ERHVERVSAKADEMI ÅRHUS

Programmering 2

Del-løs og kombiner

Dagens emner

- Opgaver fra sidst

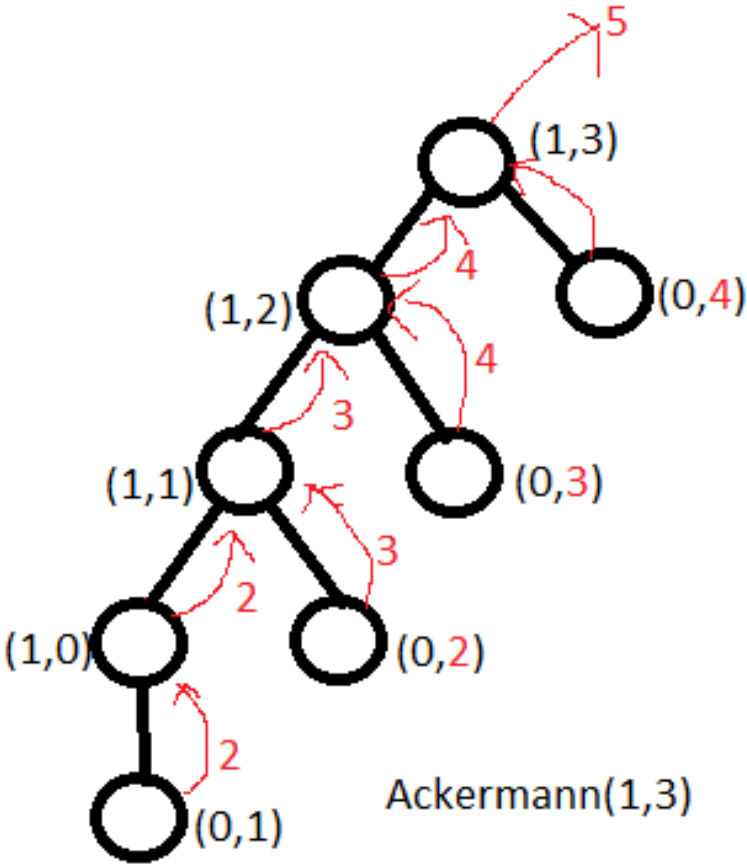
  Simpel rekursion

  - Opgave 1: Lige tal i liste (Sidsels gruppe)
  - Opgave 2: Palindrom (Jespers gruppe)
  - Opgave 3: Binær søgning (Magnus gruppe)
  - Opgave 4: Ackermann (Mikkels gruppe)
  - Opgave 6: Towers of Hanoi (Mikes gruppe)
- Del løs og kombiner
  - Skabelonen
  - Simple implementeringer
  - Flettesortering
  - Quicksortering

Opgave 4: Ackermanm

```
A(x,y) = y+1                  når x = 0
         A(x-1,1)             når y = 0
         A(x-1, A(x,y-1))     ellers
```

