Algoritmer og datastrukturer

Økt 3 – Permutasjoner, søking og sortering

Repetisjon

- Åpne og lukkede intervaller
- Exceptions – feilhåndtering
- Algoritmeanalyse – best case, average case, worst case
- Turneringstrær og nest største tall
- Notasjon for binære trær
  - Bladnode, indre node
  - Forelder, høyre barn, venstre barn, søsken
  - Høyde
  - Fullt tre, komplett tre, perfekt tre
- Implementasjon av binære trær i tabell

Trekke tilfeldige tall

- Med tilbakelegging:
  - Eksempel: 1, 3, 5, 3, 3, 2, 7
  - Samme tall kan trekkes gjentatte ganger

- Uten tilbakelegging:
  - Eksempel: 1, 3, 5, 4, 6, 2, 7
  - Samme tall kan kun forekomme én gang

Permutasjoner – trekking uten tilbakelegging

- Trekke tre tilfeldige tall [1, 3]:
  - Første tall: kan velge mellom {1, 2, 3}, trekker 2
  - Andre tall: kan velge mellom {1, 3}, trekker 3
  - Tredje tall: kan velge mellom {1}, trekker 1
- Hvor mange muligheter har vi totalt sett?
  - 3*2*1 = 6
- For fire tall har vi 4*3*2*1 = 24
- For fem tall har vi 5*4*3*2*1 = 120

Permutasjoner

- Fakultet
  n! = n*(n-1)*(n-2)*…*1

```
 2!  =  2
 3!  =  6
 4!  =  24
 5!  =  120
 6!  =  720
10!  =  3.628.800
20!  =  2.432.902.008.176.640.000
```

Inversjon

- «usortert» tallpar
- 1, 3, 5, 4, 6, 2, 7
- Har vi null inversjoner, så er rekken sortert!
- For en rekke med n tall har vi mellom null (sortert) og n-1 (invers sortert) inversjoner

Bubble sort

- Gå gjennom hver posisjon i tabellen og «boble» oppover
- For hvert tallpar, bytt så største kommer til høyre
- n*(n-1)/4 operasjoner (antall inversjoner!)

4: [4, 2, 3, 1]
4: [2, 4, 3, 1]
4: [2, 3, 4, 1]
4: [2, 3, 1, 4]

3: [2, 3, 1, 4]
3: [2, 3, 1, 4]
3: [2, 1, 3, 4]

2: [2, 1, 3, 4]
2: [1, 2, 3, 4]

```
public static void boblesortering(int[] a)      // hører til klassen Tabell
{
  for (int n = a.length; n > 1; n--)            // n reduseres med 1 hver gang
  {
    for (int i = 1; i < n; i++)                 // går fra 1 til n
    {
      if (a[i - 1] > a[i]) bytt(a, i - 1, i);   // sammenligner/bytter
    }
  }
}
```

Programkode 1.3.3 e)

Utvalgssortering – selection sort

- Finn minste tall og bytt med posisjon 1
- Repeter med tabell størrelse n-1
- n*(n-1)/2 operasjoner

```
public static void utvalgssortering(int[] a)
{
  for (int i = 0; i < a.length - 1; i++)
    bytt(a, i, min(a, i, a.length));  // to hjelpemetoder
}
```

[6, 7, 1, 4, 8, 9, 2, 5, 3, 10]
[1, 7, 6, 4, 8, 9, 2, 5, 3, 10]
[1, 2, 6, 4, 8, 9, 7, 5, 3, 10]
[1, 2, 3, 4, 8, 9, 7, 5, 6, 10]
[1, 2, 3, 4, 8, 9, 7, 5, 6, 10]
[1, 2, 3, 4, 5, 9, 7, 8, 6, 10]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Søke i usortert tabell

- Usortert søk: sammenlikn med alle verdier
- n sammenlikninger

```
public static int usortertsøk(int[] a, int verdi)  // tabell og søkeverdi
{
  for (int i = 0; i < a.length; i++)  // går gjennom tabellen
    if (verdi == a[i]) return i;      // verdi funnet - har indeks i

  return -1;                          // verdi ikke funnet
}
```

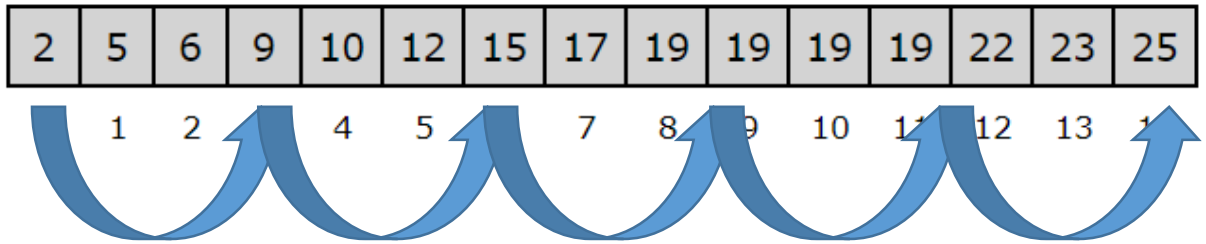
Lineær søk med sentinel

```java
public static int lineærsøk(int[] a, int verdi) // legges i class Tabell
{
  if (a.length == 0 || verdi > a[a.length-1])
    return -(a.length + 1);  // verdi er større enn den største

  int i = 0; for( ; a[i] < verdi; i++);  // siste verdi er vaktpost

  return verdi == a[i] ? i : -(i + 1);   // sjekker innholdet i a[i]
}
```

