Velkommen til IN2010 Repitisjonstime for sortering

(Bubblesort, Selectionsort, Insertionsort)

Repetisjons plan for sortering

- Denne uken
  - Sortering generelt
  - Bubble, Selection, Insertion
- Neste uke (fredag og ikke torsdag)
  - Merge, Quick
- Neste neste uke (torsdag og ikke fredag)
  - Heap, Bucket, Radix

Planen for i dag

- Motivasjon for sortering
- Hvordan man sorterer naivt
- Kjøretid til sorteringsalgoritmene
- Gjennomgang av eksamensoppgaver
- Kattis oppgaver

Hvorfor sortere?

- Se for deg en fin jul og du har fått favorittleken din i gave… LEGO!
- En dag åpner du opp legoen og heller ALLE legobitene i en boks og rister på boksen.
- Så starter du å sette sammen legoen og bruker O(n) tid på å lete for hver eneste brikke…
- Juleferien er ødelagt

Med sortering så kan vi...

- **kategorisere** forskjellige elementer
- finne **matches** av elementer i to forskjellige lister
- gjøre **søk** etter et element i et **array** betydelig raskere

Over til noen begreper

In-place

- En algoritme bruker kun **datastrukturen** fra **input**
- Er ikke **in-place** dersom det skapes **midlertidige datastrukturer** under kjøring

- Eksempel på ikke **In-place**, Two-sum:
  - Du får en **liste med tall** og et **mål**
  - Skal finne **to elementer** i listen som summerer opp til **målet**
    - Skal returnere indeksen til disse to elementene
  - Kan løses i O(n) tid ved å…

In-place

- En algoritme bruker kun **datastrukturen** fra **input**
- Er ikke **in-place** dersom det skapes **midlertidige datastrukturer** under kjøring

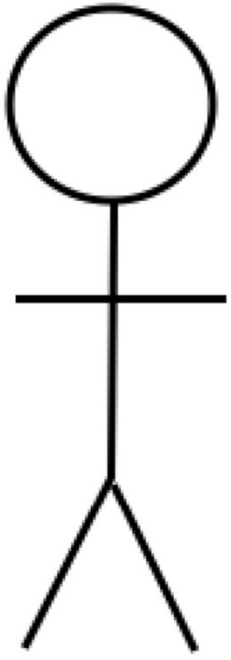
- Eksempel på ikke **In-place**, Two-sum:
  - Du får en **liste med tall** og et **mål**
  - Skal finne **to elementer** i listen som summerer opp til **målet**
    - Skal returnere indeksen til disse to elementene
  - Kan løses i O(n) tid ved å gå over listen og lagre elementene i et HashMap der hvor element --> indeks også slå opp for hvert element om de matcher

```python
def two_sum(nums: List, target: int):
    num_map = {}

    for i, num in enumerate(nums):
        difference = target - num

        if difference in num_map: # look up difference in dict
            return [num_map[difference], i]

        num_map[num] = i

    return []
```

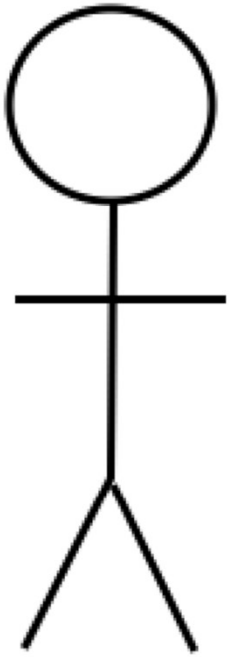
Stabilitet

- Tar for oss elementene **a** og **b**, og **a** forekommer før **b** i listen **før sortering**
- I tillegg er **a** mindre enn **b** (a skal ligge før b i listen)
- Sorteringen er stabil dersom **a** forekomme før **b** i listen **etter sortering**
- Dersom **a** forekommer etter **b etter sorteringen** er algoritmen ikke stabil

