

Velkommen til IN2010 repitisjonstimer for sortering
MergeSort, QuickSort



Dagens plan

- Mergesort
- Quicksort
- Eksamens spørsmål ish
- Kattis og andre oppgaver



Fra sist

- **Bubblesort**

- Bobbler opp de største elementene
- Bytter og sammenligner elementer som ligger ved siden av hverandre
- In-place og stabil

- **Selectionsort**

- Setter de minste elementene på plass i
- In-place og IKKE stabil (sa feil sist)
- Garantert minst antall bytter

- **Insertionsort**

- Lager en sub-liste på starten som alltid er sortert
- In-place og stabil
- Fungerer veldig bra på nesten sorterte lister

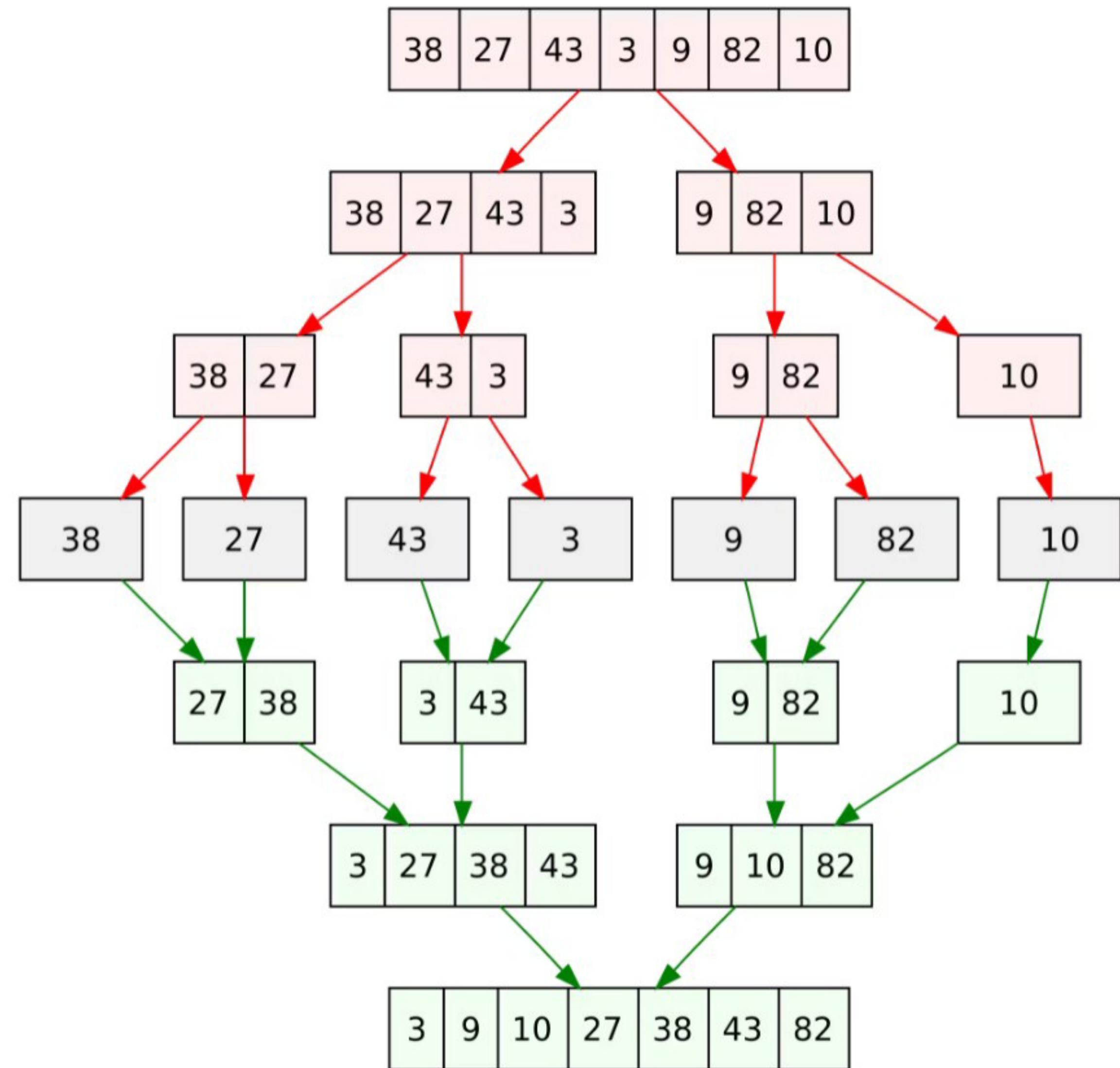
| 5 | 2 | 5 | 1 |

- Hvis vi gjør selection sort på denne så får vi den røde femmeren etter den blåe femmeren
- Dette er brudd på stabilitet