Kvadratrotsøk

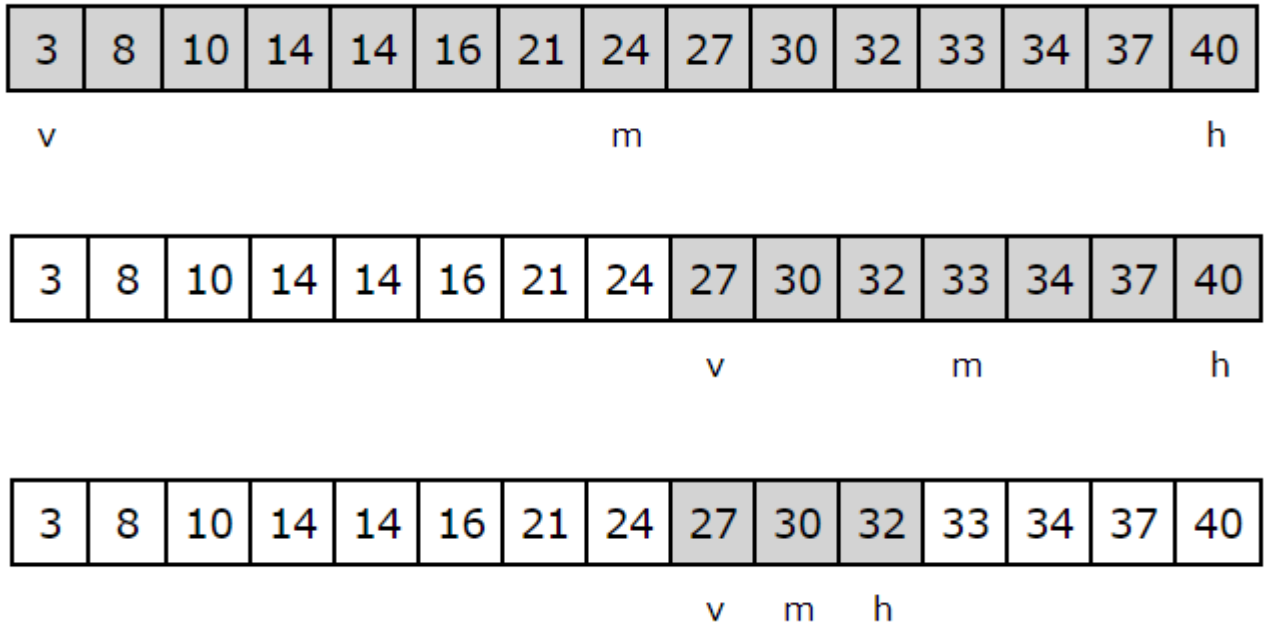
- Som lineært søk (usortert søk), men øk i med kvadratroten av tabellens lengde istedenfor i

- for (int i=0; i<a.length; ++i)
- for (int i=0; i<a.length; i+=sqrt(a.length)) {
  if (a[i] > verdi) { … }



Binærsøk

- Søk etter 30:
  - Er a[m] lik 20?
    Søket er ferdig!
  - Er 30 større enn midt
    Søk i intervallet [m+1, h]
  - Ellers
    Søk i intervallet [l, m]



Binærsøk

```
public static int binærsøk(int[] a, int fra, int til, int verdi)
{
  Tabell.fratilKontroll(a.length,fra,til);  // se Programkode 1.2.3 a)
  int v = fra, h = til - 1;  // v og h er intervallets endepunkter

  while (v <= h)    // fortsetter så lenge som a[v:h] ikke er tom
  {
    int m = (v + h)/2;       // heltallsdivisjon - finner midten
    int midtverdi = a[m];    // hjelpevariabel for midtverdien

    if (verdi == midtverdi) return m;          // funnet
    else if (verdi > midtverdi) v = m + 1;     // verdi i a[m+1:h]
    else  h = m - 1;                           // verdi i a[v:m-1]
  }

  return -(v + 1);    // ikke funnet, v er relativt innsettingspunkt
}
```

Binærsøk

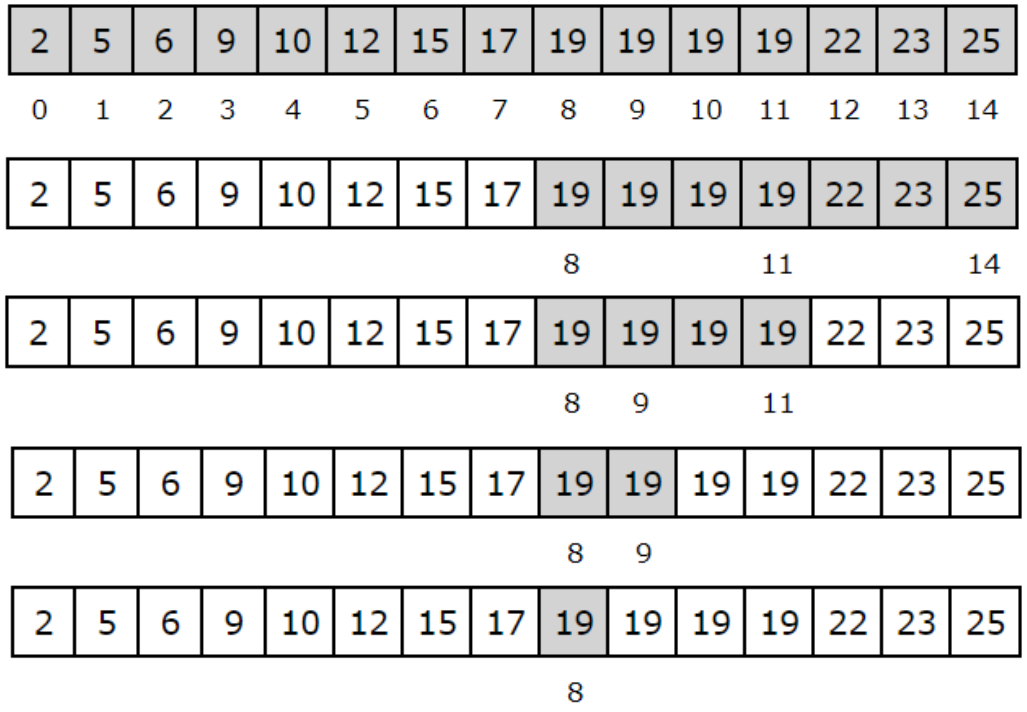
```java
// 2. versjon av binærsøk - returverdier som for Programkode 1.3.6 a)
public static int binærsøk(int[] a, int fra, int til, int verdi)
{
  Tabell.fratilKontroll(a.length,fra,til);  // se Programkode 1.2.3 a)
  int v = fra, h = til - 1;    // v og h er intervallets endepunkter

  while (v <= h)  // fortsetter så lenge som a[v:h] ikke er tom
  {
    int m = (v + h)/2;      // heltallsdivisjon - finner midten
    int midtverdi = a[m];  // hjelpevariabel for  midtverdien

    if (verdi > midtverdi) v = m + 1;         // verdi i a[m+1:h]
    else if (verdi < midtverdi) h = m - 1;    // verdi i a[v:m-1]
    else return m;                            // funnet
  }

  return -(v + 1);   // ikke funnet, v er relativt innsettingspunkt
}
```