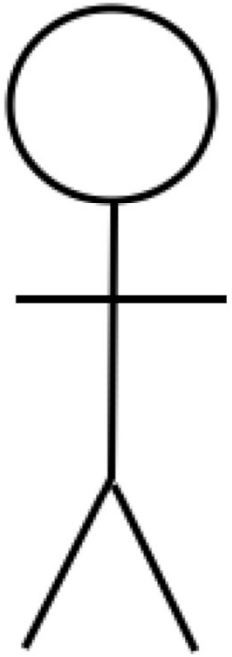
Ole
24år



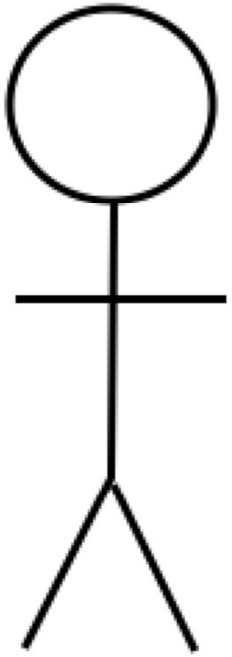
Dole
10år

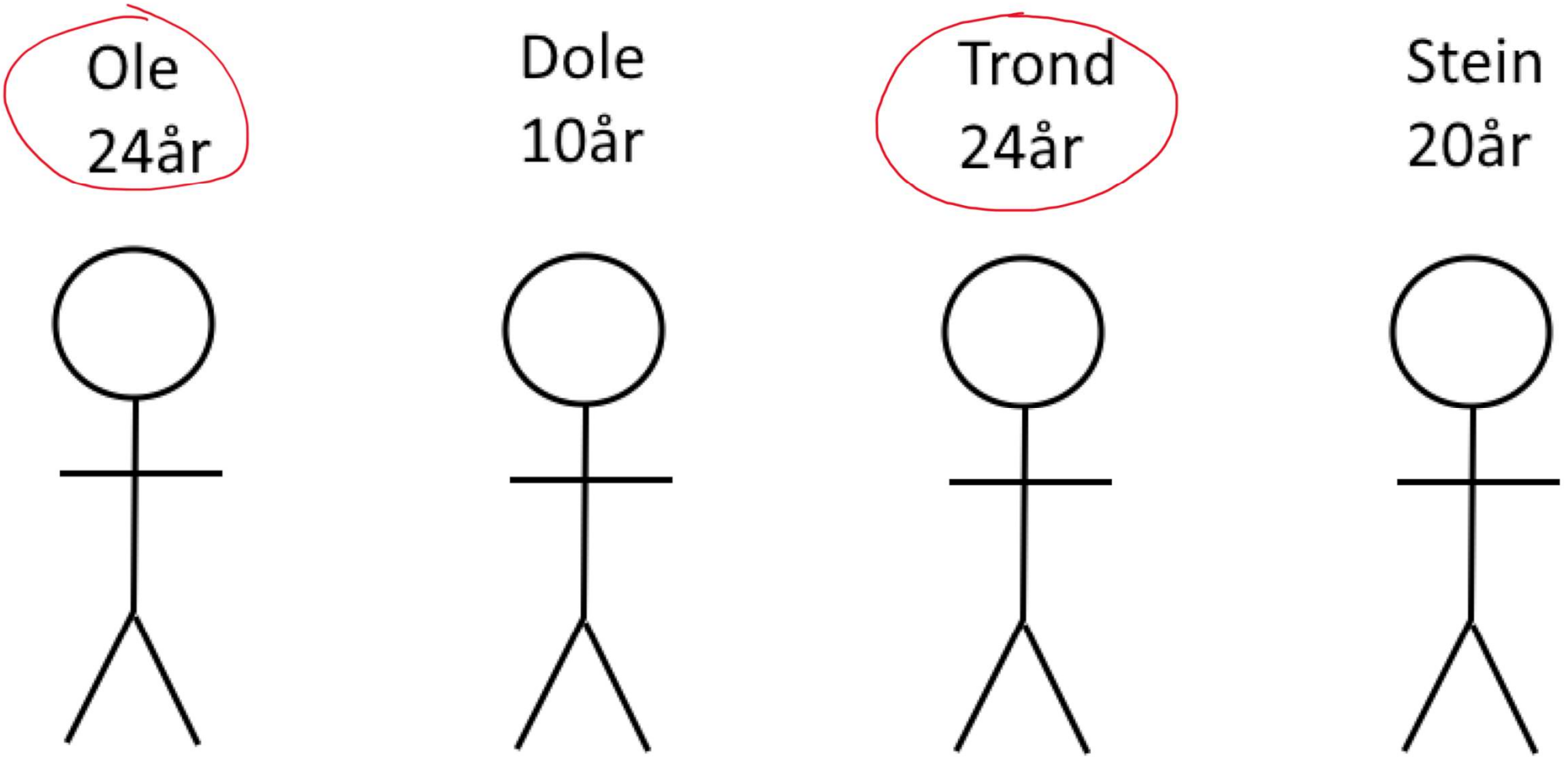


Trond
24år

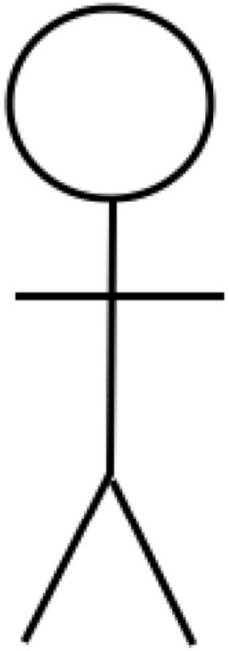


Stein
20år

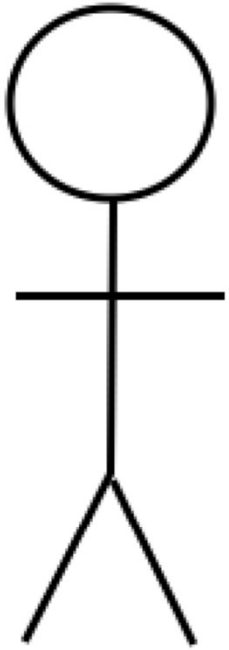