MergeSort

- Divide and conquer
- Algoritmen ligner på noe i duren av dette:
 1. Splitt listen i to deler
 2. Kombiner listen til en sortert liste



```

def mergesort(A):
    n = len(A)
    if n <= 1:
        return A
    i = n//2

    A1 = mergesort(A[:i])
    A2 = mergesort(A[i:])
    return sort(A, A1, A2)

def sort(A, A1, A2):
    i = j = 0

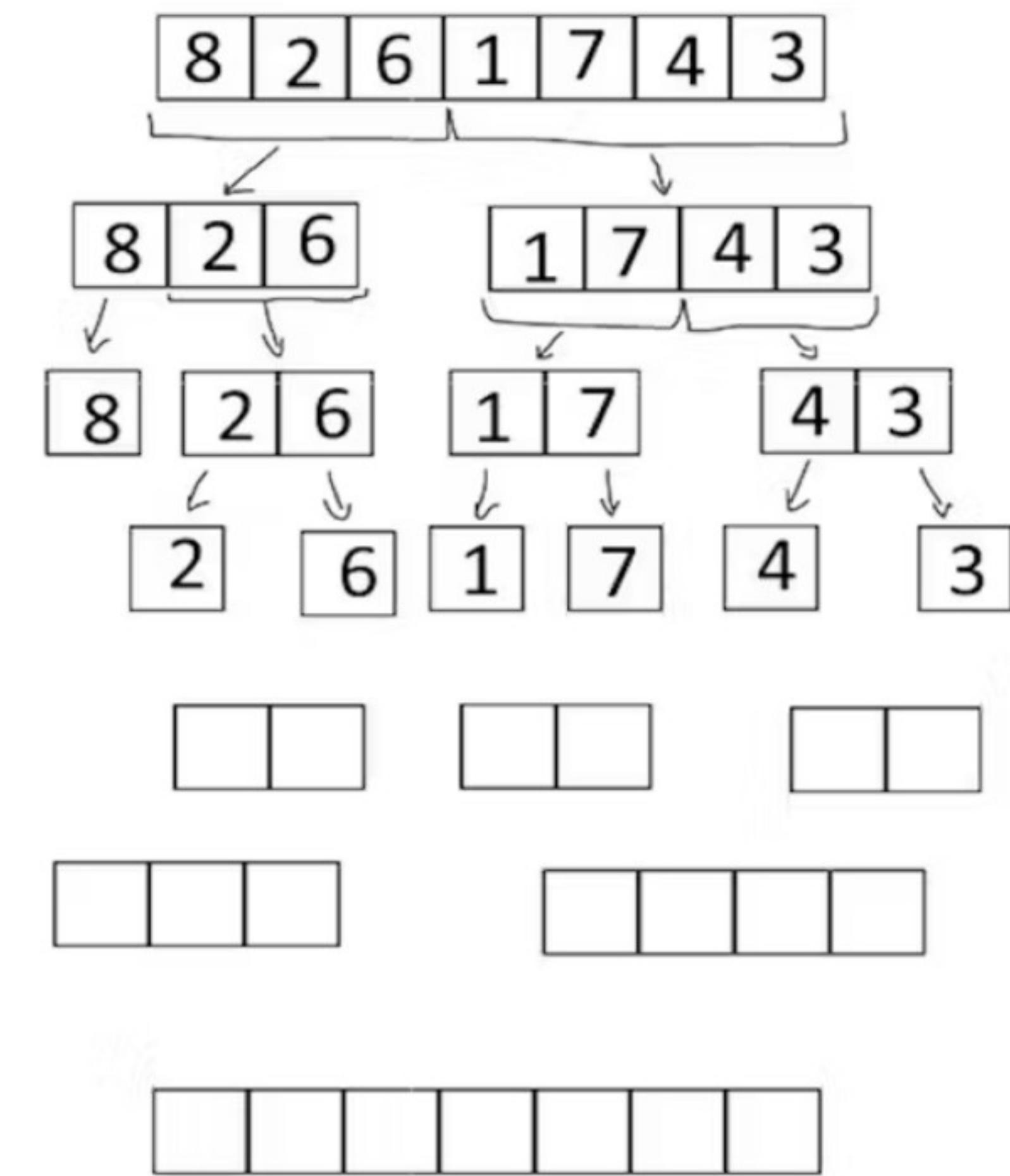
    while i < len(A1) and j < len(A2):
        if A1[i] <= A2[j]:
            A[i + j] = A1[i]
            i += 1
        else:
            A[i + j] = A2[j]
            j += 1

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

    while j < len(A2):
        A[i + j] = A2[j]
        j += 1

    return A

