UPPSALA UNIVERSITET

Sortering

Tom Smedsaas

Experiment med några sorteringsmetoder.

Avsikten är att visa hur man kan verifiera teori med experiment samt att peka på när rekursion är bra och när den är dålig.

UPPSALA UNIVERSITET

Insticksortering igen

```
def ins_sort_rec(lst):
    return _ins_sort_rec(lst, len(lst))


def _ins_sort_rec(lst, n):
    if n > 1:
        _ins_sort_rec(lst, n-1)
        x = lst[n-1]
        i = n - 2
        while i >= 0  and lst[i] > x:
            lst[i+1] = lst[i]
            i -= 1
        lst[i+1] = x
    return lst
```

UPPSALA UNIVERSITET

Mergesort igen

```
def merge_sort(lst):
    if len(lst) <= 1: return lst
    else:
        n = len(lst)//2
        l1 = lst[:n]
        l2 = lst[n:]
        l1 = merge_sort(l1)
        l2 = merge_sort(l2)
        return merge(l1, l2)

def merge(l1, l2):
    if len(l1) == 0:
        return l2
    elif len(l2) == 0:
        return l1
    elif l1[0] <= l2[0]:
        return [l1[0]] + merge(l1[1:], l2)
    else:
        return [l2[0]] + merge(l1, l2[1:])
```

Hur kommer tiden växa för dessa metoder?

Vi ska titta på hur tiderna förändras när storleken dubblas.

Insticksorteringen är en $\Theta(n^2)$ – metod.

Dubblas storleken växer tiden med en faktor $\frac{c\cdot(2n)^2}{c\cdot n^2} = 4$

Samsorteringen är en $\Theta(n \log n)$ – metod.

Dubblas storleken växer tiden med en faktor

$$\frac{c \cdot 2n \log(2n)}{c \cdot n \log n} = 2 \cdot \frac{\log 2 + \log n}{\log n} = 2 \cdot (1 + \frac{1}{\log_2 n})$$

Om $n > 1000$ är faktorn mindre än 2.1.

UPPSALA UNIVERSITET

Testkod

```
sort_functions = [ins_sort_rec, merge_sort]


for sort in sort_functions:
    print('\n', sort.__name__)
    for n in [1000, 2000, 4000, 8000]:
        lst = []
        for i in range(n):
            lst.append(random.random())
        tstart = time.perf_counter()
        lst = sort(lst)
        tstop = time.perf_counter()
        print(f" Time for {n}\t : {tstop - tstart:4.2f}")
```

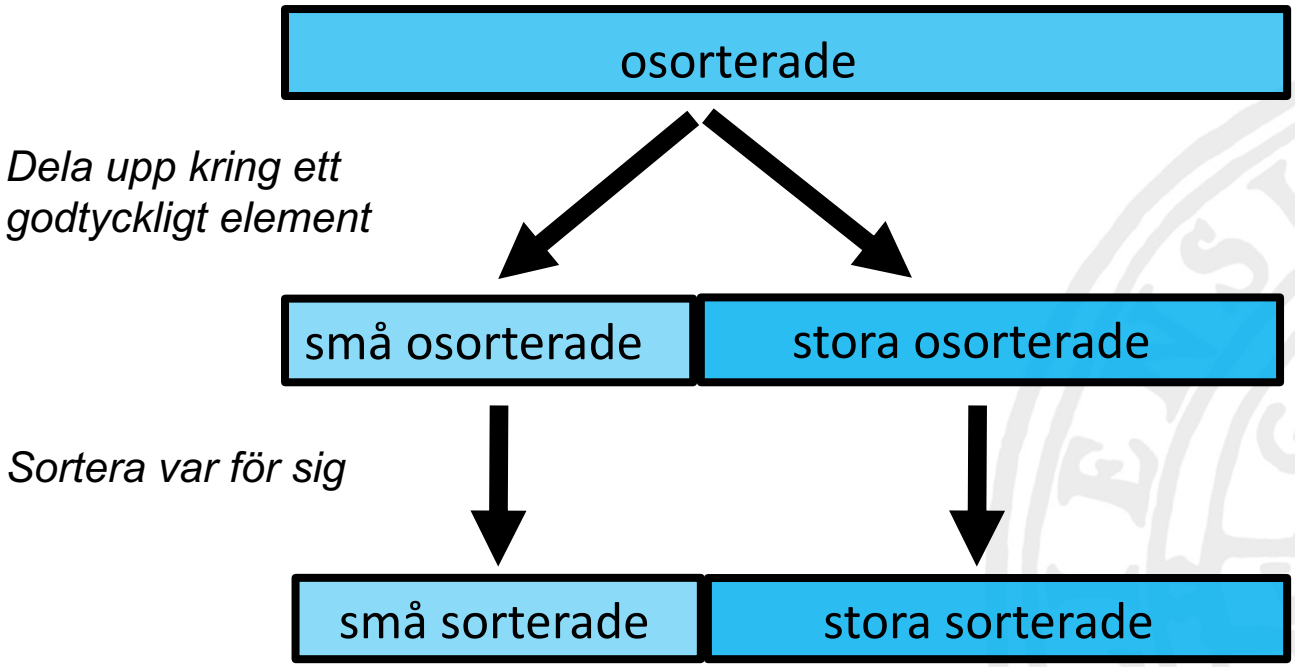
UPPSALA UNIVERSITET

Testkörning

UPPSALA UNIVERSITET

En annan sorteringsmetod

Idé: Börja med att ordna om elementen så att ”små” ligger (osorterade) till vänster och ”stora” ligger (osorterade) till höger.



UPPSALA UNIVERSITET

Partition sort

```
def psort(lst):
    _psort(lst, 0, len(lst)-1)
    return lst

def _psort(lst, n, m):
    if m > n:
        ip = partition(lst, n, m)
        _psort(lst, n, ip-1)
        _psort(lst, ip+1, m)
```

Likheter och skillnader mot mergesort?

UPPSALA UNIVERSITET

Partitioneringen

```
def partition(lst, n, m):
    if m>n:
        p = lst[n]
        i = n
        j = m
        while j > i:
            while j>i and lst[j] > p:
                j -= 1
            lst[i] = lst[j]
            while i < j and lst[i] < p:
                i += 1
            lst[j] = lst[i]
            j -= 1
        lst[i] = p
        return i
```

Anmärkning: Metoden kallas ofta för *quicksort.*

UPPSALA UNIVERSITET

Demo

UPPSALA UNIVERSITET

Sammanfattning

- Tidmätningar kan användas för att verifiera teoretiska resultat.
- De teoretiska resultaten för instickssortering och samsortering stämmer mycket bra i praktiken.
- $\Theta(n \log n)$ är mycket bättre än $\Theta(n^2)$!
- Bra att balansera algoritmerna.
- Rekursera inte över långa listor.
- Inga problem med rekursiondjupet i mergesort och quicksort.

UPPSALA UNIVERSITET

The end

.