Ole
24år
Dole
10år
Trond
24år
Stein
20år

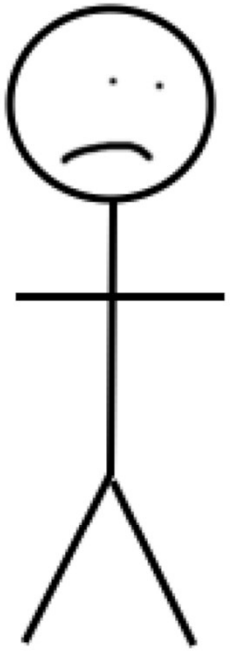
Dole
10år



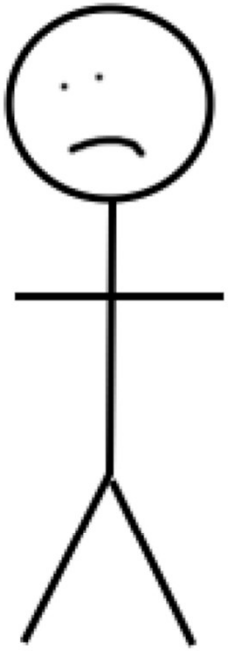
Stein
20år



Trond
24år



Ole
24år

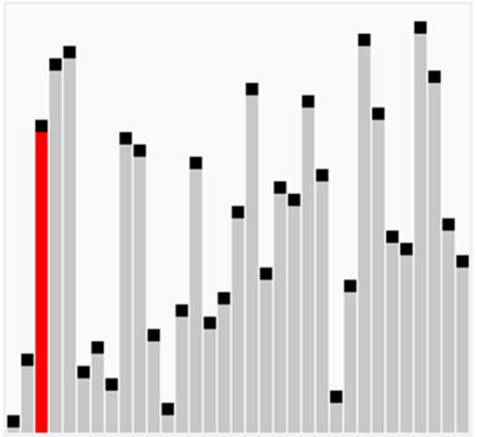


Hvordan å sortere (naivt)