```



MergeSort Kjøretidsanalyse

- Deler opp hver liste i 2 rekursivt til listene ikke kan deles lenger
 - Dette er en $O(\log n)$ prosess
 - Se for dere AVL-trær der hvor man halverer antall elementer man har tilgang til for hvert hopp ned i treeet man gjør
- For hvert rekursive kall gjør vi $|liste1| + |liste2|$ antall operasjoner
 - Hvis man adderer alt sammen opp så får man til slutt $|liste|$ eller n operasjoner



Tilbake til pensum fra sist

- **In-place**

- En algoritme er In-place dersom den ikke midlertidig bruker andre datastrukturer

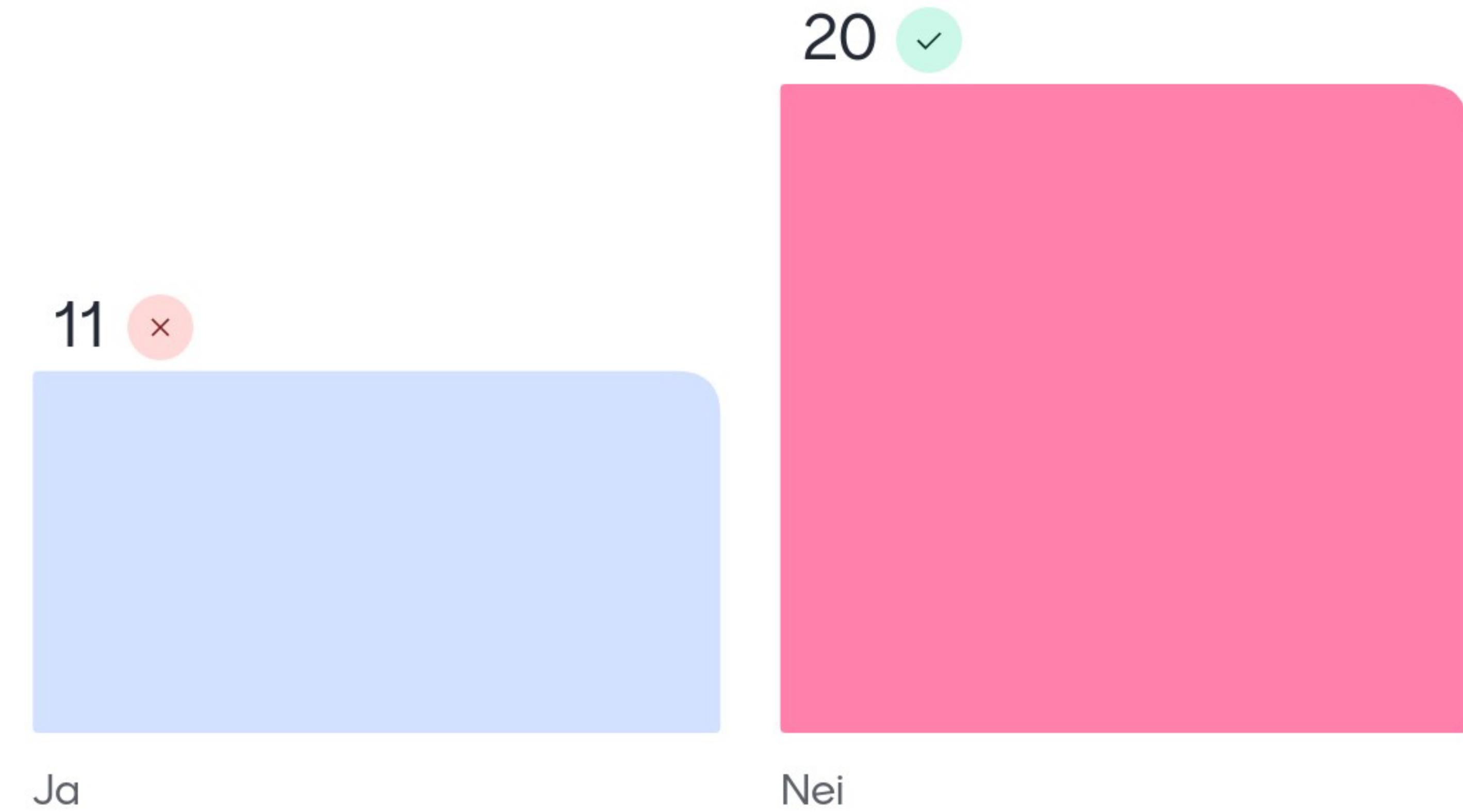
- **Stabilitet**

- Element a og b der hvor $a \leq b$ er i en liste
 - Dersom a befinner seg før b før sortering og etter sortering er sorteringen stabil

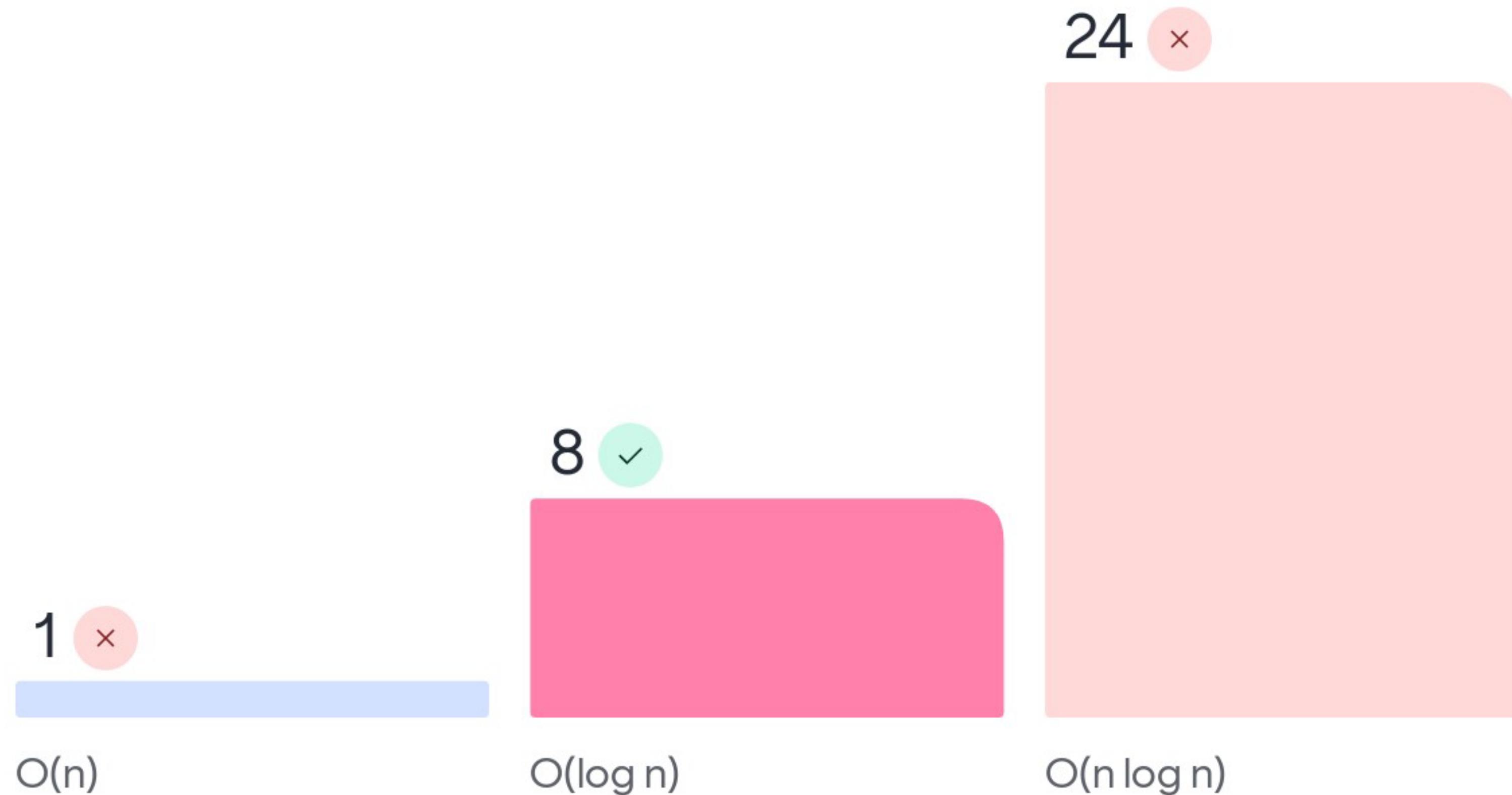


Bli med

Er MergeSort fra pensum In-place?