Binærsøk

```java
// 3. versjon av binærsøk - returverdier som for Programkode 1.3.6 a)
public static int binærsøk(int[] a, int fra, int til, int verdi)
{
  Tabell.fratilKontroll(a.length,fra,til);  // se Programkode 1.2.3 a)
  int v = fra, h = til - 1;  // v og h er intervallets endepunkter

  while (v < h)  // obs. må ha v < h her og ikke v <= h
  {
    int m = (v + h)/2;  // heltallsdivisjon - finner midten

    if (verdi > a[m]) v = m + 1;   // verdi må ligge i a[m+1:h]
    else  h = m;                   // verdi må ligge i a[v:m]
  }
  if (h < v || verdi < a[v]) return -(v + 1);  // ikke funnet
  else if (verdi == a[v]) return v;            // funnet
  else  return -(v + 2);                       // ikke funnet
}
```

Binærsøk – duplikate verdier

```
// 3. versjon av binærsøk - returverdier som for Programkode 1.3.6 a)
public static int binærsøk(int[] a, int fra, int til, int verdi)
{
  Tabell.fratilKontroll(a.length,fra,til);  // se Programkode 1.2.3 a)
  int v = fra, h = til - 1;  // v og h er intervallets endepunkter

  while (v < h)  // obs. må ha v < h her og ikke v <= h
  {
    int m = (v + h)/2;  // heltallsdivisjon - finner midten

    if (verdi > a[m]) v = m + 1;   // verdi må ligge i a[m+1:h]
    else  h = m;                   // verdi må ligge i a[v:m]
  }
  if (h < v || verdi < a[v]) return -(v + 1);  // ikke funnet
  else if (verdi == a[v]) return v;            // funnet
  else  return -(v + 2);                       // ikke funnet
}
```



Søking og effektivitet

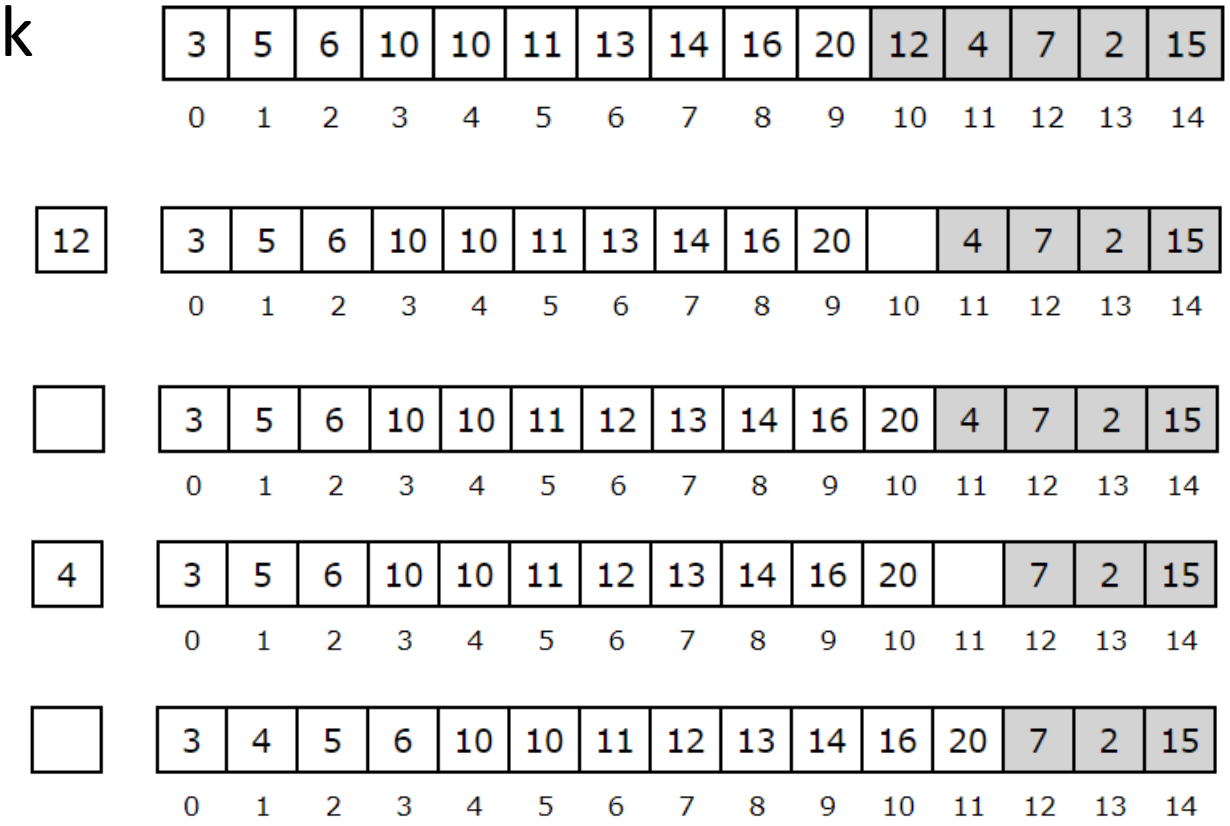
Søkealgoritme	Det gjennomsnittlige antallet sammenligninger					
Navn	Formel	n = 10	n = 100	n = 10.000	n = 1.000.000	Orden
Lineærsøk	$(n + 1)/2 + 2$	7,5	52,5	≈ 5 000	≈ 500 000	n
Kvadratrotsøk	$\sqrt{n} + 2$	5,2	12	102	1002	$\sqrt{n}$
Binærsøk, 1. versjon	$2 \cdot \log_2(n+1) - 3$	4,8	10,3	23,6	37	$\log_2(n)$
Binærsøk, 2. versjon	$1{,}5 \cdot \log_2(n+1) - 1$	4,8	9,0	18,9	29	$\log_2(n)$
Binærsøk, 3. versjon	$\log_2(n) + 1$	4,4	7,7	14,3	21	$\log_2(n)$