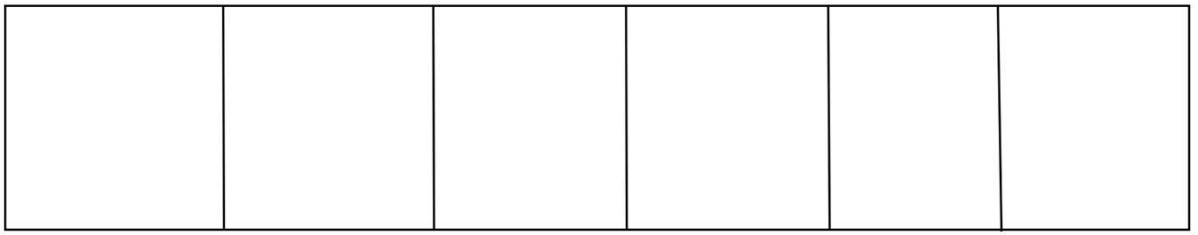
Bubble sort

- Går over listen og **drar** med seg det **største** elementet til plass **n**-1
- Går over listen igjen og **drar** opp det **nest største** opp til plass **n**-2
- Repeterer dette til **n-n**

- Mer konkret går vi over **hvert element** i listen
- For hvert element sjekker vi med **neste element**
- Dersom det neste elementet er mindre **bytter** vi om på plassene deres



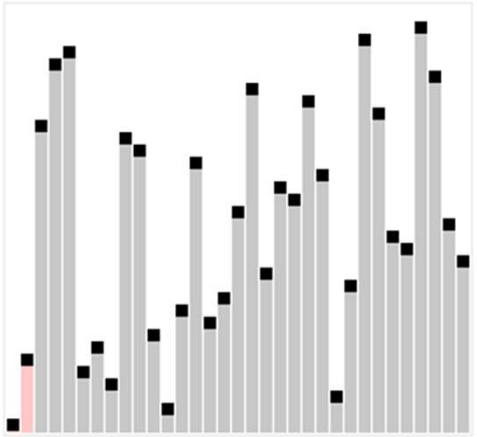
```python
n = len(a)
for i in range(n-1):
    for j in range(n-i-1):
        if a[j] > a[j+1]:
            a[j], a[j+1] = a[j+1], a[j]
```



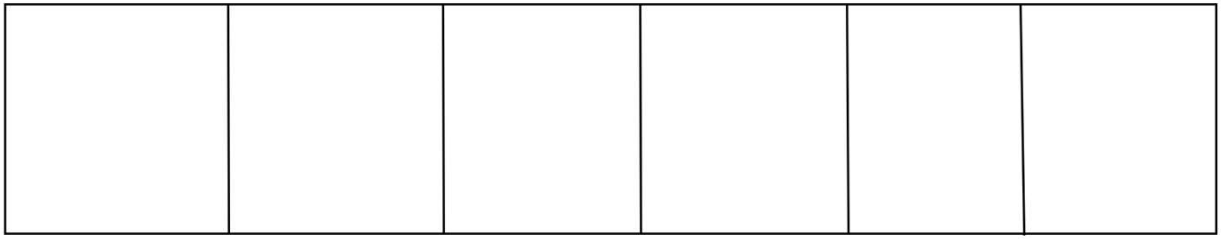
Selection sort

- Starter på **plass 0** og leter igjennom **resten av listen** (1 til n-1)
- Finner det **minste** elementet og **bytter** om plass
- Går til neste plass og repeterer

- Mer konkret går vi over hver indeks ***i***
- Finner hvor det minste elementet fra ***i***+1 til **n**-1 ligger
- Hvis det nye elementet ikke ligger på ***i*** gjør vi et bytte



