Sortering



IN2010 – Uke 3

Icebreaker (3 min)

Hva er favorittemnet ditt du har tatt på ifi/?

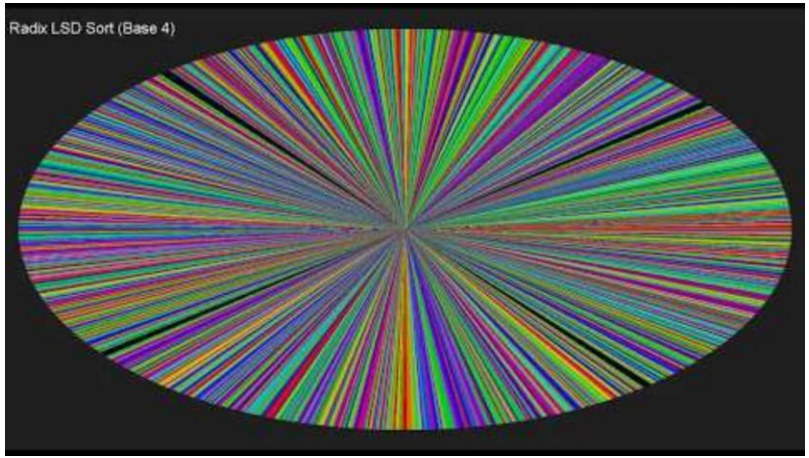
Eller: Katter eller hunder?

Fire sorteringsalgoritmer



Ukas video

Sorteringsalgoritmer visualisert som en fargesirkel



https://youtu.be/y9Ecb43qw98

Bubble Sort

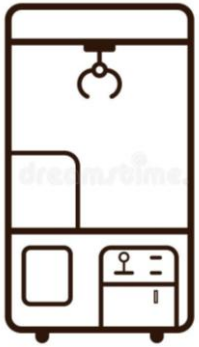
- Sjekker ut to og to tall av gangen (j-løkken)
  - Dersom tall 1 (A[j]) er større enn tall 2 (A[j+1]): Switch!
- Det største tallet 'bobler opp' til enden av lista
- i-løkken har oversikt over hvor mange tall på enden av lista som er 'riktig plassert'

ALGORITHM: BUBBLE SORT

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

```
1 Procedure BubbleSort(A)
2     for i ← 0 to n − 2 do
3         for j ← 0 to n − i − 2 do
4             if A[j] > A[j + 1] then
5                 A[j], A[j + 1] ← A[j + 1], A[j]
```

Selection Sort



- j-løkken leter gjennom arrayet etter det minste tallet (som k holder styr på)
- Deretter flytter den det minste tallet til bakerste posisjon (i-løkken)

ALGORITHM: SELECTION SORT

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

```
1 Procedure SelectionSort(A)
2     for i ← 0 to n − 1 do
3         k ← i
4         for j ← i + 1 to n − 1 do
5             if A[j] < A[k] then
6                 k ← j
7         if i ≠ k then
8             A[i], A[k] ← A[k], A[i]
```

Insertion Sort

- j-løkka sjekker par med tall for å se om tall 1 (A[j-1]) er større enn tall 2 (A[j]) : I så fall ‘sender’ den A[j] fremover helt til det ikke er noe større enn den.
- i-løkken holder styr på hvor i arrayet vi er ‘ferdig’

ALGORITHM: INSERTION SORT

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

```
1 Procedure InsertionSort(A)
2     for i ← 1 to n − 1 do
3         j ← i
4         while j > 0 and A[j − 1] > A[j] do
5             A[j − 1], A[j] ← A[j], A[j − 1]
6             j ← j − 1
```

Merge Sort

- Består av to deler:
  - MergeSort(A) som deler opp i nye, mindre arrayer
  - Merge(A1,A2,A) som fletter sammen to arrayer

ALGORITHM: MERGE SORT

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

```
1 Procedure MergeSort (A)
2     if n ≤ 1 then
3         return A
4     i ← ⌊n/2⌋
5     A1 ← MergeSort(A[0..i-1])
6     A2 ← MergeSort(A[i..n-1])
7     return Merge(A1, A2, A)
```

Merge Sort

ALGORITHM: SORTERT FLETTING AV TO ARRAYER

Input: To sorterte arrayer A_1 og A_2 og et array A, der $|A_1| + |A_2| = |A| = n$
Output: Et sortert array A med elementene fra A_1 og A_2

```
1  Procedure Merge (A1, A2, A)
2      i ← 0
3      j ← 0
4      while i < |A1| and j < |A2| do
5          if A1[i] ≤ A2[j] then
6              A[i + j] ← A1[i]
7              i ← i + 1
8          else
9              A[i + j] ← A2[j]
10             j ← j + 1
11     while i < |A1| do
12         A[i + j] ← A1[i]
13         i ← i + 1
14     while j < |A2| do
15         A[i + j] ← A2[j]
16         j ← j + 1
17     return A
```

Merge(A1,A2,A) er i O(n)

Dybden på rekursjonen i MergeSort(A) er O(log(n)) (Fordi den halveres for hver gang)

Altså gjør vi Merge O(log(n)) ganger og kjøretiden blir O(n*log(n))

Merge Sort

ALGORITHM: MERGE SORT

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

```
Procedure MergeSort (A)
    if n ≤ 1 then
        return A
    i ← ⌊n/2⌋
    A1 ← MergeSort(A[0..i-1])
    A2 ← MergeSort(A[i..n-1])
    return Merge(A1, A2, A)
```

ALGORITHM: SORTERT FLETTING AV TO ARRAYER

Input: To sorterte arrayer A_1 og A_2 og et array A, der $|A_1| + |A_2| = |A| = n$
Output: Et sortert array A med elementene fra A_1 og A_2

```
1  Procedure Merge (A1, A2, A)
2      i ← 0
3      j ← 0
4      while i < |A1| and j < |A2| do
5          if A1[i] ≤ A2[j] then
6              A[i + j] ← A1[i]
7              i ← i + 1
8          else
9              A[i + j] ← A2[j]
10             j ← j + 1
11     while i < |A1| do
12         A[i + j] ← A1[i]
13         i ← i + 1
14     while j < |A2| do
15         A[i + j] ← A2[j]
16         j ← j + 1
17     return A
```

Stabilitet og In-place

- Stabilitet: Dersom **x** (<- Et tall) forekommer før **y** (<- ett annet tall) FØR sortering og x=y, skal **x** også forekomme før **y** ETTER sortering.
  - Selection sort er ikke stabil
- In-place: Bruker ikke ekstra datastrukturer.

Oppgaver + logaritme repetisjon