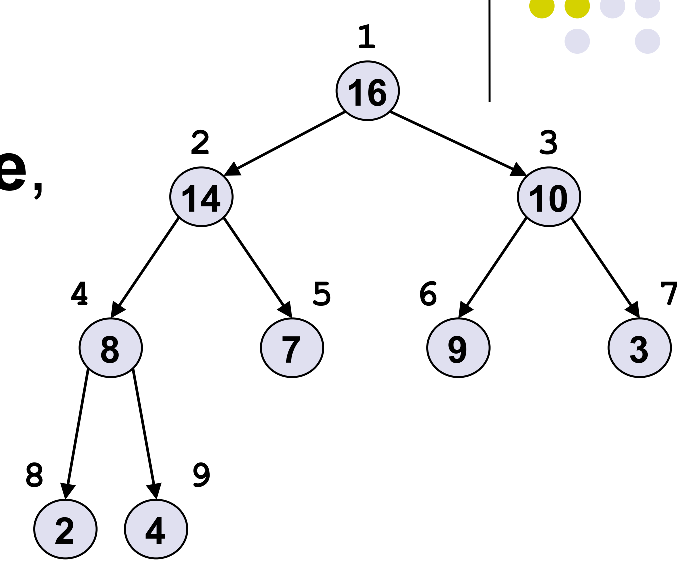
G

A's venstre barn

A's høyre barn

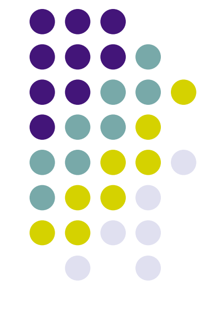
Søskennoder

Hva er en Heap?

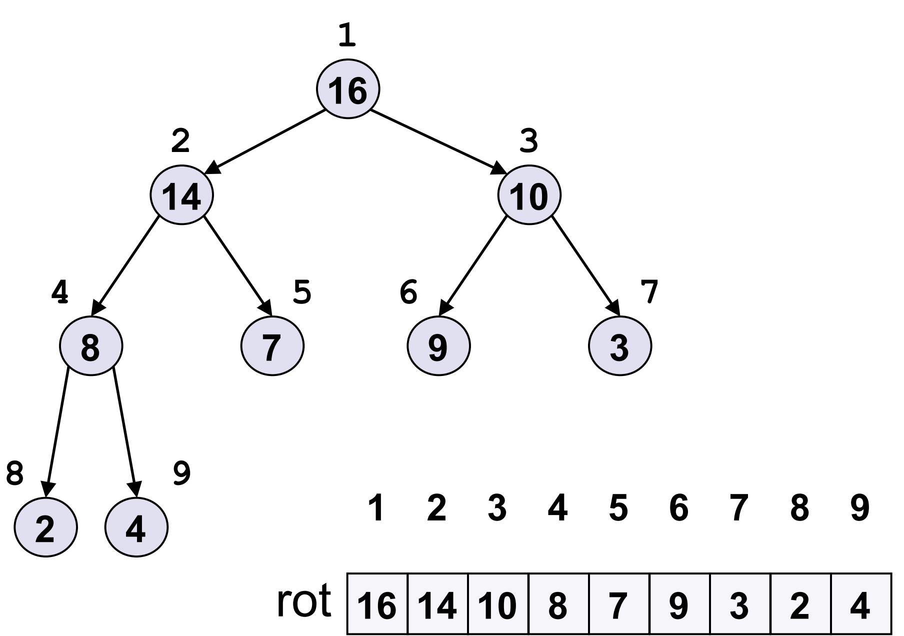


- En heap er et **komplett binærtre**,
  - der alle nivå er fylt opp,
  - untatt eventuelt det siste,
  - som er fylt opp fra venstre til høyre
- For alle noder i en heap gjelder:
  - Barna har lavere eller lik verdi(for max-heap)
  - Barna har større eller lik verdi(for min-heap)
- En heap **brukes til/av** blant annet
  - Prioritetskøer
  - Heapsort

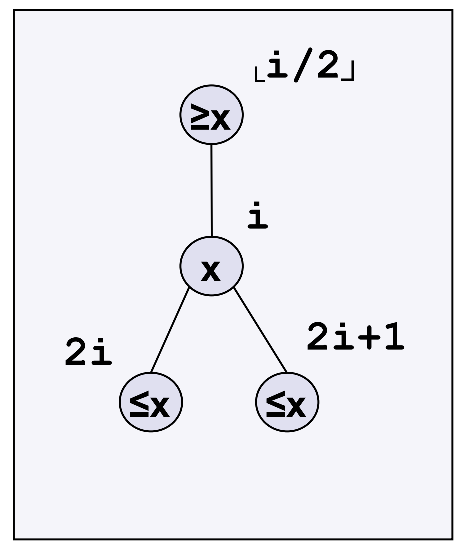
Heap : Datastruktur



- Representasjon av et binærtre i et array:

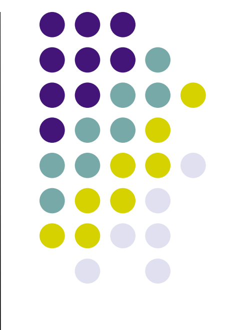


Generelle far/barn relasjoner

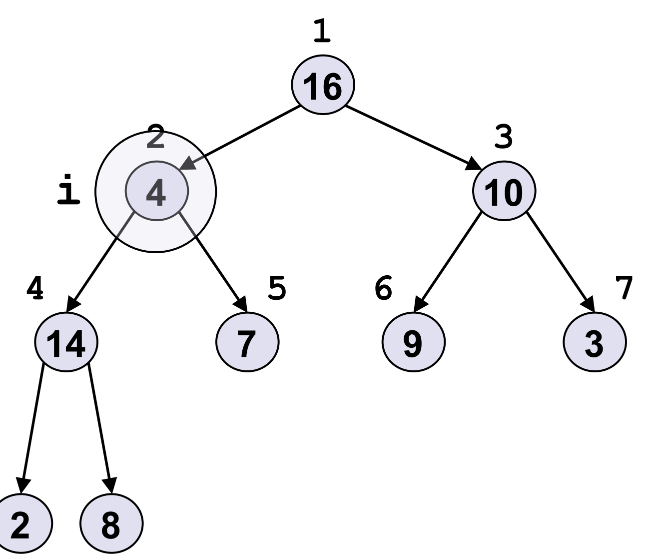


(Heap-egenskapen)

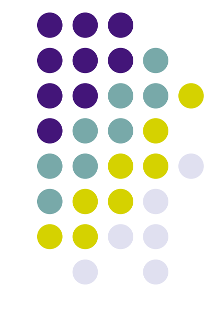
”Max-Heapify”: O(lg n)



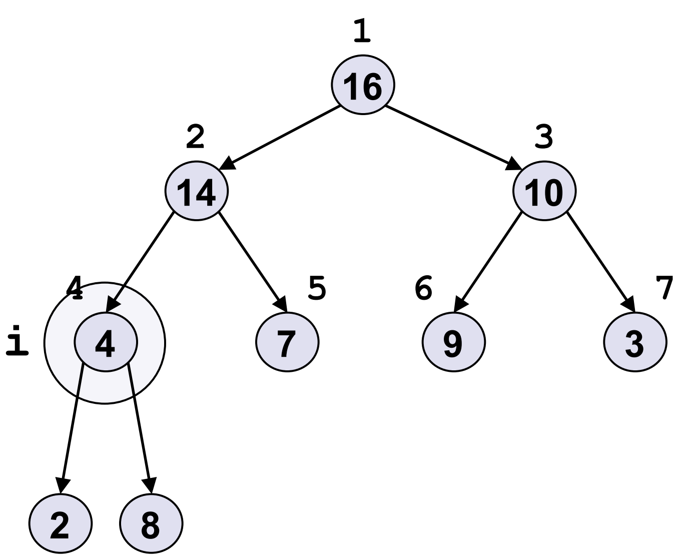
- Subrutine som **vedlikeholder Heap-egenskapen**,
- ...dvs at barna til roten er mindre enn roten




```
Max-Heapify(A,i)
   l = left(i)
   r = right(i)
   if l ≤ heap-size[A] and A[l] > A[i]
      then largest = l
      else largest = i
   if r ≤ heap-size[A] and A[r] > A[largest]
      then largest = r
   if largest ≠ i
      then exchange A[i]↔ A[largest]
         Max-Heapify(A,largest)
```