Hva er verste kjøretiden her?



```
def mergesort(A):  
    n = len(A)  
    if n <= 1:  
        return A  
    i = n//2  
  
    A1 = mergesort(A[:i])  
    A2 = mergesort(A[i:])  
    return sort(A, A1, A2)
```

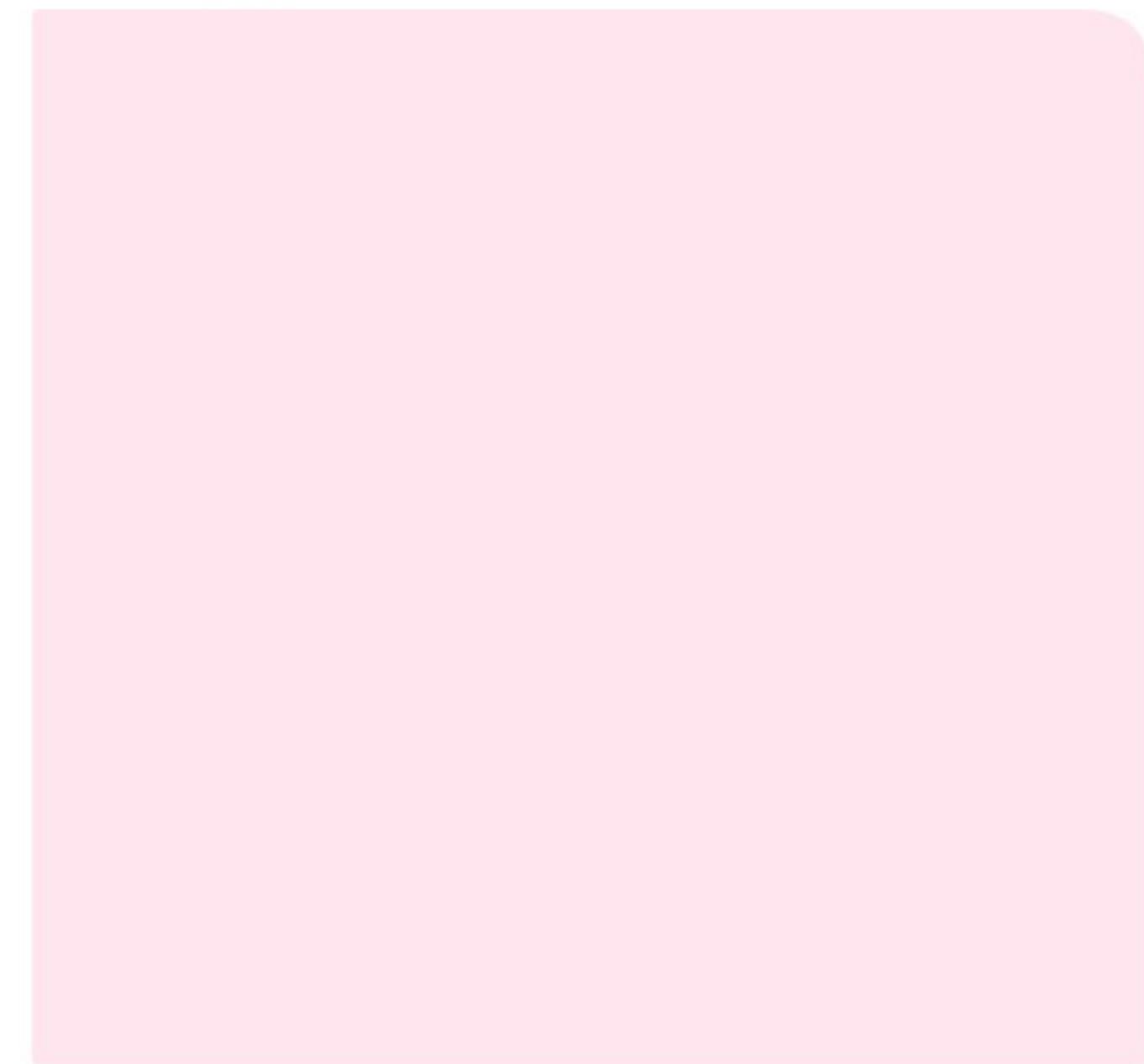
Er MergeSort fra pensum stabil?

