Algoritmer og datastrukturer

Økt 4 – Rekursjon, flerdimensjonelle tabeller

Repetisjon

- Trekke tilfeldige tall – med og uten tilbakelegging
- Permutasjoner: 52! forskjellige permutasjoner i en kortstokk
- Inversjoner – ombyttede tallpar i en sortert rekke
- Bubble sort – store tall «bobler» opp
- Selection sort – Sorter minste tall, gjenta på resterende tallrekke
- Søking i usortert og sortert tabell
- Binærsøk
- Partisjonering i quicksort

Rekursive metoder

```java
public static int a(int n)          // n må være et ikke-negativt tall
{
  if (n == 0) return 1;             // a0 = 1
  else if (n == 1) return 2;        // a1 = 2
  else return 2*a(n-1) + 3*a(n-2);  // to rekursive kall
}
```

```java
public static int tverrsum(int n)              // n må være >= 0
{
  if (n < 10) return n;                        // kun ett siffer
  else return tverrsum(n / 10) + (n % 10);     // metoden kalles
}
```

Rekursjon

```java
public static int tverrsum(int n)
{
  System.out.println("tverrsum(" + n + ") starter!");
  int sum = (n < 10) ? n : tverrsum(n / 10) + (n % 10);
  System.out.println("tverrsum(" + n + ") er ferdig!");
  return sum;
}            Programkode 1.5.2 c)
```

```java
public static void main(String... args)
{
  System.out.println("main() starter!");
  int sum = tverrsum(7295);
  System.out.println("main() er ferdig!");
}            Programkode 1.5.2 d)
```

```
main() starter!
tverrsum(7295) starter!
tverrsum(729) starter!
tverrsum(72) starter!
tverrsum(7) starter!
tverrsum(7) er ferdig!
tverrsum(72) er ferdig!
tverrsum(729) er ferdig!
tverrsum(7295) er ferdig!
main() er ferdig!
```

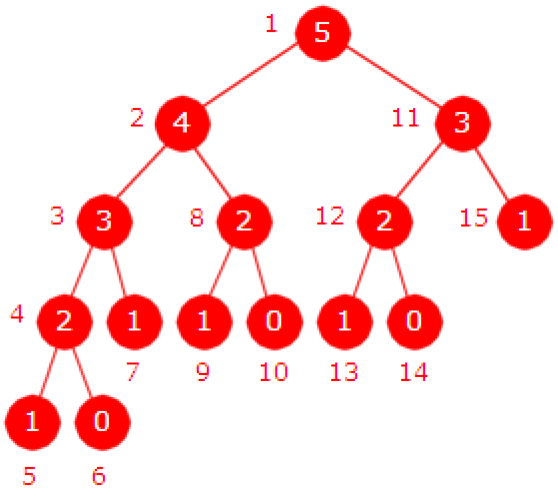
Stack overflow

```
level(2641, 2651)
level(2642, 2651)
level(2643, 2651)
level(2644, 2651)
level(2645, 2651)
level(2646, 2651)
level(2647, 2651)
level(2648, 2651)
level(2649, 2651)
level(2650, 2651)
Exception in thread "main" java.lang.StackOverflowError
    at sun.nio.cs.UTF_8.updatePositions(UTF_8.java:77)
    at sun.nio.cs.UTF_8.access$200(UTF_8.java:57)
    at sun.nio.cs.UTF_8$Encoder.encodeArrayLoop(UTF_8.java:636)
    at sun.nio.cs.UTF_8$Encoder.encodeLoop(UTF_8.java:691)
    at java.nio.charset.CharsetEncoder.encode(CharsetEncoder.java:579)
    at sun.nio.cs.StreamEncoder.implWrite(StreamEncoder.java:271)
    at sun.nio.cs.StreamEncoder.write(StreamEncoder.java:125)
    at java.io.OutputStreamWriter.write(OutputStreamWriter.java:207)
    at java.io.BufferedWriter.flushBuffer(BufferedWriter.java:129)
```

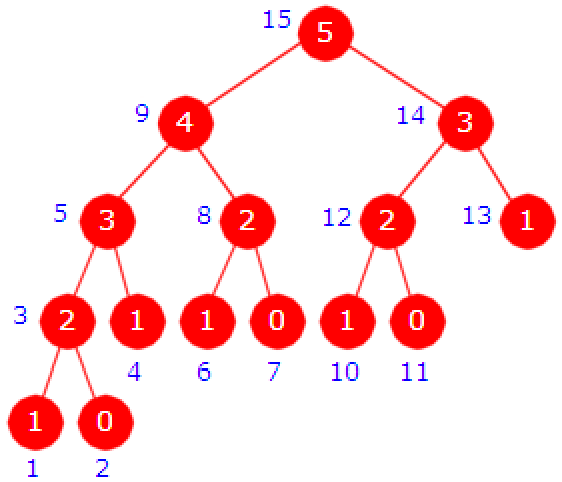
Rekursjon

```java
public static int fib(int n)          // med utskriftssetninger
{
  System.out.println("fib(" + n + ") starter!");
  int fib = n > 1 ? fib(n-1) + fib(n-2) : n;
  System.out.println("fib(" + n + ") er ferdig!");
  return fib;       // metoden er ferdig
}
```