Ordnet innsetting i tabell

```
int[] a = {3,5,6,10,10,11,13,14,16,20,0,0,0,0,0};  // en tabell
int antall = 10;                                   // antall verdier

if (antall >= a.length) throw new IllegalStateException("Tabellen er full");

int nyverdi = 10;                                  // ny verdi
int k = Tabell.binærsøk(a, 0, antall, nyverdi);    // søker i a[0:antall>
if (k < 0) k = -(k + 1);                           // innsettingspunkt

for (int i = antall; i > k; i--) a[i] = a[i-1];    // forskyver

a[k] = nyverdi;                                    // legger inn
antall++;                                          // øker antallet

Tabell.skrivln(a, 0, antall);  // Se Oppgave 4 og 5 i Avsnitt 1.2.2
```

Insertion sort (innsettingssortering)

- Ta en verdi ut av tabellen på plass k
  Krav: alt før plass k er sortert
- Bruk ordnet innsetting i intervallet [0, k-1]
- $n(n + 3)/4 - H_n$ operasjoner i gjennomsnitt



Insertion sort

```
public static void innsettingssortering(int[] a)
{
  for (int i = 1; i < a.length; i++)  // starter med i = 1
  {
    int verdi = a[i], j = i - 1;      // verdi er et tabellelemnet, j er en indeks
    for (; j >= 0 && verdi < a[j]; j--) a[j+1] = a[j];  // sammenligner og flytter
    a[j + 1] = verdi;                 // j + 1 er rett sortert plass
  }
}
```

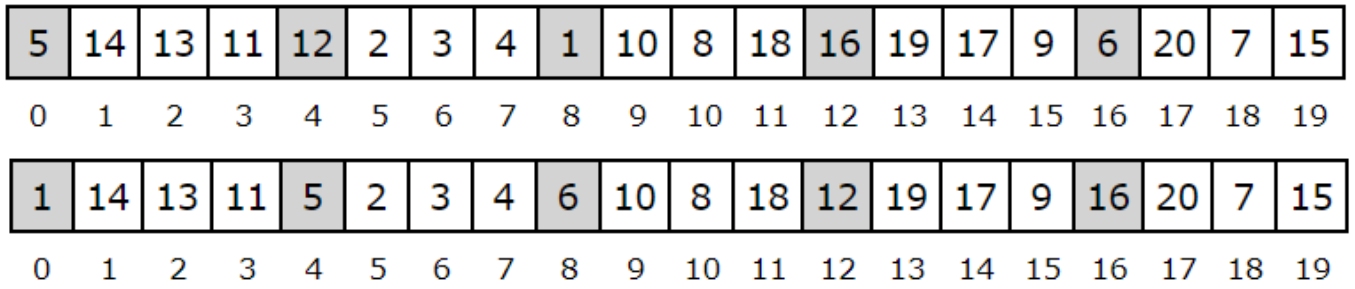
Programkode 1.3.8 c)

Algoritmeanalyse

Sorteringsalgoritme	Antall sammenligninger		
Navn	Gjennomsnittlig	Verste tilfelle	Beste tilfelle
Boblesortering	$n(n-1)/2$	$n(n-1)/2$	$n-1$
Utvalgssortering	$n(n-1)/2$	$n(n-1)/2$	$n(n-1)/2$
Innsettingssortering	$n(n+3)/4 - H_n$	$n(n-1)/2$	$n-1$