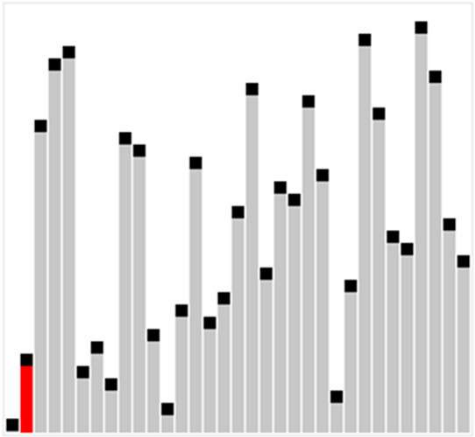
```python
n = len(a)
for i in range(n):
    k = i
    for l in range(i+1,n):
        if a[l] < a[k]:
            k = l
    if i != k:
        a[i], a[k] = a[k], a[i]
```



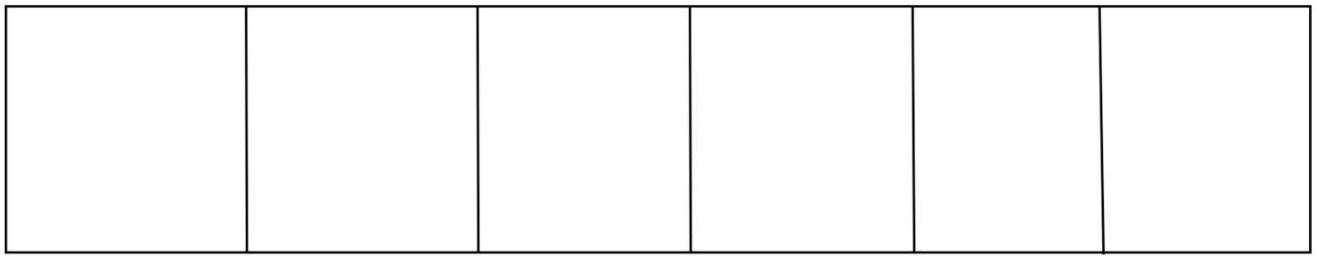
Insertion sort

- Inne i listen vi sorterer er det en **sub-liste** som er sortert til alle tider
- Ekspanderer denne sorterte listen for hver iterasjon

- Mer konkret har vi en indeks ***i*** som starter på 1
- Drar elementet på plass ***i*** bakover til det ligger sortert
- Øker ***i*** med 1 så fortsett til ***i*** = **n**



```python
n = len(a)
for i in range(1, n):
    while i > 0 and a[i-1] > a[i]:
        a[i-1], a[i] = a[i], a[i-1]
        i -= 1
```



Kjøretid

- Vi gjør ***n***-1 operasjoner (kaller dette ***i***-ende operasjon)
  - For hver av disse operasjonene gjør vi ***n*** minus ***i*** operasjoner
  - n-1 + n-2 + n-3 + n-4 … n-(n+1) => $\frac{n(n-1)}{2}$
  - $O\left(\frac{n(n-1)}{2}\right) \Rightarrow O\left(\frac{n^2-n}{2}\right) \Rightarrow O\left(\frac{n^2}{2}\right) \Rightarrow O(n^2)$

```
n = len(a)
for i in range(n-1):
    for j in range(n-i-1):
        if a[j] > a[j+1]:
            a[j], a[j+1] = a[j+1], a[j]
```

Mentimeter

Eksamensoppgaver

Finn duplikat — 12 poeng

Du er gitt et array `A` med sammenlignbare elementer. Du får vite at `A` inneholder nøyaktig ett duplikat. Altså er alle elementer i `A` unike, bortsett fra *ett* element, som forekommer nøyaktig to ganger.

(a) Anta at du er gitt et *sortert* array `A`, og får vite at x er duplikatet i `A`. Skriv en effektiv prosedyre som skriver ut de to posisjonene som inneholder x. Oppgi kjøretidskompleksiteten på algoritmen.