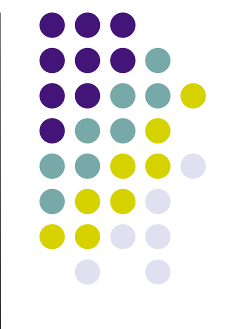


- Subrutine som **vedlikeholder Heap-egenskapen**,
- ...dvs at barna til roten er mindre enn roten

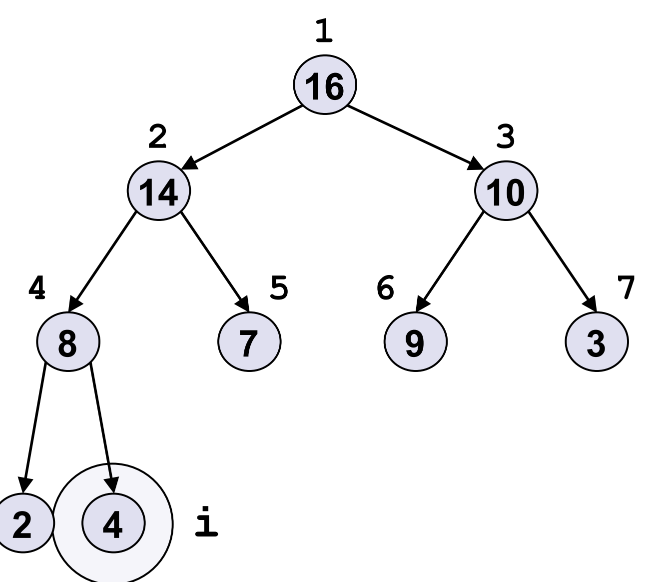



```
Max-Heapify(A,i)
   l = left(i)
   r = right(i)
   if l ≤ heap-size[A] and A[l] > A[i]
      then largest = l
      else largest = i
   if r ≤ heap-size[A] and A[r] > A[largest]
      then largest = r
   if largest ≠ i
      then exchange A[i]↔ A[largest]
         Max-Heapify(A,largest)
```

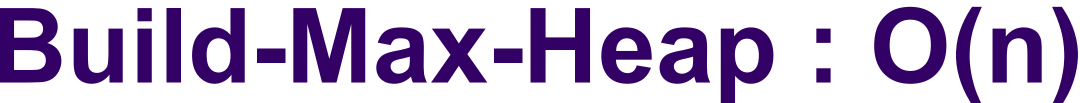
”Max-Heapify”: O(lg n)



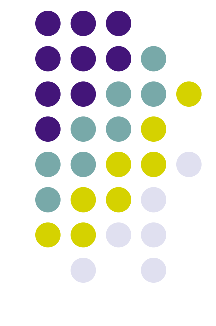
- Subrutine som **vedlikeholder Heap-egenskapen**,
- ...dvs at barna til roten er mindre enn roten



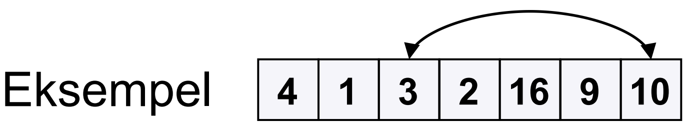

```
Max-Heapify(A,i)
  l = left(i)
  r = right(i)
  if l ≤ heap-size[A] and A[l] > A[i]
    then largest = l
    else largest = i
  if r ≤ heap-size[A] and A[r] > A[largest]
    then largest = r
  if largest ≠ i
    then exchange A[i] ↔ A[largest]
      Max-Heapify(A,largest)
```

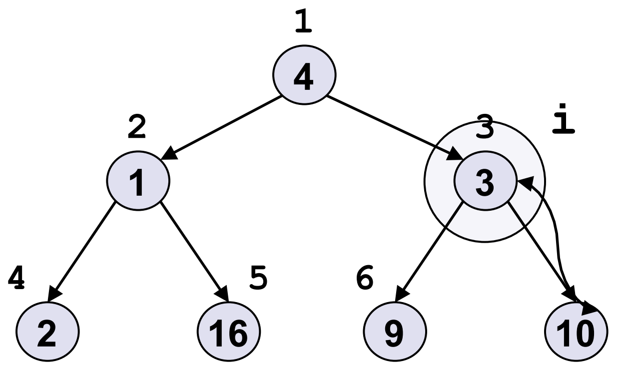


Build-Max-Heap : O(n)

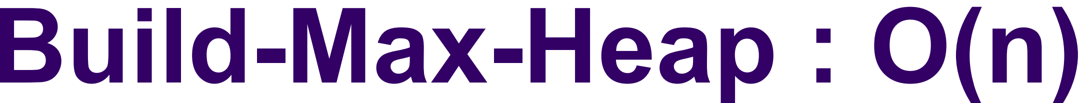


- Lager en heap fra et (ikke-ordnet) array,
- i angir de indre nodene i heap'en

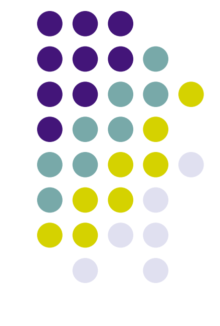




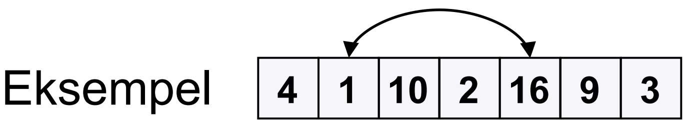

```
Build-Max-Heap(A)
  heap-size[A] = length[a]
  for i = [length[a]/2] downto 1
    do Max-heapify(A,i)
```

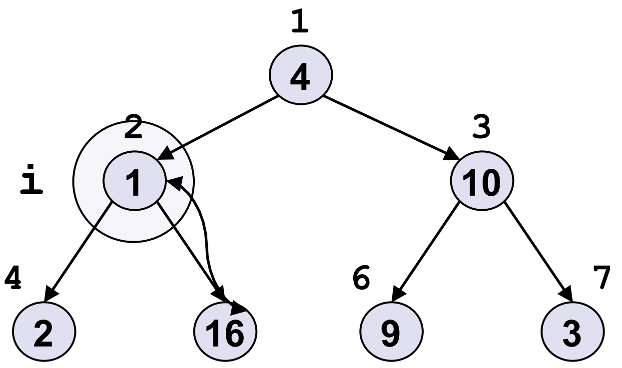


Build-Max-Heap : O(n)