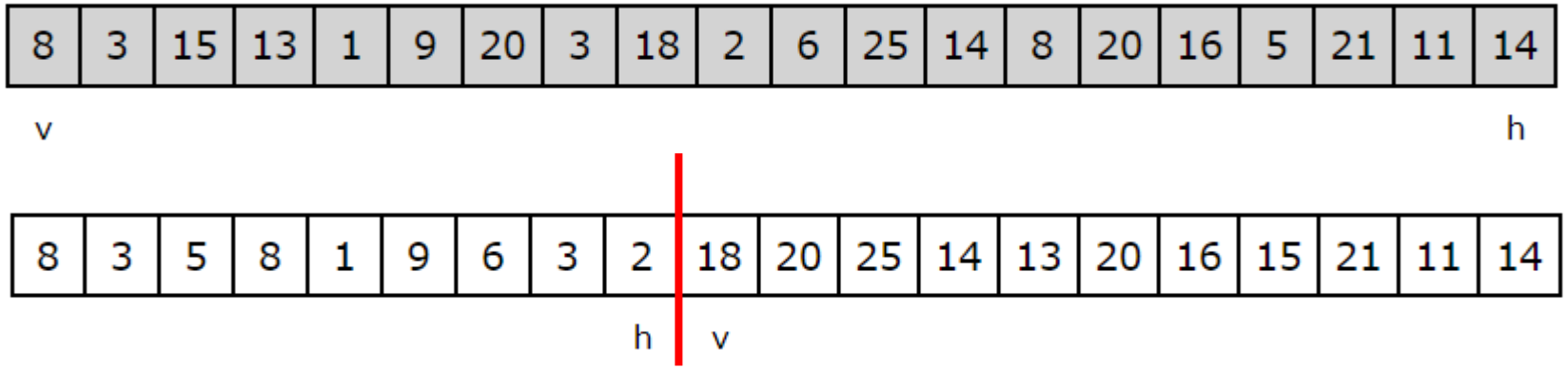
Shell sort (Shellsortering)

- Start med en gap length, f.eks fire.
- Lag en subliste av alle tallene på hver fjerde plass,
  [5, 12, 1, 16, 6]
- Sorter listen (selection sort)
  [1, 5, 6, 12, 16]
- Repeter for startposisjon
  0, 1, 2, 3
- Gjenta hele prosessen med gap length 2.
- I siste iterasjon sorteres hele listen med selection sort



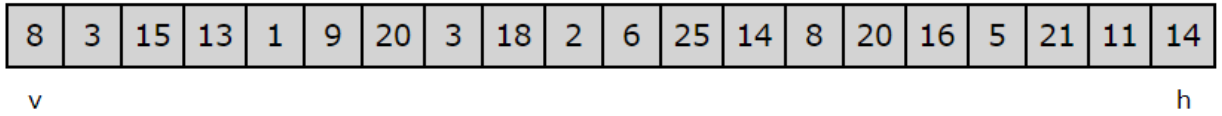
Quick sort (kvikksortering)

- Bygger på konseptet partisjonering med en «pivot»
- Sorter tabellen slik at alle tall mindre enn pivot ligger til venstre og alle tall større ligger til høyre
- Eksempel: Pivot = 10