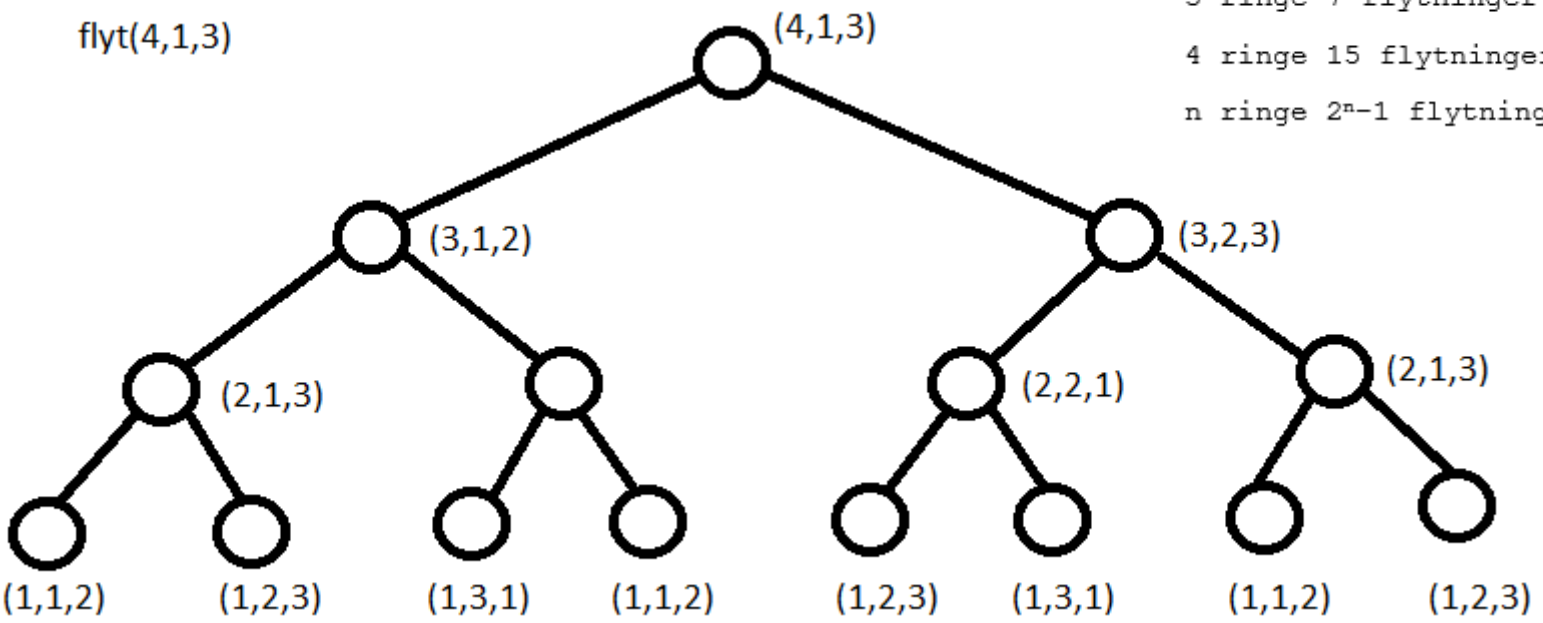
```
public static int ackermann(int x, int y) {
    int result;
    if (x == 0) {
        result = y + 1;
    }
    else if (y == 0) {
        result = ackermann(x - 1, 1);
    }
    else {
        int temp = ackermann(x, y - 1);
        result = ackermann(x - 1, temp);
    }
    return result;
}
```



Opgave 6: Towers of Hanoi

```
public static void flyt(int n, int fra, int til){
    if (n==1) {
        System.out.println("Flyt fra " +fra +" til " + til);
    }
    else {
        int temp = 6 - fra - til;
        flyt(n-1,fra,temp);
        System.out.println("Flyt fra " + fra +" til " + til);
        flyt(n-1,temp,til);
    }
}
```

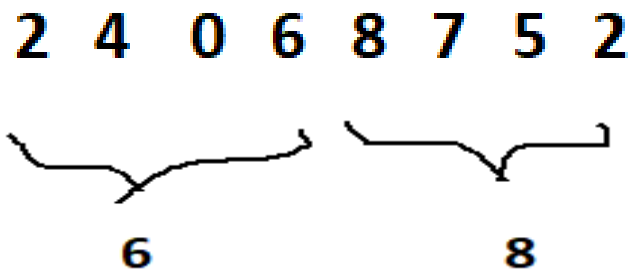
```
1 ring 1 flytning
2 ringe 3 flytninger
3 ringe 7 flytninger
4 ringe 15 flytninger
n ringe 2ⁿ−1 flytninger
```



Rekursivt nedbrydeligt problem

Et problem er rekursivt nedbrydeligt, hvis det

- Er simpelt og kan nedbrydes direkte
- Ikke er simplet, men kan nedbrydes i (typisk to) delproblemer, der hver især er simplere, men af samme type som det oprindelige, og hvis løsningen kan kombineres til en løsning på det oprindelige problem