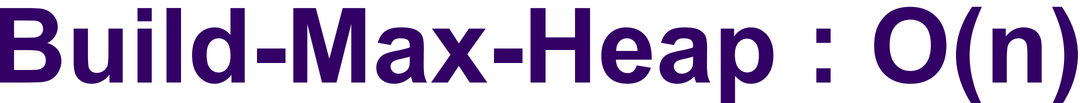


- Lager en heap fra et (ikke-ordnet) array,
- i angir de indre nodene i heap'en

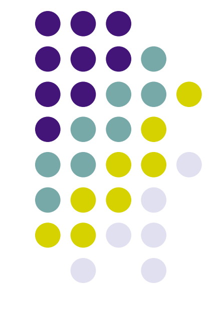




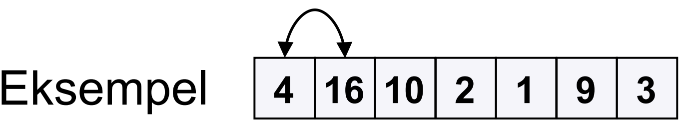

```
Build-Max-Heap(A)
  heap-size[A] = length[a]
  for i = [length[a]/2] downto 1
    do Max-heapify(A,i)
```

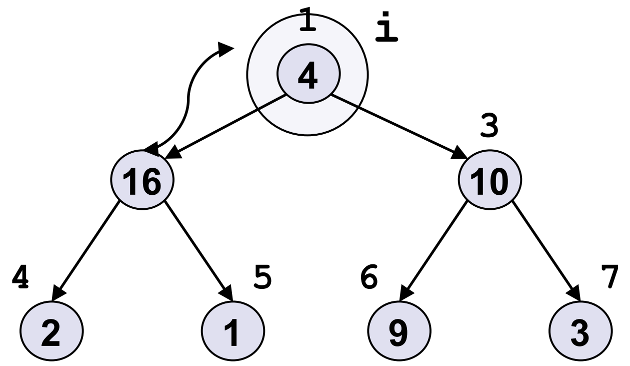


Build-Max-Heap : O(n)

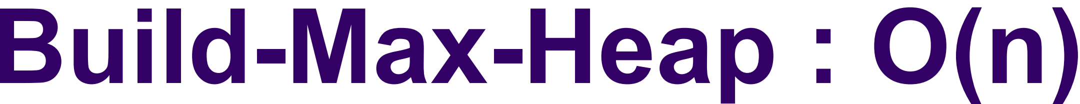


- Lager en heap fra et (ikke-ordnet) array,
- i angir de indre nodene i heap'en

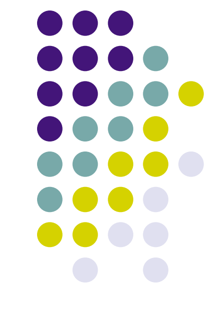





```
Build-Max-Heap(A)
  heap-size[A] = length[a]
  for i = [length[a]/2] downto 1
    do Max-heapify(A,i)
```



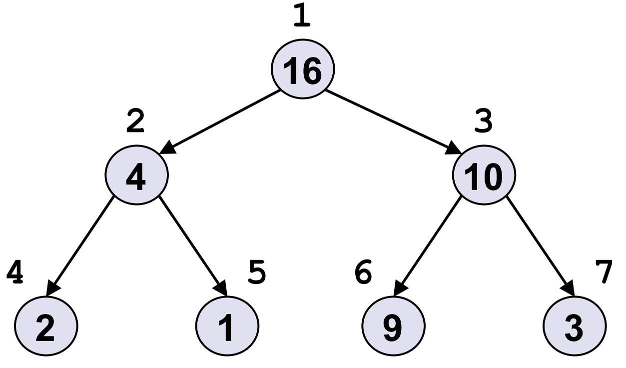
Build-Max-Heap : O(n)



- Lager en heap fra et (ikke-ordnet) array,
- i angir de indre nodene i heap’en

Eksempel

16	4	10	2	1	9	3




```
Build-Max-Heap(A)
  heap-size[A] = length[a]
  for i = [length[a]/2] downto 1
    do Max-heapify(A,i)
```

Heapsort-Algoritmen:

- Lag heap av tallene (i arrayet)
- For siste posisjon til 2.:
  - Bytt med rota, og la heap-størrelsen minke med 1
  - Kall Max-heapify på rotnoden for å gjenopprette heapegenskapen.
- Selection sort, men med heap.

Eksempel

14	8	7	2	4	1


```
Heapsort(A)
  for i = [length[a]] downto 2
    do exchange A[1]↔A[i]
      heap-size[A] = heap-size[A]-1
      Max-Heapify(A,1)
```

Heapsort : O(n*lg n)

Eksempel

14	8	7	2	4	1

Usortert

```
Heapsort(A)
  for i = [length[a]] downto 2
    do exchange A[1]↔A[i]
      heap-size[A] = heap-size[A]-1
      Max-Heapify(A,1)
```

Heapsort : O(n*lg n)

Eksempel

14	8	7	2	4	1

Usortert


```
Heapsort(A)
  for i = [length[a]] downto 2
    do exchange A[1]↔A[i]
      heap-size[A] = heap-size[A]-1
      Max-Heapify(A,1)
```

Heapsort : O(n*lg n)

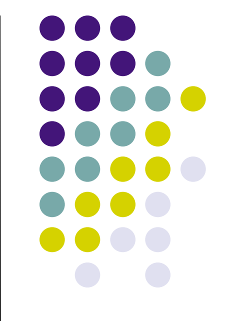
Eksempel

14	8	7	2	4	1

Usortert


```
Heapsort(A)
  for i = [length[a]] downto 2
    do exchange A[1]↔A[i]
      heap-size[A] = heap-size[A]-1
      Max-Heapify(A,1)
```

Heapsort : O(n*lg n)



Eksempel

14	8	7	2	4	1

Usortert


```
Heapsort(A)
  for i = [length[a]] downto 2
    do exchange A[1]↔ A[i]
      heap-size[A] = heap-size[A]-1
      Max-Heapify(A,1)
```

Heapsort : O(n*lg n)

Eksempel

14	8	7	2	4	1

Usortert

Sortert : A

1	2	4	7	8	14

```
Heapsort(A)
  for i = [length[a]] downto 2
    do exchange A[1]↔ A[i]
      heap-size[A] = heap-size[A]-1
      Max-Heapify(A,1)
```

i 1 2 4 7 8 14 (5)

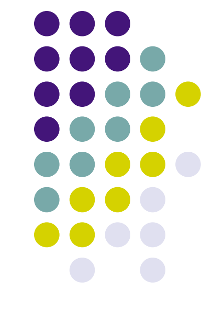
1 2 4 7 8 14 (2)

Max-Heapify(A,1)

Kapplinje

1 2 4 7 8 14 (3)

Heapsort : O(n*lg n)



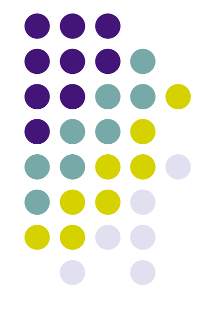
- Kjøretid til heapsort
  - Heapsort bruker **O(n lg n) tid**
    - Siden kallet til Build-Max-Heap tar O(n) tid og
    - Hver av de n-1 kallene til Heapify tar O(lg n) tid
    - O(n) + O(n lg n) = O(n lg n)

```
Heapsort(A)
  for i = [length[a]] downto 2
    do exchange A[1]↔ A[i]
       heap-size[A] = heap-size[A]-1
       Max-Heapify(A,1)
```

Prioritetskøer bruker Heap

- En prioritetskø er en datastruktur for å holde vedlike et sett av S verdier
- Vi har to typer prioritetskøer:
  - En max-prioritets-kø
  - En min-prioritets-kø