Starter



Ferdig



```
1  fib(5) starter!
2  fib(4) starter!
3  fib(3) starter!
4  fib(2) starter!
5  fib(1) starter!
1  fib(1) er ferdig
6  fib(0) starter!
2  fib(0) er ferdig
3  fib(2) er ferdig
7  fib(1) starter!
4  fib(1) er ferdig
5  fib(3) er ferdig!
8  fib(2) starter!
9  fib(1) starter!
6  fib(1) er ferdig!
10 fib(0) starter!
7  fib(0) er ferdig!
8  fib(2) er ferdig!
9  fib(4) er ferdig!
11 fib(3) starter!
12 fib(2) starter!
13 fib(1) starter!
10 fib(1) er ferdig!
14 fib(0) starter!
11 fib(0) er ferdig!
12 fib(2) er ferdig!
15 fib(1) starter!
13 fib(1) er ferdig!
14 fib(3) er ferdig!
15 fib(5) er ferdig!
```

Rekursjon – krav til terminering

Krav 1. Når metoden kalles seg selv én eller flere ganger må kallet (eller kallene) utføres på et tilfelle (eller en situasjon) som er enklere enn det tilfellet (den situasjonen) vi opprinnelig hadde. I tilegg må kallene være slik at ting ikke gjentas, dvs. at noe som allerede er løst ikke løses på nytt.

Krav 2. Metodekallene må utformes slik at det før eller senere oppstår et tilfelle (eller en situasjon) som kan behandles uten et nytt eller nye kall på metoden. Dette kalles et *basistilfelle* (eller en *basissituasjon*).

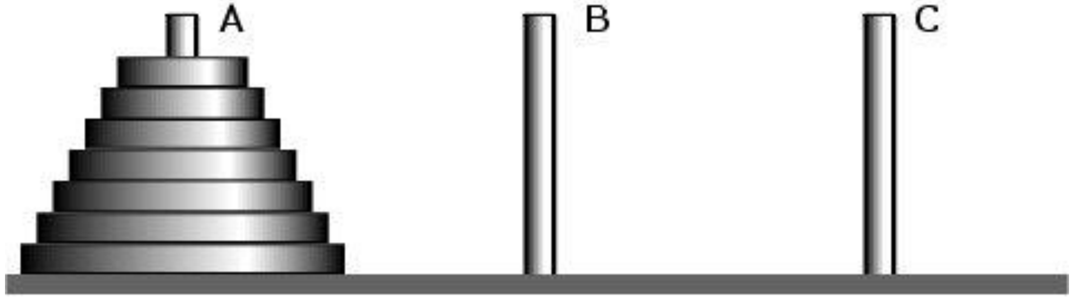
Quicksort

```
private static void kvikksortering0(int[] a, int v, int h)
{
  if (v >= h) return;   // tomt eller maks ett element

  int k = sParter0(a,v,h,(v + h)/2);   // se Programkode 1.3.9 f)
  kvikksortering0(a,v,k-1);
  kvikksortering0(a,k+1,h);
}
```

Merge sort

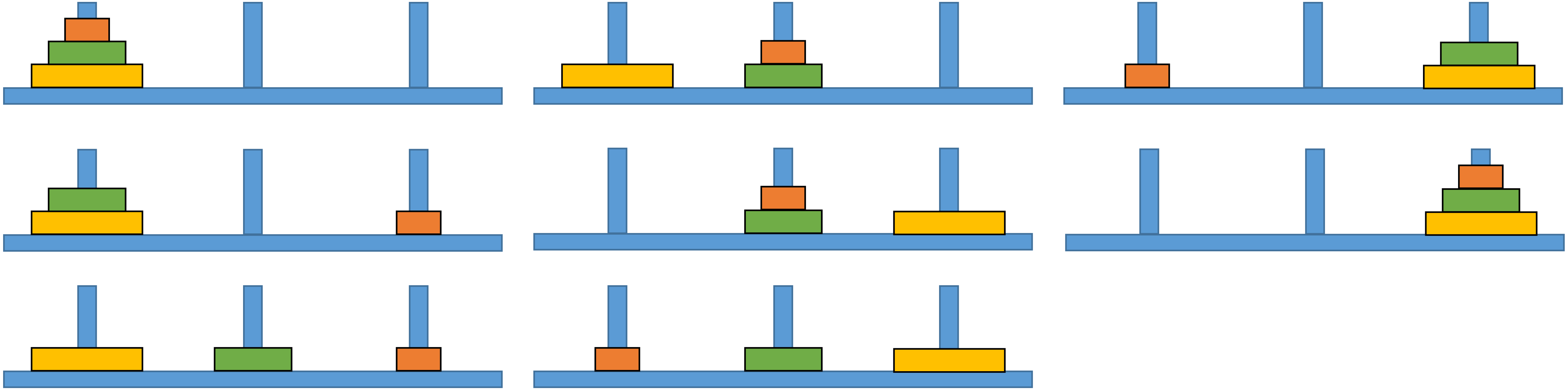
```java
private static void flettesortering(int[] a, int[] b, int fra, int til)
{
  if (til - fra <= 1) return;   // a[fra:til> har maks ett element

  int m = (fra + til)/2;        // midt mellom fra og til

  flettesortering(a,b,fra,m);   // sorterer a[fra:m>
  flettesortering(a,b,m,til);   // sorterer a[m:til>

  // fletter a[fra:m> og a[m:til>
  flett(a,b,fra,m,til);         // Programkode 1.3.11 f)
}

public static void flettesortering(int[] a)
{
  int[] b = new int[a.length/2];    // en hjelpetabell for flettingen
  flettesortering(a,b,0,a.length);  // kaller metoden over
}
```