16 ✓



Ja

16 ✗



Nei



Hva er verste kjøretiden her?

23 ✓

 $O(n)$

1 ✗

 $O(\log n)$

7 ✗

 $O(n \log n)$

```
def sort(A, A1, A2):
    i = j = 0

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

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

    while j < len(A2):
        A[i + j] = A2[j]
        j += 1

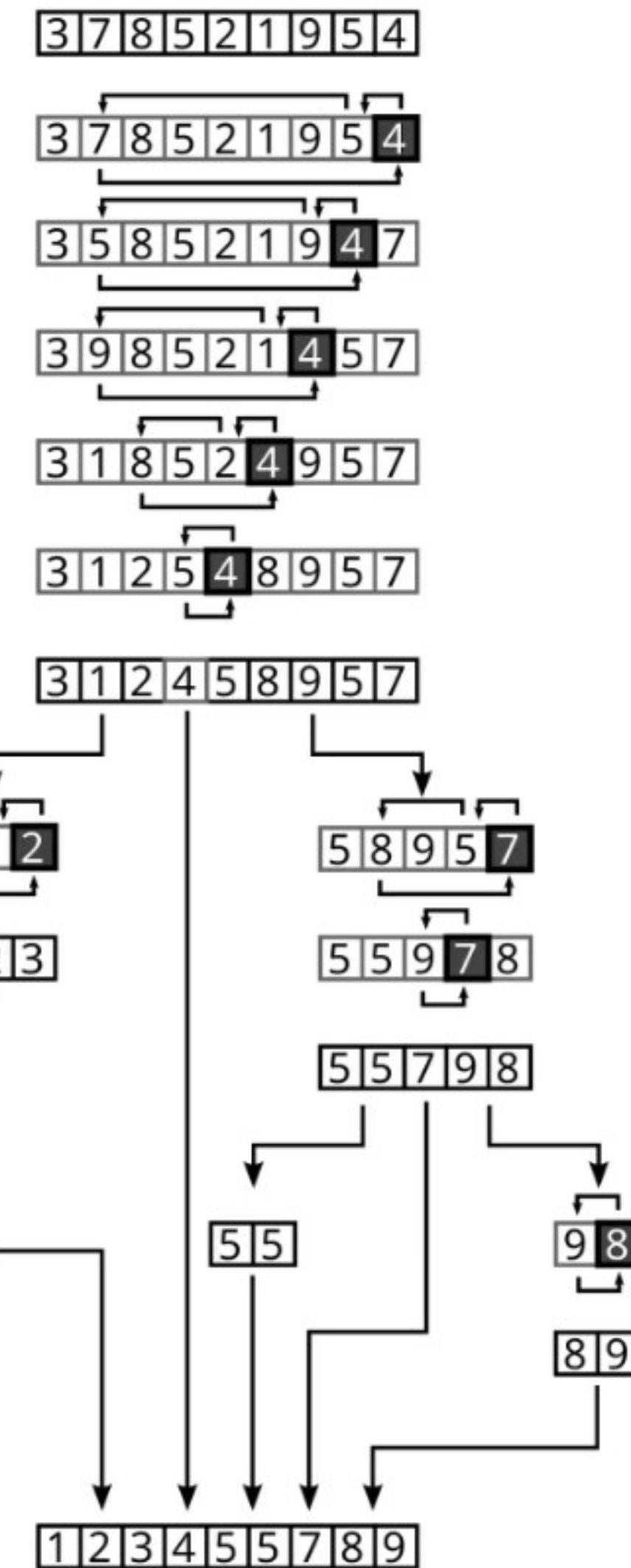
    return A
```

Quicksort :)



Quicksort

- Velger et pivot element, **P**
 - Finner medianen av elementene på første, midterste og siste indeks
 - Eller velge et tilfeldig element
- Flytter om på alle andre elementer i listen slik at:
 - Alle elementer mindre enn **P** ligger til venstre for **P**
 - Alle elementer større enn **P** ligger til høyre for **P**
- Gjør dette rekursivt på venstre og høyre siden av hvor **P** ender opp