Prioritets-kø : Operasjoner

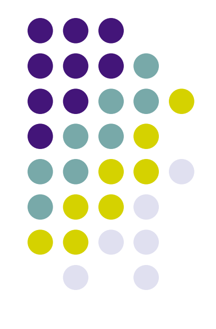


- Operasjoner på en **max-prioritets-kø**
  - Insert(S,x), kjøretid O(lg n)
    - Setter en node inn i heapen
  - Maximum(S), kjøretid O(1)
    - returnerer elementet S med den største verdien
  - Extract-Max(S), kjøretid O(lg n)
    - Returnerer elementet S med den største verdien
    - ...og fjerner elementet fra heapen

Prioritets-kø : Operasjoner

- Operasjoner på en **max-prioritets-kø**
  - Increase-Key(S,x,k), kjøretid O(lg n)
    - Øker verdien til et element x til en ny verdi k,
    - ...som en antar er minst like stor som x
- Operasjoner på en **min-prioritets-kø**
  - Insert(S,x), Minimum(S), Extract-Min(S), Decrease-Key(S,x,k)

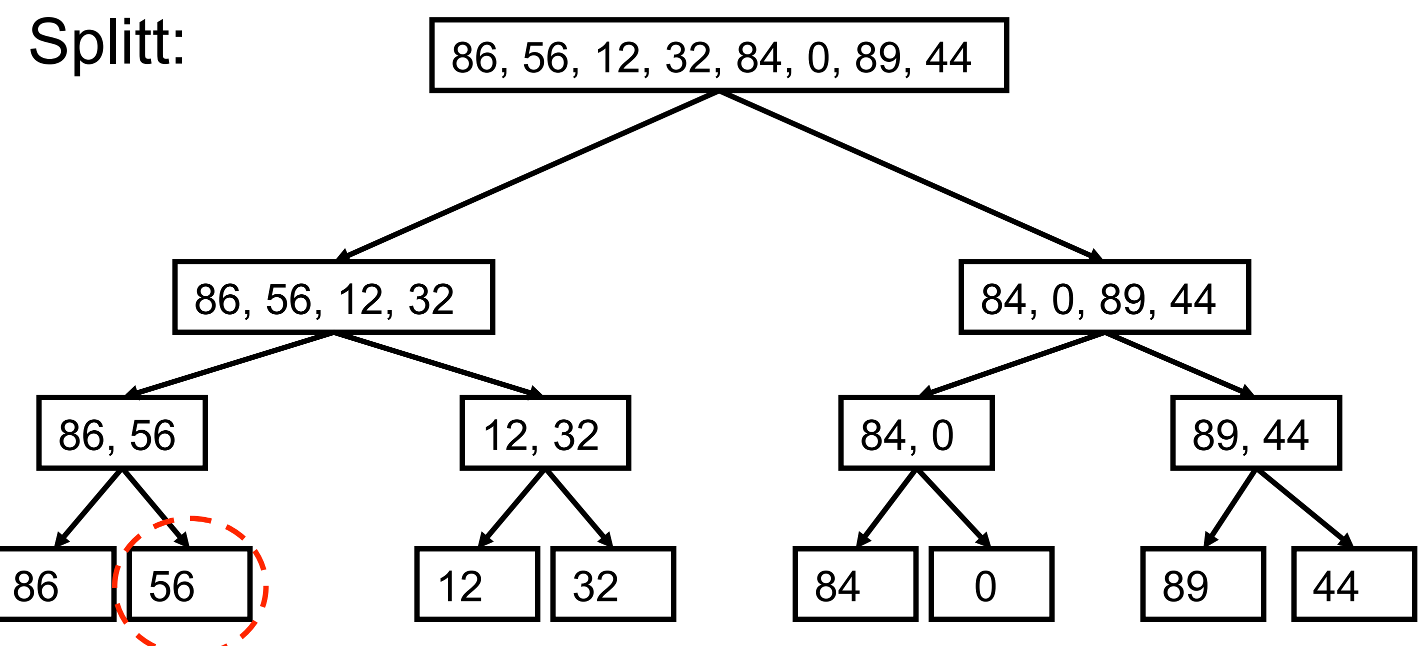
Merge sort:

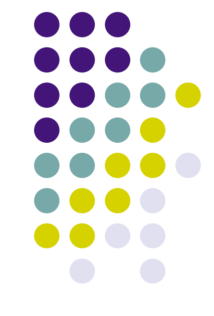


- Splitt og hersk-sortering.
- Del input i to like store biter.
- Kjør mergesort rekursivt på hver av de to bitene.
- Flett (merge) de to bitene sammen:
  - Velg det minste av bitenes første elementer helt til begge er tomme.
- Returner den flettede listen.

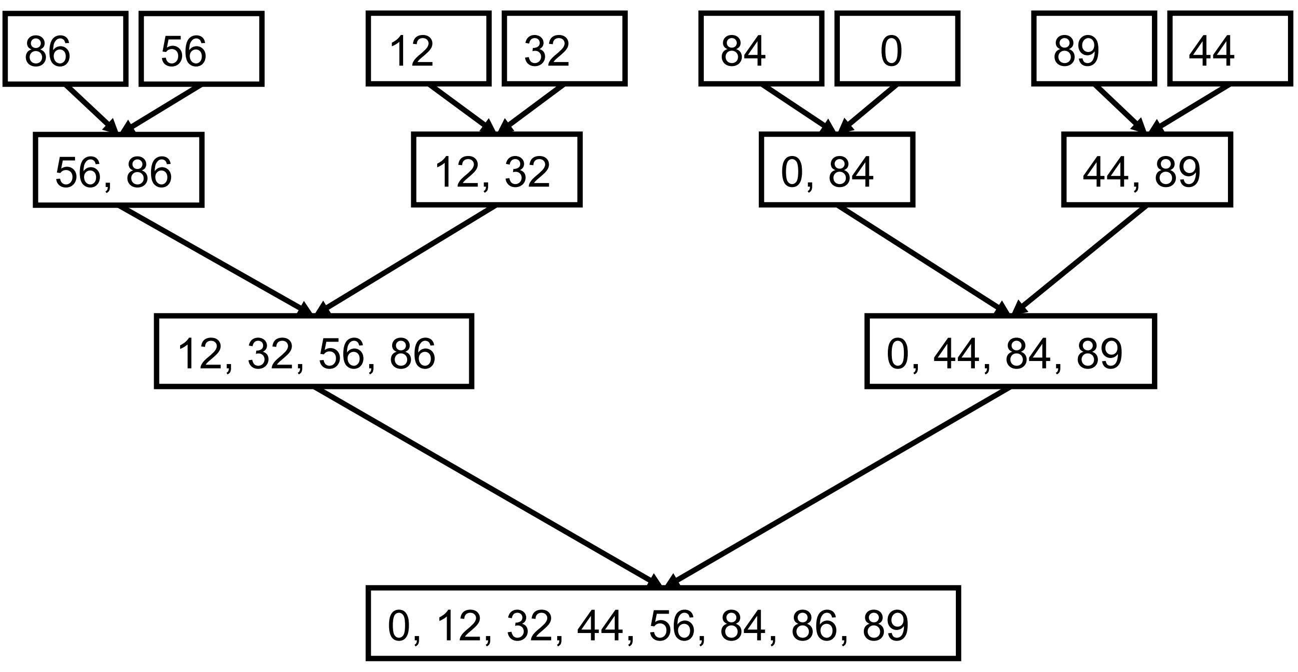
Eksempel: merge sort

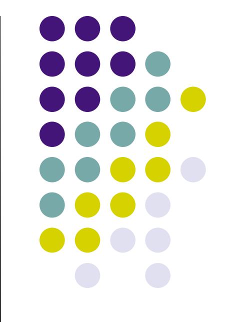
Splitt:



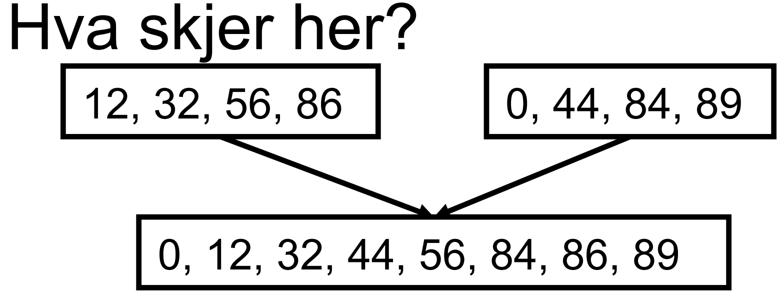


Hersk(Merge):





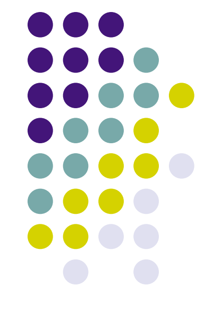
Merge sort



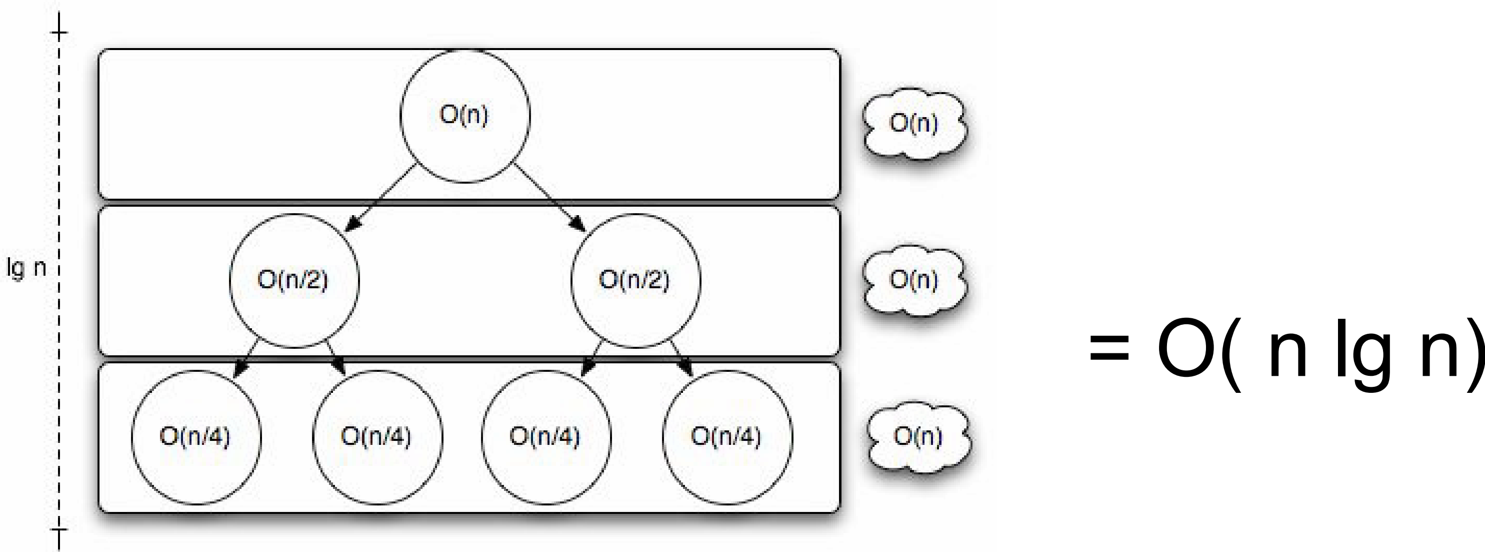
12	32	56	86
0	44	84	89

0	12	32	44	56	84	86	89

Kjøretid: Merge sort

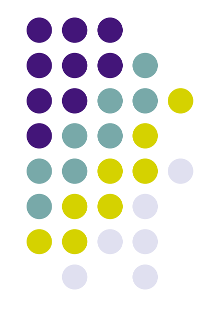


- T(n) = 2*T(n/2)   + O(n)
- Tid = Rekursive kall + Fletting



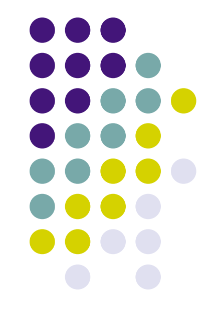
= O(n lg n)

Merge sort:



- Fordeler:
  - Worst-case kjøretid $\Theta(n \lg n)$ for generelle sorteringsproblemer
  - Deler opp i uavhengige oppgaver, dvs parallelliserbar
  - Stabil, hvis merge implementeres riktig
  - Fungerer godt på lenket liste
- Ulemper:
  - Trenger $\Theta(n)$ ekstra minne, ikke in-place

Quicksort:



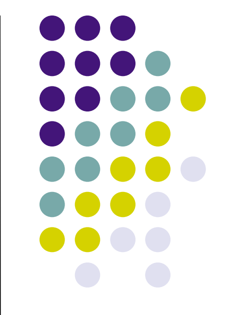
- Til forveksling lik mergesort, men:
  - ”Hersker før splitt”
  - Bruker mindre minne
- Istedet for fletting (merge) brukes partisjonering.

Quicksort: Partisjonering

- Velger et element i input som kalles ”pivot element”
- Deler resten av input i to deler:
  - En del der alle tallene er mindre eller lik pivot elementet
  - En annen del der alle tallene er større enn pivot elementet.

Quicksort: fremgangsmåte

- Quicksort partisjonerer input og kaller seg selv på hver av de to partisjonene.
- Quicksort avslutter når det bare er ett element igjen.



Quicksort

Pivot element

86, 56, 12, 32, 84, 0, 89, 44

0, 12

32

84, 89, 44, 86, 56

0

12

44

56

84, 89, 86

84

86

89

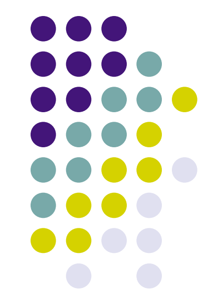
84, 86, 89

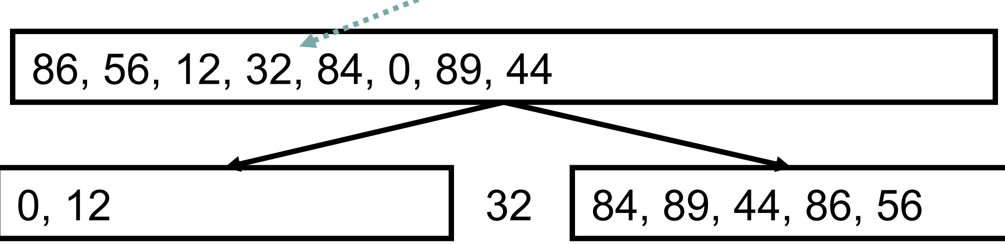
0, 12

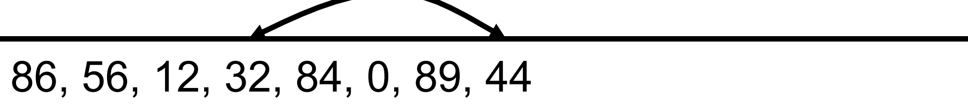
44, 56, 84, 86, 89

0, 12, 32, 44, 56, 84, 86, 89

Quicksort





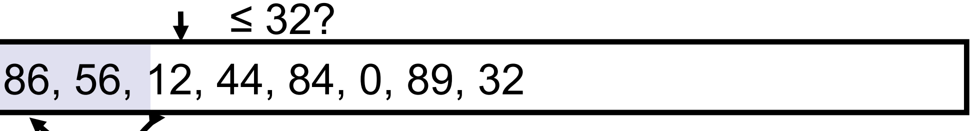


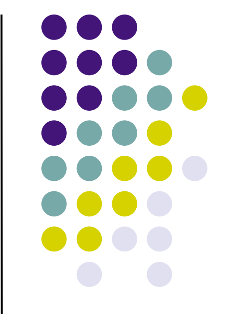
≤32?

