Velkommen til IN2010 gruppe 6

Mini promo

Will Code for drinks

https://peoply.app/events/TISASELC

# Dagens plan

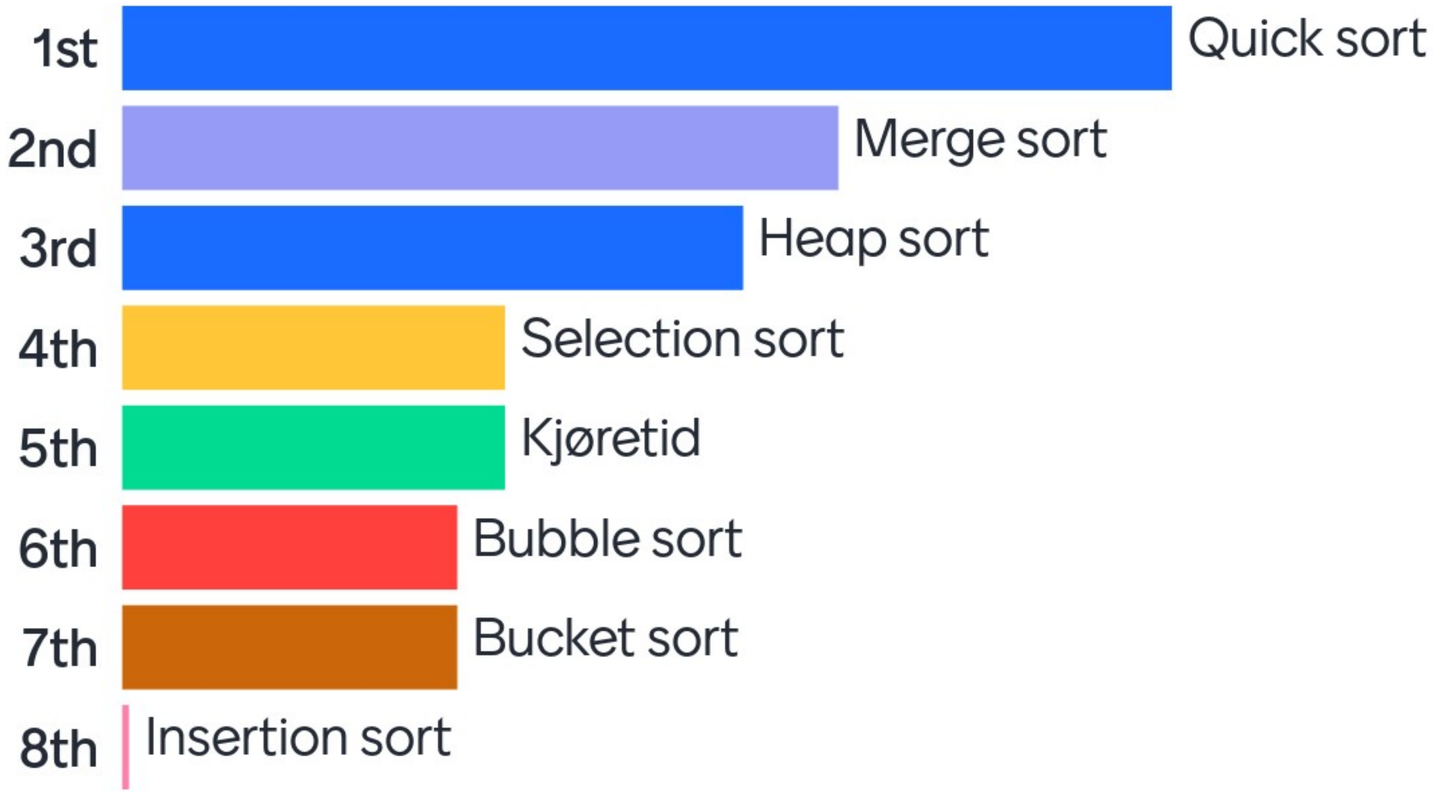
- Sortering
- Kjøretid
- Oppgave gjennomgang
- Lab

Tips til hvordan man fordøyer ukas pensum

- Prøv å forstå algoritmene ut ifra pseudo koden
- Deretter implementer koden basert på egen kunnskap
- Veldig lurt å visualisere ved bruk av tegninger eller nettsider

Mentimeter

# Hva var det vanskeligste med ukas pensum?




5

# Hvorfor sortere?

- Først og fremst så ser det gøy ut
- Lager datastrukturer som tillater raskere algoritmer
  - Sparer mye tid

## Bubble sort

- Går over hele listen to n * n ganger
- Sammenligner elementer med neste element
- Dersom neste element er større byttes plass

```
a = [9,4,5,3,2,8,0,1]
n = len(a)
for i in range(n-1):
    for l in range(n-i-1):
        if a[l] > a[l+1]:
            # her bytter man mellom l og l+1
            a[l], a[l+1] = a[l+1], a[l]
```