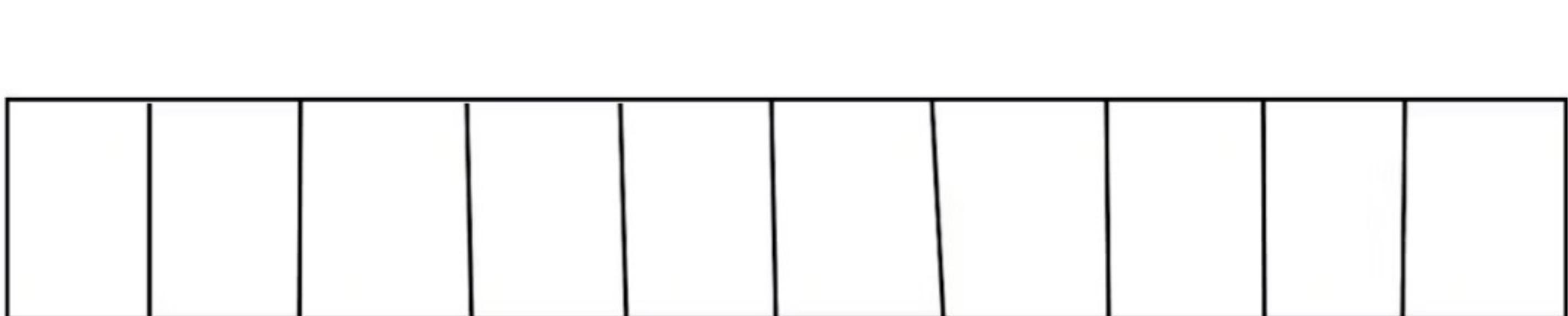
```
def partition(A, low, high):
    p = randint(low, high)
    A[high], A[p] = A[p], A[high]

    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
    p = partition(A, low, high)
    quicksort(A, low, p - 1)
    quicksort(A, p + 1, high)
    return A
```