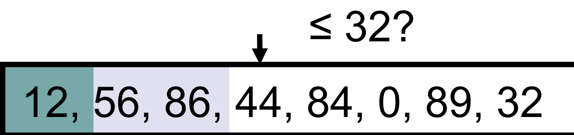
86, 56, 12, 44, 84, 0, 89, 32

≤ 32?

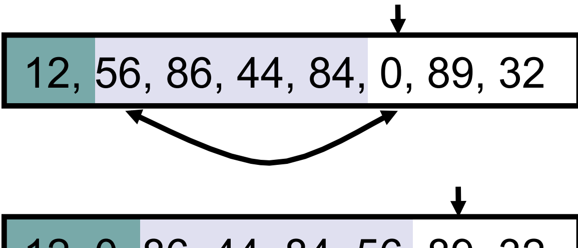
86, 56, 12, 44, 84, 0, 89, 32



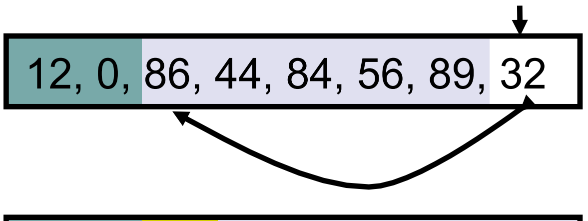




12, 56, 86, 44, 84, 0, 89, 32



12, 0, 86, 44, 84, 56, 89, 32

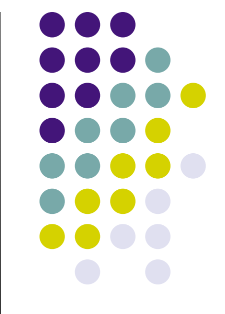


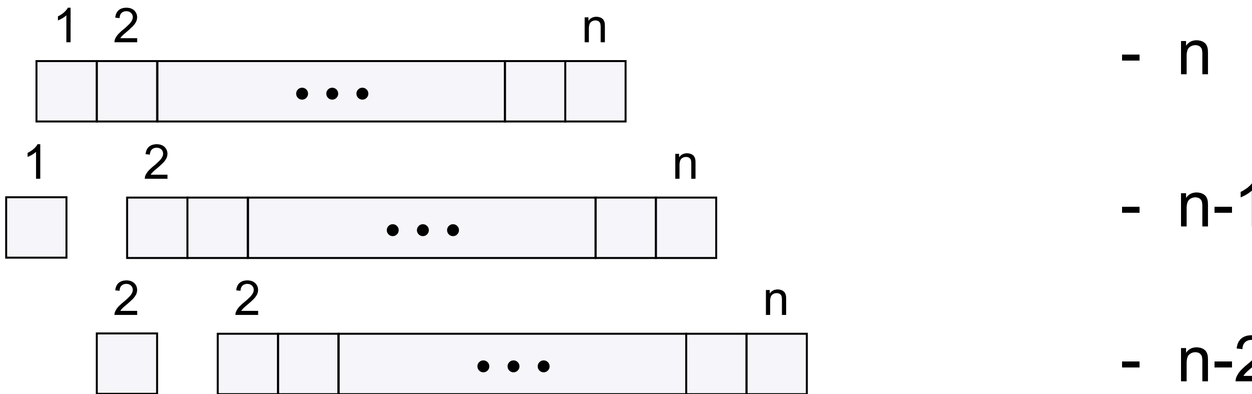
12, 0, 32, 44, 84, 56, 89, 86

Quicksort : Algoritmen

```
Quicksort(A,p,r):
  if p <r
  then q = Partition(A,p,r)↰
    Quicksort(A,p,q-1)↰
    Quicksort(A,q+1,r)↰


Partition(A,p,r):
  x = A[r]
  i = p-1
  for j = p to r-1
    do if A[j] ≤ x
      then i = i+1
        exchange A[i]↔A[j]
  exchange A[i] ↔ A[j]
  return i+1
```

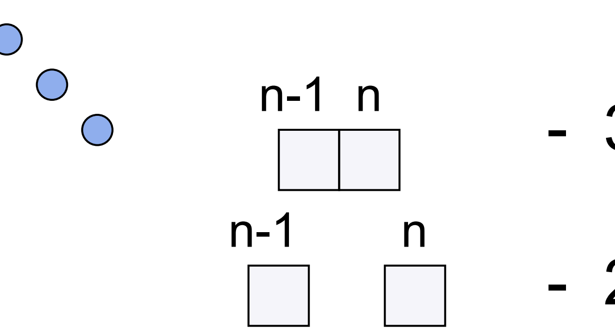
"Worst case Partition"





- n
- n-1
- n-2

Dårlig pga **svært skjeve splittinger**; en region med bare ett element og den andre regionen (subarray) inneholder resten av elementene.

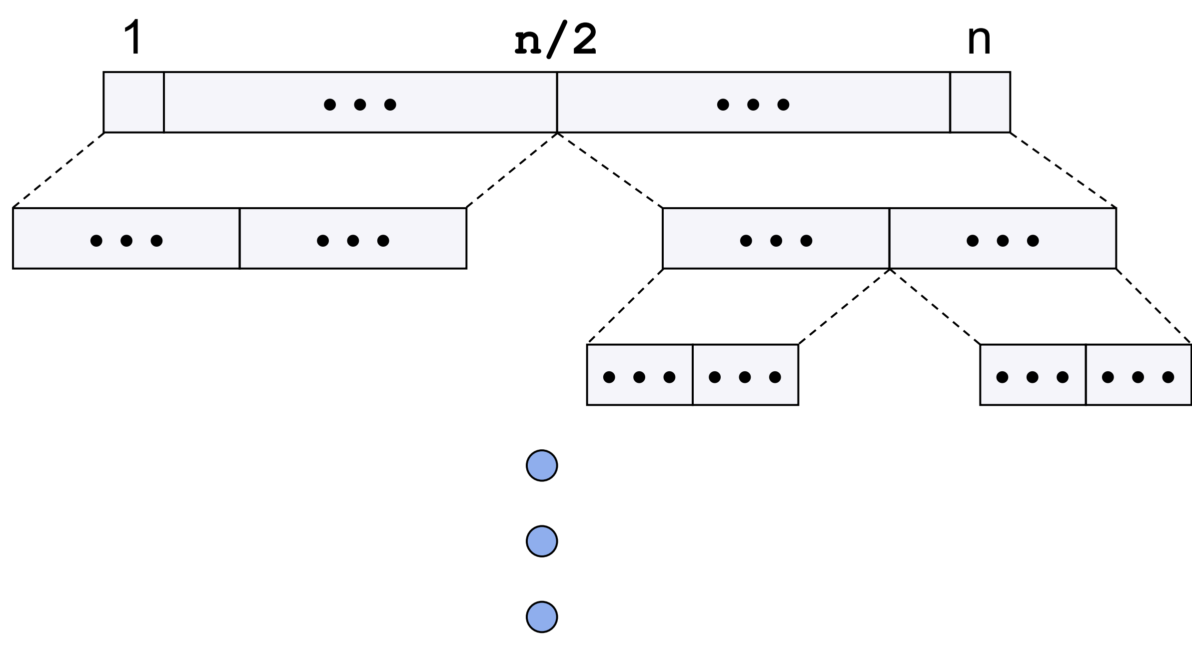


- 3
- 2

$\Theta(n^2)$ hvis input-array allerede er sortert

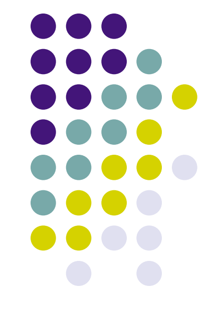
$T(n) = T(n-1) + \Theta(n)$ $\mathbf{\Theta(n^2)}$