Selection sort

- Går over listen n * n ganger
- Sammenligner elementer med det minste elementet i resten av listen
- Dersom det minste elementet er mindre byttes det

```
n = len(a)
for i in range(n):
    k = i
    for l in range(i+1,n):
        if a[l] < a[k]:
            k = l
    if i != k:
        # her skjer byttet
        a[i], a[k] = a[k], a[i]
```

Insertion sort

- Itererer over listen en gang og flytter elementet til venstre
- Sparer mye tid på nesten sorterte lister

```
a = [1,3,6,2,7,8,5]
n = len(a)
for i in range(1, n):
    while i > 0 and a[i-1] > a[i]:
        # her skjer byttet
        a[i-1], a[i] = a[i], a[i-1]
        i -= 1
```

Merge sort

- Halverer listen rekursivt helt til de er 1 store
- Setter sammen mini-listene sammen sortert
  - Plukker og henter riktig element fra de mindre listene

```
def sort(A):
    if len(A) <= 1:
        return A
    midten = len(A)//2
    A1 = sort(A[:midten])
    A2 = sort(A[midten:])

    i = 0
    l = 0
    while i < len(A1) and l < len(A2):
        if A1[i] <= A2[l]:
            A[i + l] = A1[i]
            i += 1
        else:
            A[i + l] = A2[l]
            l += 1

    while i < len(A1):
        A[i + l] = A1[i]
        i += 1

    while l < len(A2):
        A[i + l] = A2[l]
        l += 1
    return A
```

Mentimeter

Mini pause

Mentimeter

Noen begreper

## Stabilitet

- La oss si at vi sorterer basert på en verdi **k**
  - Verdien **k** kan være alt fra bokstaver, heltalls verdier eller lengde på noe

- Dersom to elementer x og y med samme verdi **k** eksisterer i en liste skal x ligge til venstre for y før sortering og etter sortering

## In-place

- Bruker ingen andre eksterne datastrukturer i algoritmen
- Analyse av minne bruk vil gi O(1) dersom den er in-place

Mentimeter

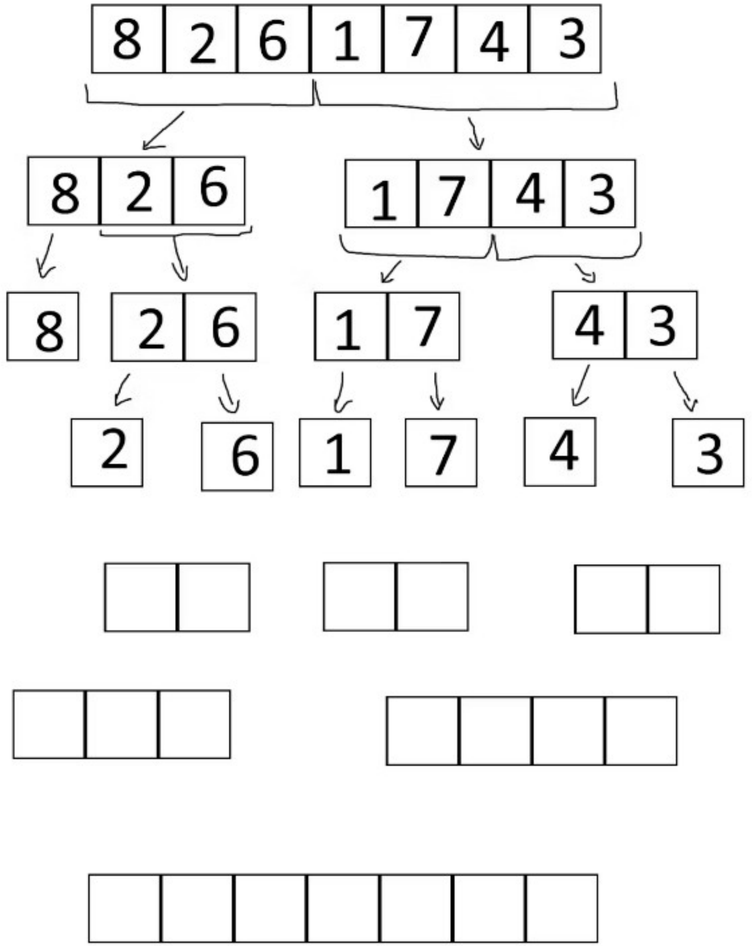
# Merge sort eksempel

```python
def sort(A):
    if len(A) <= 1:
        return A
    midten = len(A)//2
    A1 = sort(A[:midten])
    A2 = sort(A[midten:])

    i = 0
    l = 0
    while i < len(A1) and l < len(A2):
        if A1[i] <= A2[l]:
            A[i + l] = A1[i]
            i += 1
        else:
            A[i + l] = A2[l]
            l += 1

    while i < len(A1):
        A[i + l] = A1[i]
        i += 1

    while l < len(A2):
        A[i + l] = A2[l]
        l += 1
    return A
```