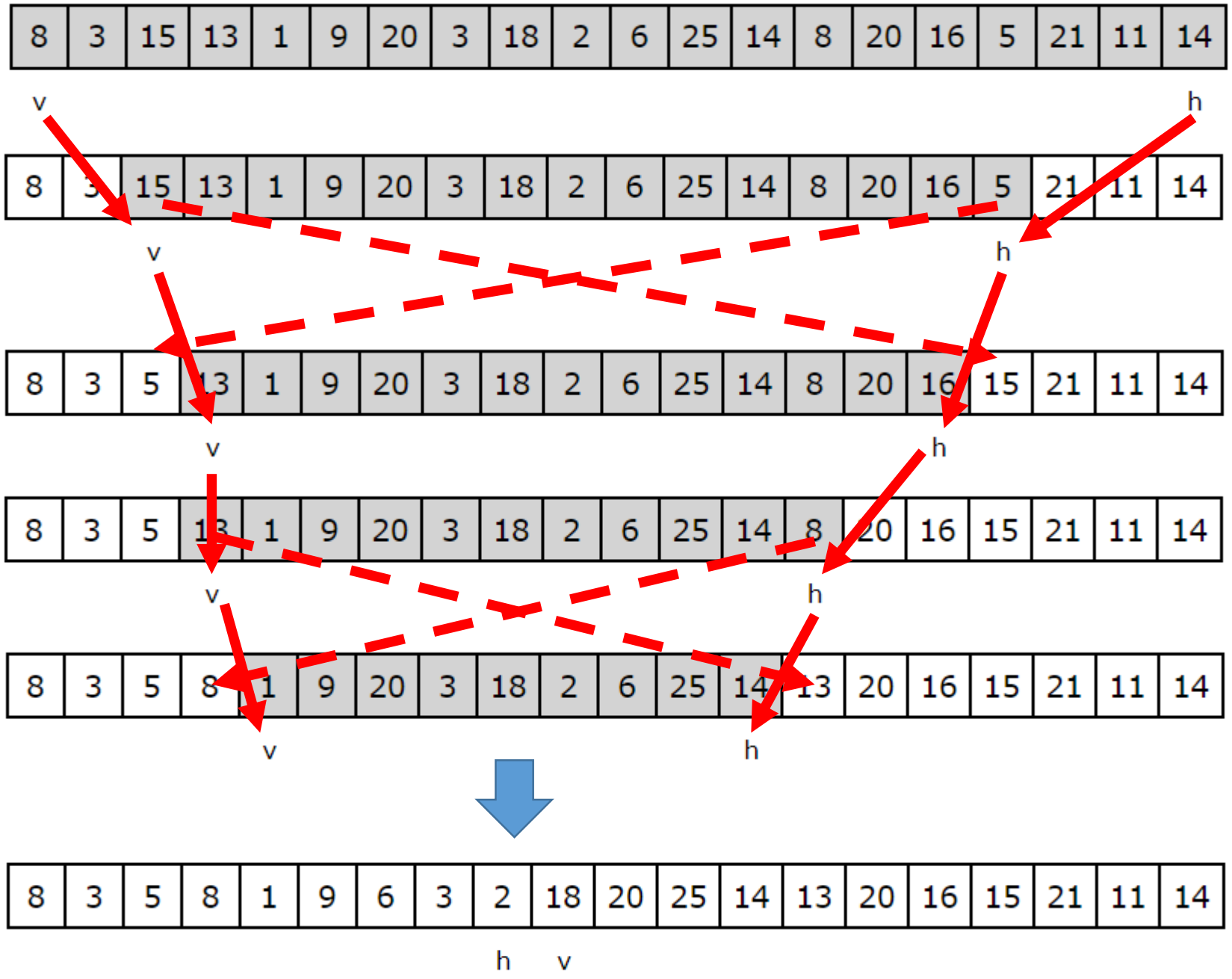


Partisjonering med pivot

- Bruk en venstre-peker, høyre-peker, og pivot=10



- Flytt venstre-peker mot midten så lenge tallet er mindre enn pivot.
- Flytt høyre-peker mot midten så lenge tallet er større enn pivot
- Bytt plass når venstre er større enn pivot og høyre er mindre enn pivot
- Repeter


8 3 15 13 1 9 20 3 18 2 6 25 14 8 20 16 5 21 11 14
v
h
8 3 15 13 1 9 20 3 18 2 6 25 14 8 20 16 5 21 11 14
v
h
8 3 5 13 1 9 20 3 18 2 6 25 14 8 20 16 15 21 11 14
v
h
8 3 5 13 1 9 20 3 18 2 6 25 14 8 20 16 15 21 11 14
v
h
8 3 5 8 1 9 20 3 18 2 6 25 14 13 20 16 15 21 11 14
v
h
8 3 5 8 1 9 6 3 2 18 20 25 14 13 20 16 15 21 11 14
h
v

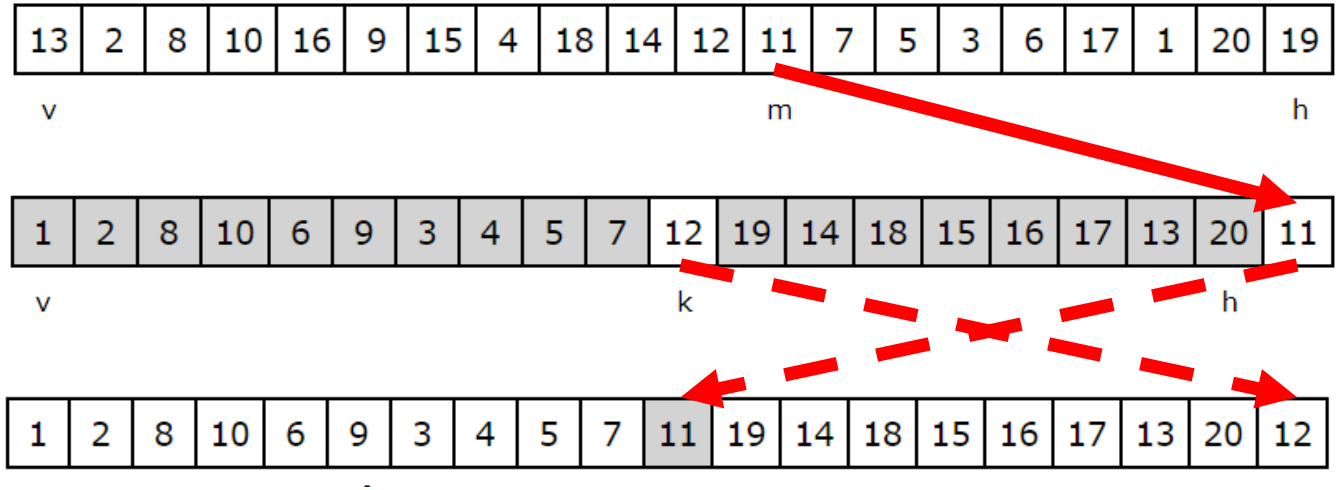
Partisjonering med pivot

```
private static int parter0(int[] a, int v, int h, int skilleverdi)
{
  while (true)                                    // stopper når v > h
  {
    while (v <= h && a[v] < skilleverdi) v++;     // h er stoppverdi for v
    while (v <= h && a[h] >= skilleverdi) h--;    // v er stoppverdi for h

    if (v < h) bytt(a,v++,h--);                   // bytter om a[v] og a[h]
    else  return v;  // a[v] er nåden første som ikke er mindre enn skilleverdi
  }
}
```