"Best Case Partition"



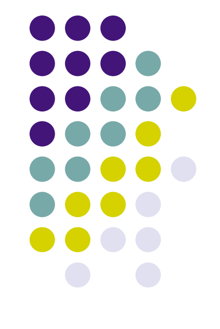
$T(n) = 2T(n/2) + \Theta(n)$ ⇒ $\Theta(n \lg n)$

Valg av pivot:



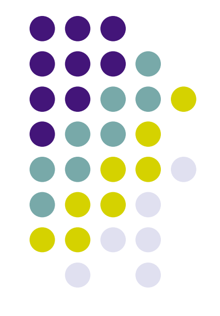
- Valg av pivot har stor innflytelse på kjøretiden til quicksort.
- For sortert input vil valg av første/siste element alltid gi kjøretid $\Theta(n^2)$.
- Om man ikke kan finne et pivot element i O (1) vil det påvirke ytelsen.
- Tilfeldig valgt pivot gir god ytelse, og worst case kjøretid $\Theta(n^2)$ oppstår med ekstremt lav sannsynlighet.

Quicksort:

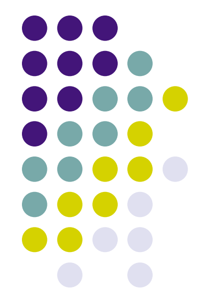


- Fordeler:
  - Effektiv på store lister
  - Deler opp i uavhengige oppgaver
  - Kan kombineres med andre sorteringsalgoritmer når listene blir små for å øke ytelsen (dette er det mest populære valget blant generelle sorteringsalgoritmer)
- Ulemper:
  - Vanskelig å velge gode pivot-elementer uten å påvirke ytelsen.

Egenskaper:



	Heap sort	Merge Sort	Quicksort
Stabil		✓	
Parallelliserbar		✓	✓
In-place	✓		✓



Oppsummering kvaliteter

Stabil

En stabil sorteringsalgoritme tar vare på den eksisterende rekkefølgen til elementer som har samme verdi.

Parallelliserbar

Sorteringen kan splittes opp i deler som kan utføres uavhengig av hverandre.

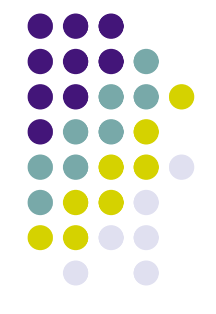
In Place

Elementene kan befinne seg i lista under hele sorteringen.

Telle-Sortering (Counting-Sort)

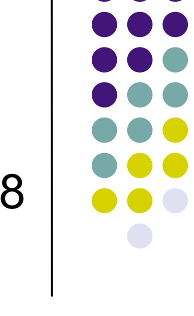
- Sorteringsrutinen antar at hvert av de n-elementene, som skal sorteres, er **heltall mellom 1 og k**, der k er et heltall
- **Idé**: For hvert element x skal vi finne antall elementer **mindre enn eller lik x**.
  - Informasjonen brukes til å plassere x direkte i det sorterte arrayet

Plassforbruk Tellesortering



- Sorteringsrutinen **krever stor plass** da 3 array brukes
  - Array A[1..n] som skal sorteres
  - Array B[1..n] er sortert resultat
  - Array C[1..k] er temporært arbeidslager
- Algoritmen **sorterer i lineær tid**
  - $\Theta(n)$, dersom $k = O(n)$
  - Generelt: $\Theta(n + k)$