Del, løs og kombiner skabelonen

Metode: Løs(P)

hvis p er simpel så returner L = løsning

ellers

opdel P i P1 og P2;

L1 = Løs(P1);

L2 = Løs(P2);

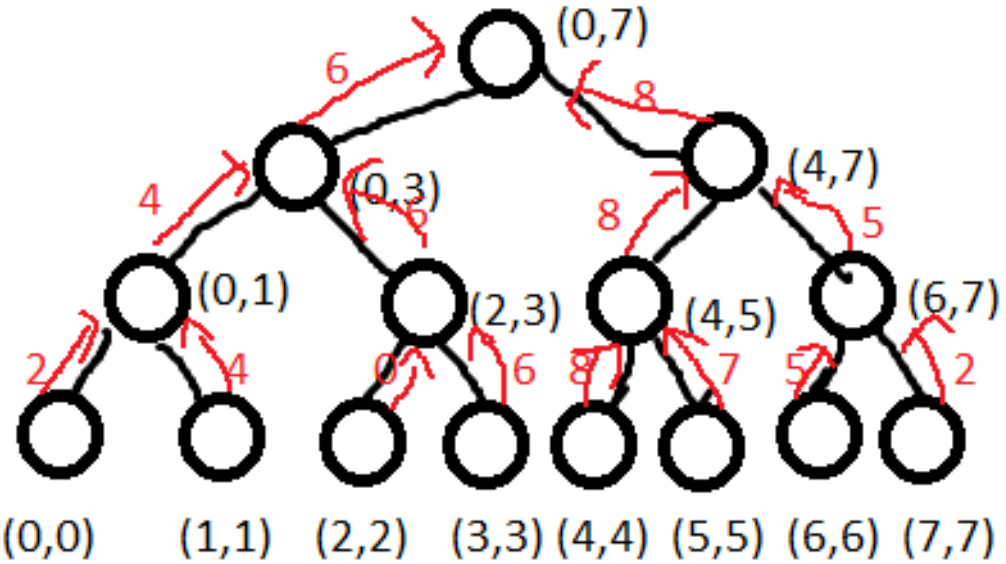
L = kombinér(L1,L2);

returner L;

Del, løs og kombiner skabelonen

```
private int maximum(ArrayList<Integer> list, int l, int h) {
    if (l == h) {
        return list.get(l);
    } else {
        int m = (l + h) / 2;
        int max1 = maximum(list, l, m);
        int max2 = maximum(list, m + 1, h);
        if (max1 > max2) {
            return max1;
        } else {
            return max2;
        }
    }
}
```

2 4 0 6 8 7 5 2



MergeSort-1

- At sortere en liste er et rekursivt nedbrydeligt problem.
- MergeSort er en effektiv sorteringsalgoritme der benytter rekursion.
- MergeSort er karakteriseret ved en simpel opsplitning i simplere problemer. Arbejdet ligger i at kombinere løsningerne fra de simplere problemer.
  - easy to split – hard to join

MergeSort-2

del listen deles i to lige store dele

løs rekursive kald på hver af de to dele af listen

kombiner

de to sorterede dele flettes til en sorteret liste

ERHVERVSAKADEMI ÅRHUS

MergeSort-3

4, 2, 12, 7, 8, 6, 9, 1

2, 4, 7, 12, 1, 6, 8, 9

1, 2, 4, 6, 7, 8, 9, 12

QuickSort - 1

- Med QuickSort er det svært at lave opdelingen, men nemt at kombinerer løsningerne.
  - hard to split – easy to join
- QuickSort er næsten altid hurtigere end MergeSort, men har den dårligste ”worst case” performance.
- Idéen med QuickSort er at lave en opsplitning af en liste i to, så alle elementer i den ene del er mindre end alle elementer i den anden del.