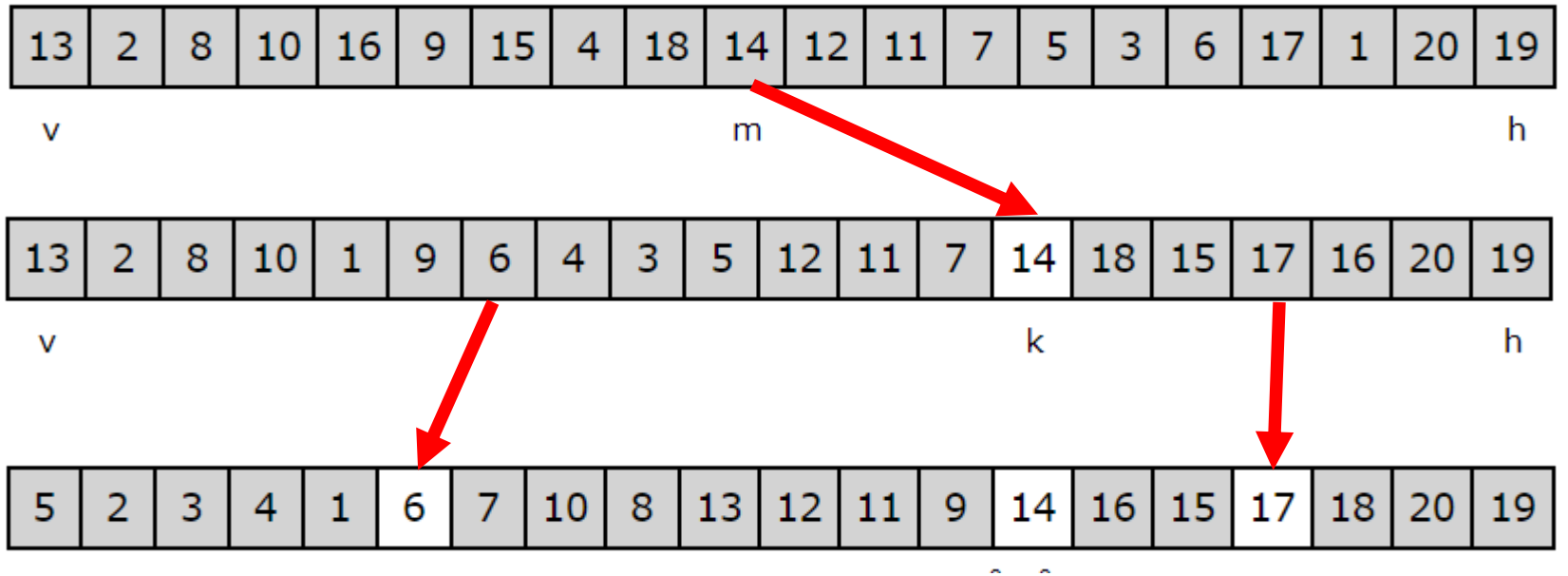
Programkode 1.3.9 a)

Sortere ett tall basert på pivot

- Velg en pivot-verdi, f.eks a[11] = 11
- Flytt a[11] til slutten av tabellen
- Partisjoner resten av tabellen med pivot 11
- Alle tall mindre enn 11 er nå i posisjon [0, k)
- Alle tall større enn eller lik 11 er nå i posisjon [k, n)
- Bytt a[k] (pivot-posisjon) med a[n-1] (pivot verdi)
- Tallet 11 er nå på riktig plass!



Quicksort – rekursiv pivotering



Quicksort – rekursiv pivotering

```
private static void kvikksortering0(int[] a, int v, int h)  // en privat metode
{
  if (v >= h) return;  // a[v:h] er tomt eller har maks ett element
  int k = sParter0(a, v, h, (v + h)/2);  // bruker midtverdien
  kvikksortering0(a, v, k - 1);     // sorterer intervallet a[v:k-1]
  kvikksortering0(a, k + 1, h);     // sorterer intervallet a[k+1:h]
}

public static void kvikksortering(int[] a, int fra, int til) // a[fra:til>
{
  fratilKontroll(a.length, fra, til);  // sjekker når metoden er offentlig
  kvikksortering0(a, fra, til - 1);  // v = fra, h = til - 1
}

public static void kvikksortering(int[] a)   // sorterer hele tabellen
{
  kvikksortering0(a, 0, a.length - 1);
}
```

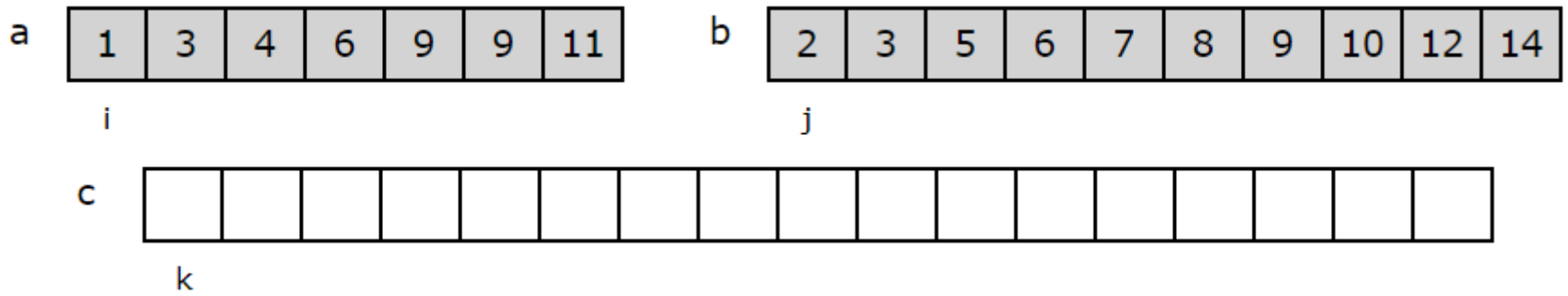
Quick search

- Blanding av binærsøk og quicksort
- Søk etter tallet m
- Partisjoner tabellen slik at alle til venstre er mindre enn m, og alle større er til høyre.
- Hvis tallet på pivot-posisjonen er større enn søkeverdien, flytt høyre til midtverdien-1
- Hvis tallet på pivot-posisjonen er mindre enn søkeverdien, flytt venstre til midtverdien+1
- Hvis tallet på pivot-posisjonen er lik søkeverdien er vi ferdige!