Eksempel



	1	2	3	4	5	6	7	8	j
A	3	6	4	1	3	4	1	4	

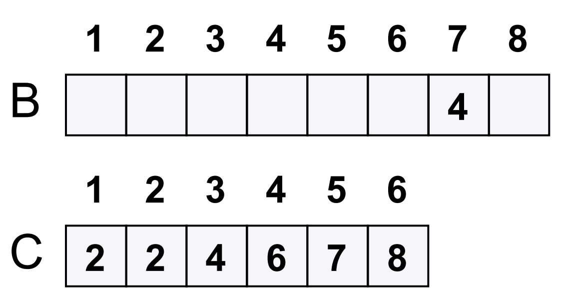
Array som skal sorteres

	1	2	3	4	5	6
C	2	0	2	3	0	1

Antall 4'ere i A

	1	2	3	4	5	6
C	2	2	4	7	7	8

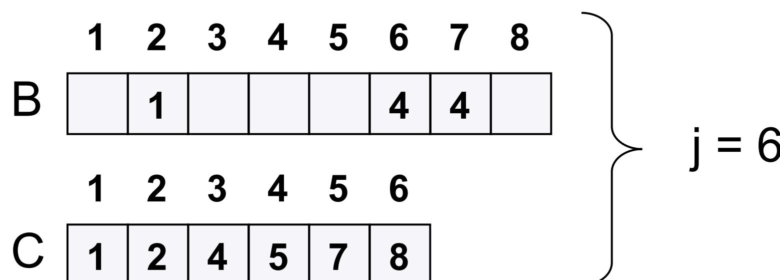
Antall elementer ≤ 3



	1	2	3	4	5	6	7	8
B		1					4	

	1	2	3	4	5	6
C	1	2	4	6	7	8

j = 7



	1	2	3	4	5	6	7	8
B		1		3		4	4	

	1	2	3	4	5	6
C	1	2	3	5	7	8

j = 5

Eksempel

A

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

Array som skal sorteres

C

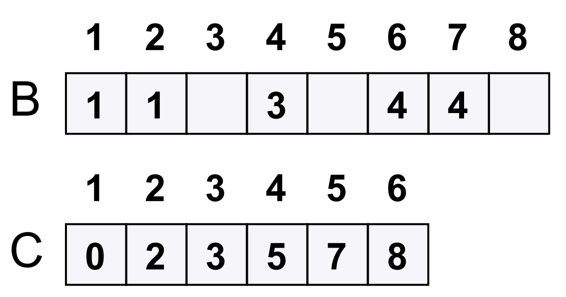
1	2	3	4	5	6
2	0	2	3	0	1

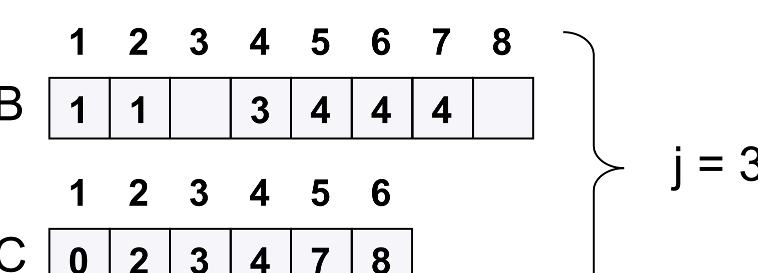
Antall 4'ere i A

C

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

Antall elementer ≤ 3





j = 2

B

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

C

1	2	3	4	5	6
0	2	3	4	7	7

j = 1

B

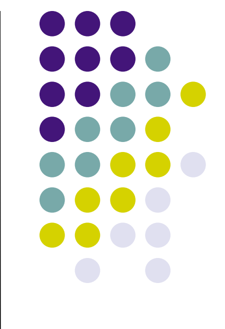
1	2	3	4	5	6	7	8
1	1	3	3	4	4	4	6

C

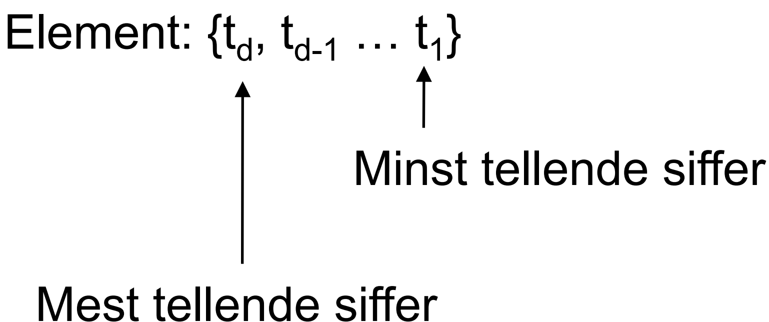
1	2	3	4	5	6
0	2	2	4	7	7

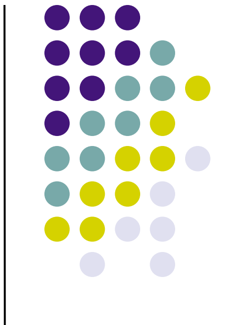
Ferdig!

Radix-sort



- Radix-sort sorterer input i flere omganger.
- Først på minst tellende siffer, så nest minst tellende, osv til den til slutt sorterer på mest tellende siffer.



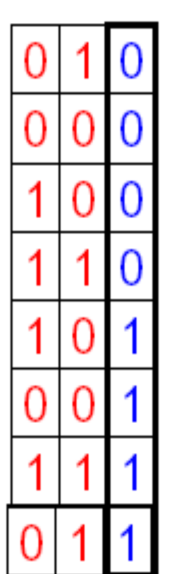


Radix-sort

- Sorterer bits fra høyre til venstre

```
Radix-Sort(A,d)
  for i=1 to d
     do use a stable sort to sort
     array A on digit i
```

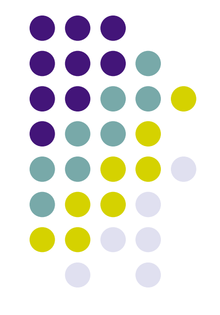
0	1	0
0	0	0
1	0	1
0	0	1
1	1	1
0	1	1
1	0	0
1	1	0



0	1	0
0	0	0
1	0	0
1	1	0
1	0	1
0	0	1
1	1	1
0	1	1

0	0	0
1	0	0
1	0	1
0	0	1
0	1	0
1	1	0
1	1	1
0	1	1

0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1



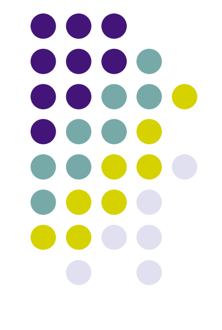
Analyse Radix-Sort

- Kjøretiden avhenger av hvilken sorteringsalgoritme som blir brukt
  - Ved bruk av **Counting-sort**: $\Theta(d(n + k))$
    - d er antall siffer. (k er tall i tallsystemet)
    - Nar d er konstant og $k = O(n) \rightarrow \Theta(n)$
    - Krever ekstra lagerplass, ikke in-place

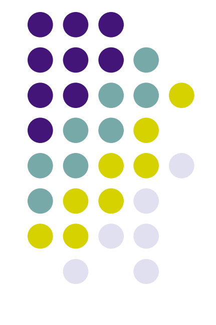
Bucket sort:

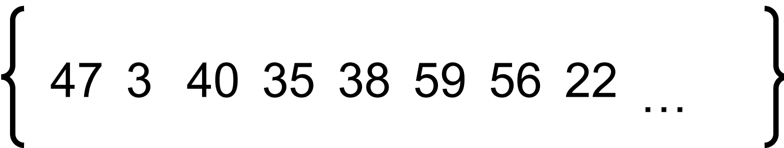
- Antar at input er jevnt fordelt over et intervall
- Idé:
  - Vi deler opp intervallet i $k = \Theta(n)$ like store bøtter, for eksempel $k = n$.
  - Elementene som skal sorteres puttes i sine respektive bøtter, de minste elementene i den minste bøtta osv.
  - Hver bøtte sorteres med insertion sort.
  - Bøttene settes sammen til det ferdige resultatet.

Kjøretid

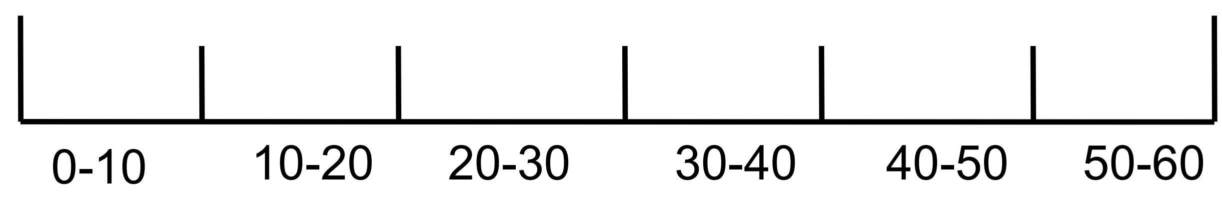


- Innsetting i bøtter tar O(n) worst case.
- Sorteringen i hver bøtte er avhengig av hvor mange elementer som finnes i den enkelte bøtte.





60 tall fra 0 til 60



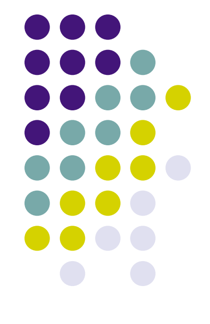
Forventet bøttestr.
10 elementer

Genererte bøtter hvor forventet
innhold er et konstant antall elementer

Egenskaper:

	Counting sort	Radix sort	Bucket sort
Stabil	✓	✓	✓
Parallelliserbar			
In-place			

Selection

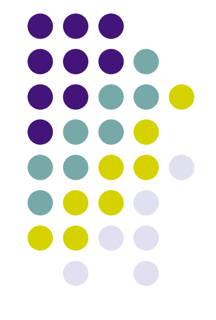


- Finne min, max eller median.
- En måte er å sortere, men det kan gjøres raskere.

Randomized Select

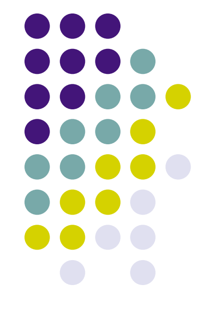
```
RandomizedSelect(A, p, r, i)
    if (p == r) then return A[p];
    q = RandomizedPartition(A, p, r)
    k = q - p + 1;
    if (i == k) then return A[q];   // not in book
    if (i < k) then
        return RandomizedSelect(A, p, q-1, i);
    else
        return RandomizedSelect(A, q+1, r, i-k);
```

Select



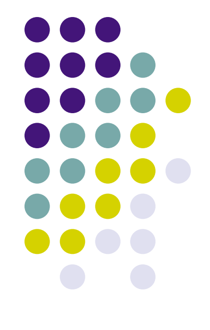
- Del i grupper på 5, og finn medianen i hver
- Finn «medianen til medianene», rekursivt
- Bruk denne som pivot i Partition
- Select rekursivt på «halvparten»

Ressurser



- http://www.sorting-algorithms.com/
- Wikipedia er som regel veldig bra på algoritmer
- Som alltid: algdat@idi.ntnu.no
- Stud.asser

Forrige øving



- Burde deles i to
  - Først finne den subgrafen man ikke kan nå.
  - Så telle noder og kanter.