Sortering

GruDat
Örjan Ekeberg
Enkla metoder
Bubbelsortering
Insättningssortering
Urvalssortering
Effektiva metoder
Teoretiska gränser

Sortering

Ordna element enligt relation mellan nyckelvärden

- Flera olika algoritmer med olika fördelar

GruDat
Örjan Ekeberg
Enkla metoder
Bubbelsortering
Insättningssortering
Urvalssortering
Effektiva metoder
Teoretiska gränser

Brute-force

Brute-force

Gå igenom alla permutationer och hitta den där elementen ligger i ordning

- $\mathcal{O}(n!)$, orimligt ineffektivt

GruDat
Örjan Ekeberg
Enkla metoder
Bubbelsortering
Insättningssortering
Urvalssortering
Effektiva metoder
Teoretiska gränser

Bubbelsortering

Bubbelsortering

- Byt plats på element som ligger i fel ordning
- Upprepa detta tills alla ligger rätt

Bubbelsortering

```
def bubbleSort(v):
    modified = True
    while modified:
        modified = False
        for i in range(1, len(v)):
            if v[i-1] > v[i]:
                v[i-1], v[i] = v[i], v[i-1]
                modified = True

    return v
```

GruDat
Örjan Ekeberg
Enkla metoder
Bubbelsortering
Insättningssortering
Urvalssortering
Effektiva metoder
Teoretiska gränser

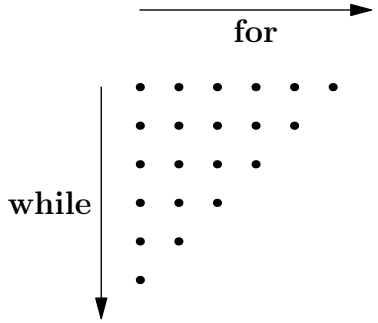
Fiffigare Bubbelsortering

```
def bubbleSort(v):
    modified = True
    surface = len(v)
    while modified:
        modified = False
        for i in range(1, surface):
            if v[i-1] > v[i]:
                v[i-1], v[i] = v[i], v[i-1]
                modified = True
        surface -= 1
    return v
```

GruDat
Örjan Ekeberg
Enkla metoder
Bubbelsortering
Insättningssortering
Urvalssortering
Effektiva metoder
Teoretiska gränser

Bubbelsortering

Hur lång tid tar bubbelsortering?



- Komplexitet: $\mathcal{O}(n^2)$

GruDat
Örjan Ekeberg
Enkla metoder
Bubbelsortering
Insättningssortering
Urvalssortering
Effektiva metoder
Teoretiska gränser

Insättningssortering

Rekursiv ansats

- Tag bort ett element
- Sortera resten
- Sätt in det borttagna på rätt plats

- Insättningssortering

Insättingssortering

```
def insertionSort(v):
  if len(v) <= 1:
    return v
  x = v[0]
  a = insertionSort(v[1:])

  p = 0
  while p<len(a) and x>a[p]:
    p += 1
  a.insert(p, x)
  return a
```

GruDat
Örjan Ekeberg
Enkla metoder
Bubbelsortering
Insättningssortering
Urvalssortering
Effektiva metoder
Teoretiska gränser

Urvalssortering

Alternativ rekursiv ansats

- Tag bort största element
- Sortera resten
- Sätt in det borttagna sist

- Urvalssortering

GruDat
Örjan Ekeberg
Enkla metoder
Bubbelsortering
Insättningssortering
Urvalssortering
Effektiva metoder
Teoretiska gränser

Urvalssortering

```
def selectionSort(v):
  if len(v) <= 1:
    return v
  im, m = 0, v[0]
  for ix, x in enumerate(v):
    if x>m:
      im, m = ix, x
  del v[im]
  return selectionSort(v)+[m]
```

GruDat
Örjan Ekeberg
Enkla metoder
Bubbelsortering
Insättningssortering
Urvalssortering
Effektiva metoder
Teoretiska gränser

Bubbelsortering	$\mathcal{O}(n^2)$
Insättningssortering	$\mathcal{O}(n^2)$
Urvalsortering	$\mathcal{O}(n^2)$

Samsortering

Smartare rekursiv ansats

- Dela på *mitten*
- Sortera båda delarna
- Sätt samman

- Samsortering (Mergesort)

Samsortering

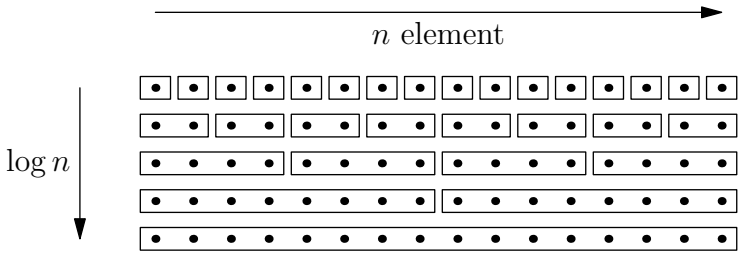
```
def mergeSort(v):
  if len(v) <= 1:
    return v
  a = mergeSort(v[:len(v)//2])
  b = mergeSort(v[len(v)//2:])
  return merge(a, b)
```


Samsortering, hjälpfunktion

```
def merge(a, b):
  v = []
  while a and b:
    if a[0] < b[0]:
      v.append(a.pop(0))
    else:
      v.append(b.pop(0))
  while a:
      v.append(a.pop(0))
  while b:
      v.append(b.pop(0))
  return v
```

Samsortering

Hur lång tid tar samsortering?



- Komplexitet: $\mathcal{O}(n \log n)$
- Nackdel: Kräver extra utrymme

Quicksort

Ännu en rekursiv ansats

- Välj ett element
- Lägg de som är mindre till vänster och de som är större till höger
- Sortera delarna

- Quicksort

Quicksort

```
def quickSort(v):
  if len(v) <= 1:
    return v
  p = v[0]
  small = [x for x in v[1:] if x<=p]
  large = [x for x in v[1:] if x>p]
  return quickSort(small) + \
         [p] + \
         quicksort(large)
```


Quicksort

Hur lång tid tar quicksort?

- Medelfallet: $\mathcal{O}(n \log n)$
- Värsta fall: $\mathcal{O}(n^2)$

- Kan göras utan extra minne (*in-place*)

Quicksort, in place

```
def quickSort(v, a, b):
  if b>a:
    s = partition(v, a, b)
    quickSort(v, a, s)
    quickSort(v, s+1, b)
```

Quicksort, del 2

```
def partition(v, a, b):
  p = v[a]
  left = a+1
  right = b-1

  while left<=right:
    while left<=right and v[left] <= p:
      left += 1
    while left<=right and v[right] > p:
      right -= 1
    if left<right:
      v[left], v[right] = v[right], v[left]

  v[a], v[right] = v[right], v[a]
  return right
```

GruDat
Örjan Ekeberg
Enkla metoder
Effektiva metoder
Samsortering
QuickSort
Teoretiska gränser

Kan man sortera snabbare än $\mathcal{O}(n \log n)$?

- Sortering $\equiv$ välj rätt permutation
- Antal tänkbara permutationer: $n!$
- Varje jämförelse kan som bäst halvera antalet
- $\frac{n!}{2^m} = 1$
- $m = \log_2(n!)$ jämförelser krävs

$$\left(\frac{n}{3}\right)^n < n! < \left(\frac{n}{2}\right)^n$$

$$\mathcal{O}(\log n!) \equiv \mathcal{O}(n \log n)$$

GruDat
Örjan Ekeberg
Enkla metoder
Effektiva metoder
Teoretiska gränser