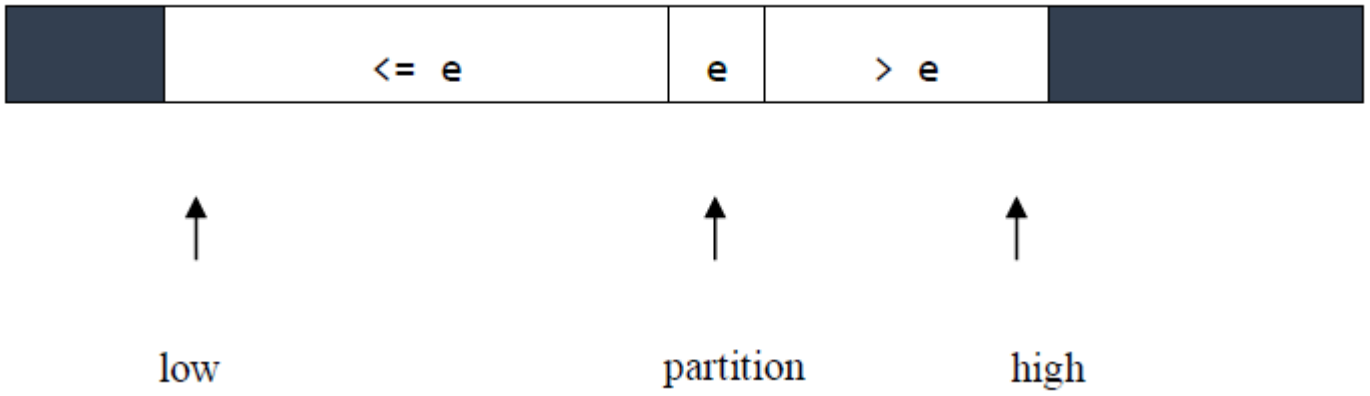
QuickSort-2

del listen deles så alle elementerne til venstre er mindre end et element og alle til højre er større end dette element

løs rekursive kald på hver af de to dele af listen

kombiner

der er intet at kombinere



QuickSort - 3

partition(low, high):

Indfør to tællevariabler i og j, hvor

```
i == low + 1
j == high
```

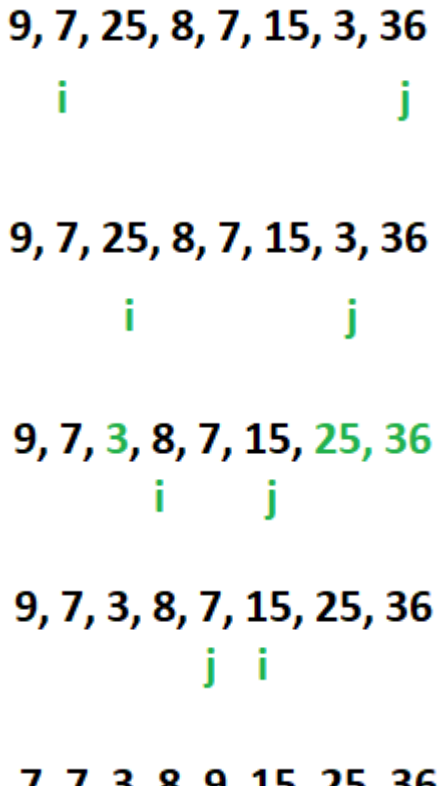
Gør herefter følgende så længe i <= j:

```
i++                          hvis list[i] <= pivot
j--                          ellers hvis list[j] > pivot
swap(i, j); i++; j--;        ellers
```

Slut af med:

```
swap(low, j) og returnér j
```

QuickSort - 4




```
private int partition(ArrayList<Integer> list, int low, int hi
    int e = list.get(low);
    int i = low + 1;
    int j = high;
    while (i <= j) {
        if (list.get(i) <= e) {
            i++;
        }
        else if (list.get(j) > e) {
            j--;
        }
        else {
            swap(list, i, j);
            i++;
            j--;
        }
    }
    swap(list, low, j);
    return j;

}
```