Input: Et *sortert* array `A` med n sammenlignbare elementer, og et element x som forekommer nøyaktig to ganger
Output: Skriver ut de to posisjonene $0 \leq i < j < n$ hvor `A[i]` $=$ `A[j]` $= x$ forekommer
Procedure `FindSortedIndicesOfDuplicate(A,` x`)`

```
// ···
```

(b) Anta at du er gitt et *sortert* array `A` (og nå får du ikke oppgitt elementet som er duplisert). Skriv en effektiv prosedyre som skriver ut de to posisjonene til elementet som forekommer to ganger. Oppgi kjøretidskompleksiteten på algoritmen.

Input: Et *sortert* array `A` med n sammenlignbare elementer
Output: Skriver ut de to posisjonene $0 \leq i < j < n$ hvor `A[i]` $=$ `A[j]`
Procedure `FindSortedDuplicateIndices(A)`

```
// ···
```

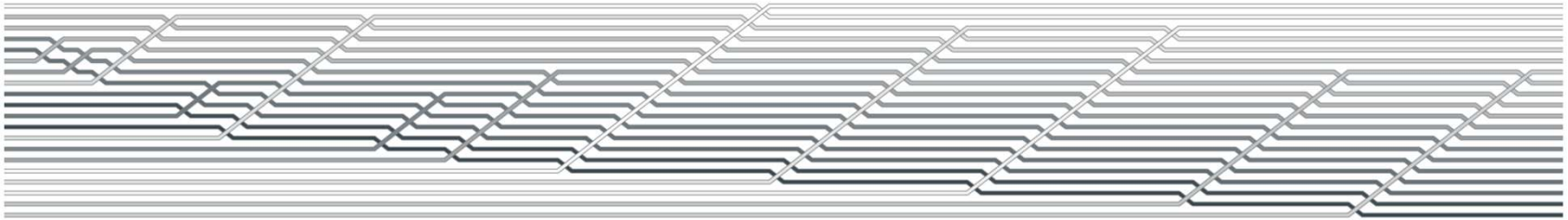
(c) Anta at du er gitt et array `A` (nå kan du ikke anta at arrayet er sortert). Skriv en effektiv prosedyre som skriver ut de to posisjonene til elementet som er duplisert. Oppgi kjøretidskompleksiteten på algoritmen.

Input: Et array `A` med sammenlignbare n elementer
Output: Skriver ut de to posisjonene $0 \leq i < j < n$ hvor `A[i]` $=$ `A[j]`
Procedure `FindDuplicateIndices(A)`

```
// ···
```

Eksamen 2023

Gnome sort — 10 poeng



Gnome sort er en enkel sorteringsalgoritme som *ikke* er kjent fra pensum. Illustrasjonen ovenfor er generert fra en kjøring av Gnome sort. Pseudokode for algoritmen er gitt nedenfor.

Input: Et array A med n elementer
Output: Et *sortert* array med de samme n elementene

```
1 Procedure GnomeSort(A)
2     i ← 0
3     while i < n do
4         if i > 0 and A[i-1] > A[i] then
5             A[i-1], A[i] ← A[i], A[i-1] // swap A[i] and A[i - 1]
6             i ← i − 1
7         else
8             i ← i + 1
```

I denne oppgaven skal du drøfte fordeler og ulemper ved Gnome sort sammenlignet med andre sorteringsalgoritmer kjent fra pensum. Teksten din bør inneholde:

- en kort forklaring på hvordan Gnome sort fungerer (med naturlig språk),
- kjøretidskompleksiteten til Gnome sort (sammen med en kort begrunnelse),
- en sammenligning av Gnome sort med sorteringsalgoritmer fra pensum, og
- en kort redegjørelse for hvorvidt Gnome sort er stabil og/eller in-place.

Eksamen 2023

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?

Eksamen 2021

Kattis oppgaver om sortering

- Emneside -> grupper -> gruppe6 -> github -> repetisjon -> README