Towers of Hanoi – Hanois tårn



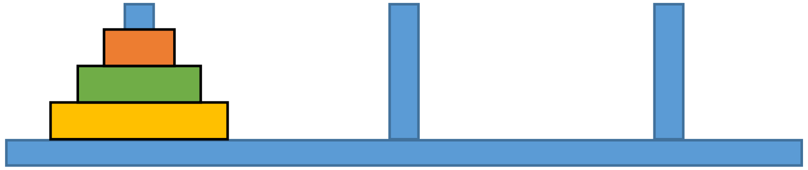
- Flytt brikkene fra pinne *A* til pinne *C* slik at det også blir et kjegleformet tårn på *C*. Under flyttingen er det ikke lov å legge en brikke oppå en som er mindre.

Algoritme



Hanois tårn

```java
public static void HanoisTårn(char> A, char B, char C, int n)
{
  if (n == 0) return;   // ingen brikker - tomt tårn
  HanoisTårn(A, C, B, n - 1);
  System.out.println("Brikke " + n + " fra " + A + " til " + C);
  HanoisTårn(B, A, C, n - 1);
}
```



```java
HanoisTårn('A','B','C',3);

//  Brikke 1 fra A til C
//  Brikke 2 fra A til B
//  Brikke 1 fra C til B
//  Brikke 3 fra A til C
//  Brikke 1 fra B til A
//  Brikke 2 fra B til C
//  Brikke 1 fra A til C
```

Flerdimensjonelle tabeller

- To dimensjoner

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15

```java
int[][] a = {{1,2,3,4,5}, {6,7,8,9,10}, {11,12,13,14,15}};
```

- a[j][i] gir oss rad j, kolonne i
- Eksempel: a[3][2] = 12

Flerdimensjonelle tabeller

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15

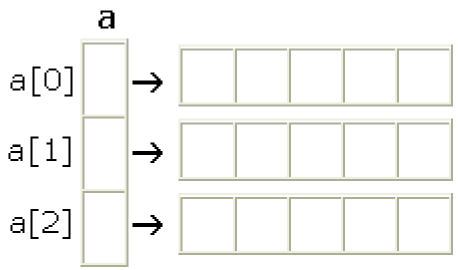
```
int[] rad1 = {1,2,3,4,5};
int[] rad2 = {6,7,8,9,10};
int[] rad3 = {11,12,13,14,15};

int[][] a = {rad1, rad2, rad3};
```

```
int[][] a = new int[3][5];        // 3 rader, 5 kolonner

 for (int i = 0; i < 3; i++)        // 3 rader
 {
   for (int j = 0; j < 5; j++)      // 5 kolonner
   {
     a[i][j] = 5*i + j + 1;
   }
 }
```

Flerdimensjonelle tabeller

```java
int[][] a = new int[3][5];
```

```java
int[][] a = new int[3][];    // lager den vertikale tabellen

a[0] = new int[5];           // lager a[0]-tabellen
a[1] = new int[5];           // lager a[1]-tabellen
a[2] = new int[5];           // lager a[2]-tabellen
```



Repetisjon av algoritmeanalyse

Funksjonstype	**n = 1**	**n = 10**	**n = 100**	**n = 1000**	**Beskrivelse**
$f_1(n) = 1$	1	1	1	1	*konstant*
$f_2(n) = \log_2 n$	0	3,3	6,6	9,97	*logaritmisk*
$f_3(n) = \sqrt{n}$	1	3,2	10	31,6	*kvadratrot*
$f_4(n) = n$	1	10	100	1000	*lineær*
$f_5(n) = n \log_2 n$	0	33,2	664,4	9.965,8	*lineæritmisk*
$f_6(n) = n^2$	1	100	10.000	1.000.000	*kvadratisk*
$f_7(n) = n^3$	1	1.000	1.000.000	10 siffer	*kubisk*
$f_8(n) = 2^n$	2	1.024	31 siffer	302 siffer	*eksponensiell*
$f_9(n) = n!$	1	3.628.800	158 siffer	2568 siffer	*faktoriell*