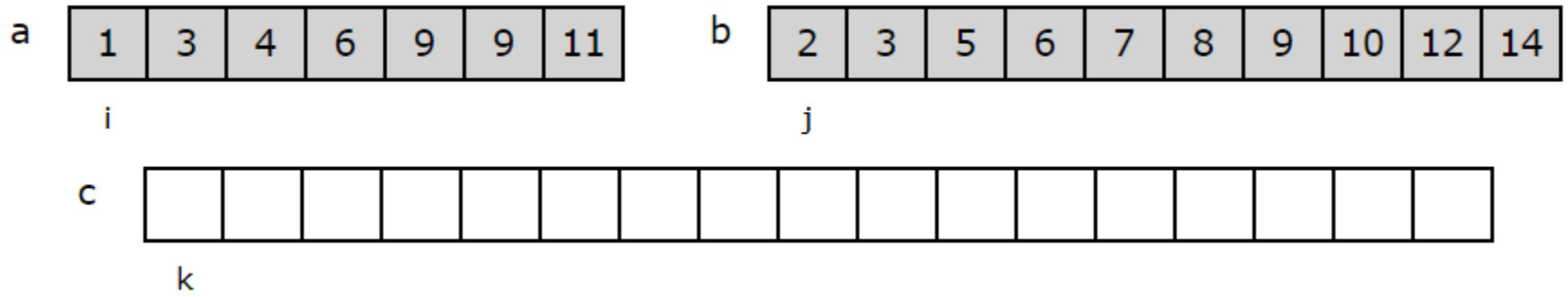
Merge sort

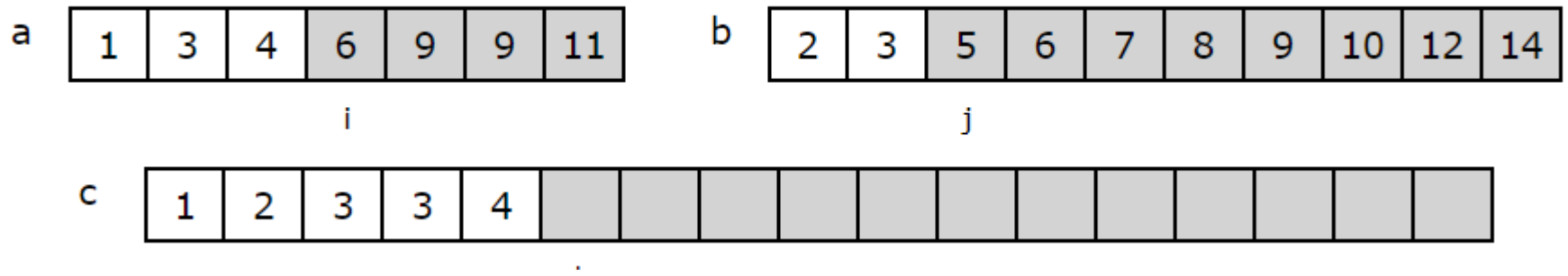
- Enkel idé:

- Gitt to lister med sorterte tall, velg det minste fra de to listene til enhver tid



Merge sort


a
1 3 4 6 9 9 11
b
2 3 5 6 7 8 9 10 12 14
i
j
c
k

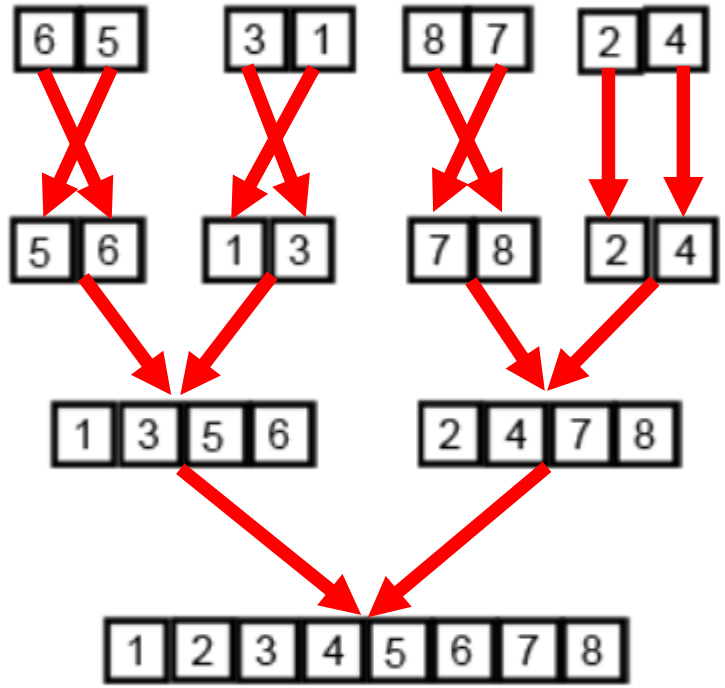

a
1 3 4 6 9 9 11
b
2 3 5 6 7 8 9 10 12 14
i
j
c
1 2 3 3 4

Merge sort

```java
public static int flett(int[] a, int m, int[] b, int n, int[] c)
{
  int i = 0, j = 0, k = 0;
  while (i < m && j < n) c[k++] = a[i] <= b[j] ? a[i++] : b[j++];

  while (i < m) c[k++] = a[i++];   // tar med resten av a
  while (j < n) c[k++] = b[j++];   // tar med resten av b

  return k;   // antallet verdier som er lagt inn i c
}
```

Merge sort

- Del opp tabellen i tall-par
- Sorter tallparene hver for seg
- Start å flette tabellene sammen



Animasjon fra Wikipedia - Mergesort