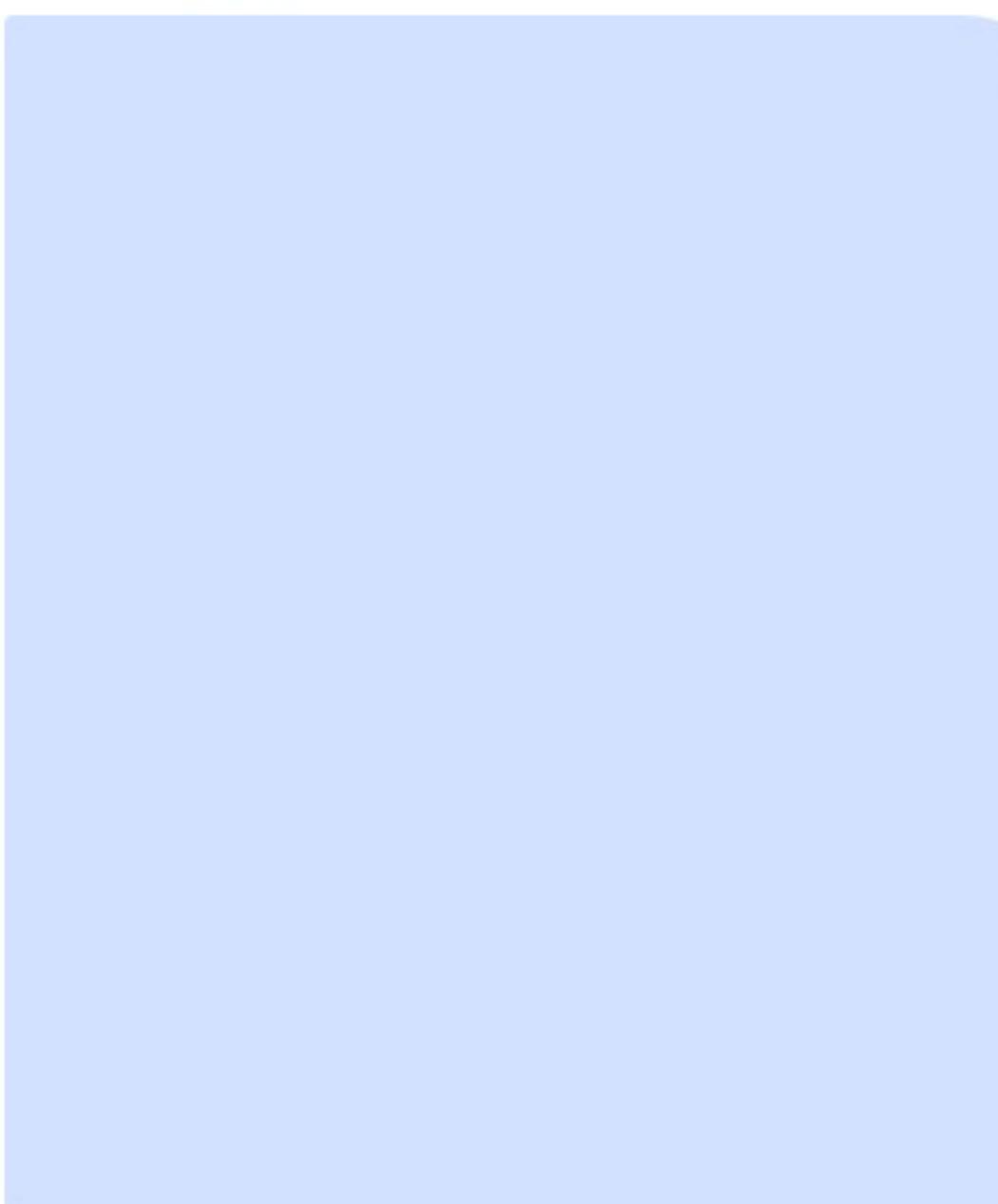
QuickSort Kjøretidsanalyse

- Deler listen i to på lik måte som mergesort, deler i to helt til $low \leq high$, til listen vi jobber med er ikke delelig
 - Dette gir en $O(\log n)$ prosess dersom man velger ikke velger en dårlig pivot i hvert kall
 - Hvis man velger dårlig pivot så blir delingen en $O(n)$ prosess
- Plassering av elementer på hver side av pivot er en $O(n)$ prosess



Gitt en liste [4, 2, 1, 7, 9, 43, 34, 70, 8] som har partition utført på seg, hva er pivot elementet her?

16 ×



9

3 ×



34

8 ✓



7

1 ×



43



28

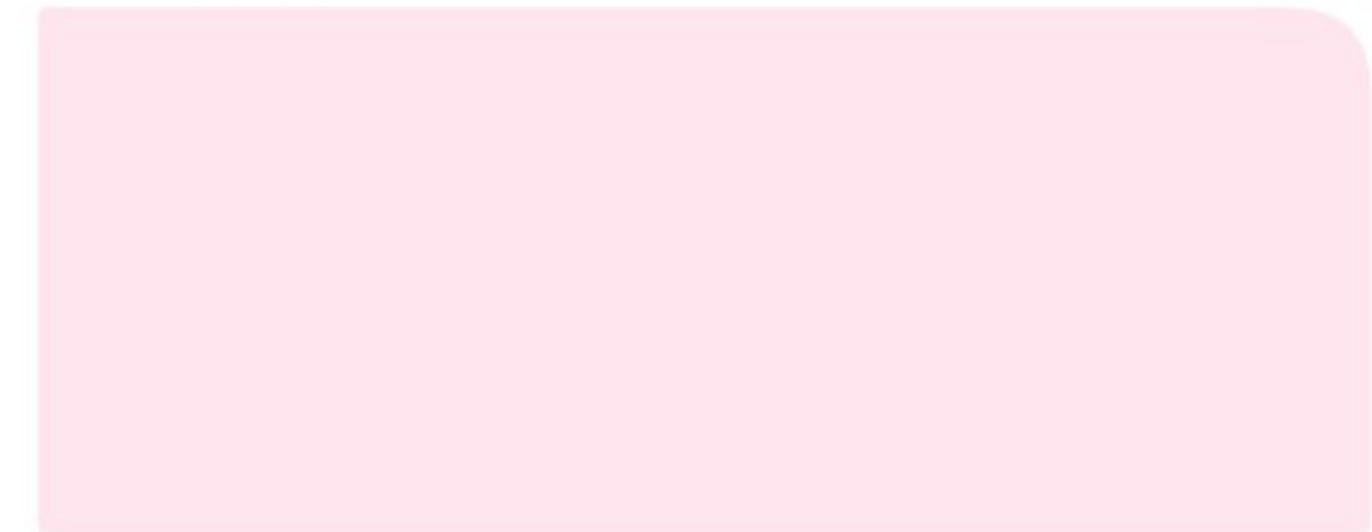
Er QuickSort fra pensum in-place?

22 ✓



Ja

9 ✗



Nei



That's it folks!

Forslag til oppgaver:

- Prøve å lage mergesort in-place
- Gjøre kattis oppgaver om sortering (Finner dem på Gruppe6 ressurser på emne siden)
- Gjøre en oppgave fra README-en fra Gruppe6 ressursene