Mentimeter

Over til noen raskere sorterings algoritmer

Heapsort

- Lager en heap med elementene i listen

- Sortere en liste med en **min**-heap (ikke in-place)
  - Popper element og setter det i en ny liste
  - Bobbler/fikser min-heapen så repeter
- Sortere en liste med en **max**-heap (in-place)
  - Flytter det største elementet bakerst i listen
  - Bobbler/fikser max-heapen men ignorerer siste index
  - bilde til høyre viser dette

```python
def bubble_down(A, i, n):
    largest = i
    left = i*2 + 1
    right = i*2 + 2

    if left < n and A[largest] < A[left]:
        left, largest = largest, left

    if right < n and A[largest] < A[right]:
        right, largest = largest, right

    if i != largest:
        A[largest], A[i] = A[i], A[largest]
        bubble_down(A, largest, n)

def build_max_heap(A, n):
    for i in range(n//2, -1, -1):
        bubble_down(A, i, n)

def heapsort(A):
    build_max_heap(A, len(A))
    for i in range(len(A)-1, -1, -1):
        A[i], A[0] = A[0], A[i]
        bubble_down(A, 0, i)
```

Quicksort

- Velger et pivot-punkt i listen med to pekere (high og low)
- pekerene går mot hverandre der hvor:
  - high går mot venstre og finner et element som er mindre enn pivot elementet
  - low går mot høyre og finner et element som er større en pivot elementet
- Dersom pekerene krysser hverandre går vi til neste iterasjon

```python
from random import randint

def partition(A, low, high):
    p = randint(low, high)
    A[p], A[high] = A[high], A[p]

    pivot = A[high]
    left = low
    right = high-1

    while left <= right:
        while left <= right and A[left] <= pivot:
            left += 1
        while right >= left and A[right] >= pivot:
            right -= 1
        if left < right:
            A[left], A[right] = A[right], A[left]

    A[left], A[high] = A[high], A[left]
    return left

def quicksort(A, low, high):
    if low >= high:
        return A
    p = partition(A, low, high)
    quicksort(A, low, p - 1)
    quicksort(A, p + 1, high)
    return A
```

# Bucket sort

- Er ingen super klar algoritme på det
  - Baserer seg på regler man lager selv
- Tanken er at man setter elementer i bøtter basert på reglene
- Bøttene vil være satt i en rekkefølge som vil følge videre til siste iterasjon

Mentimeter

Over til noen oppgaver

# Stabilitet 3 poeng

Anta at arrayet A er usortert og inneholder personobjekter som alle har et felt for alder. Anta videre at personobjektene som ligger på A[3] og A[42] begge er 22 år gamle.

Arrayet A blir sortert etter alder. Etter sorteringen får du vite at:

- Personobjektet som lå på A[3] før sorterting, ligger nå på A[9]
- Personobjektet som lå på A[42] før sorterting, ligger nå på A[7]

(a) Var sorteringen stabil?

(b) Hva er alderen til personobjektet som ligger på A[8] etter sortering?

(c) Dersom du får vite at ingen personer i A er eldre enn 100 år gammel. Hvilken sorteringsalgoritme bør da benyttes, med hensyn til kjøretidseffektivitet?

Oppgaven er hentet fra eksamen 2021

## Litt sortering 10 poeng

For hver av påstandene nedenfor kan du anta at A er et array med $n$ elementer, og at $i$ er et heltall $0 \leq i < n$.

### Bubble sort

(a) Etter $i$ iterasjoner av den ytre loopen i Bubble sort, er de $i$ *første* elementene sortert.

(b) Etter $i$ iterasjoner av den ytre loopen i Bubble sort, er de $i$ *siste* elementene sortert.

(c) Bubble sort bytter kun elementer som står direkte ved siden av hverandre.

(d) Bubble sort garanterer et minimalt antall bytter.

### Selection sort

(e) Etter $i$ iterasjoner av den ytre loopen i Selection sort, er de $i$ *første* elementene sortert.

(f) Etter $i$ iterasjoner av den ytre loopen i Selection sort, er de $i$ *siste* elementene sortert.

(g) Selection sort bytter kun elementer som står direkte ved siden av hverandre.

(h) Selection sort garanterer et minimalt antall bytter.

### Insertion sort

(i) Etter $i$ iterasjoner av den ytre loopen i Insertion sort, er de $i$ *første* elementene sortert.

(j) Etter $i$ iterasjoner av den ytre loopen i Insertion sort, er de $i$ *siste* elementene sortert.

(k) Insertion sort bytter kun elementer som står direkte ved siden av hverandre.

(l) Insertion sort garanterer et minimalt antall bytter.

Oppgaven er hentet fra eksamen 2021