Mikkels Magiske Tal Sorteringer

Sorteringsalgoritmer

Steffen & Mads

Introduktion

Velkommen til Mikkels Magiske Tal Sorteringer.
Her har vi gjort brug af forskellige sorteringsalgoritmer, og lavet statistikker for de eksekveringer vi har kørt.

Vi håber at disse grafiske inputs kan være med til at forøge glæden i at se resultaterne.

Selection Sort:

Sortere et array af tal i stigende række følge, med laveste først, så f.eks. [3, 1, 9, 8, 4] bliver til [1, 3, 4, 8, 9].

Forskellen på Mikkels version og ChatGpt version:

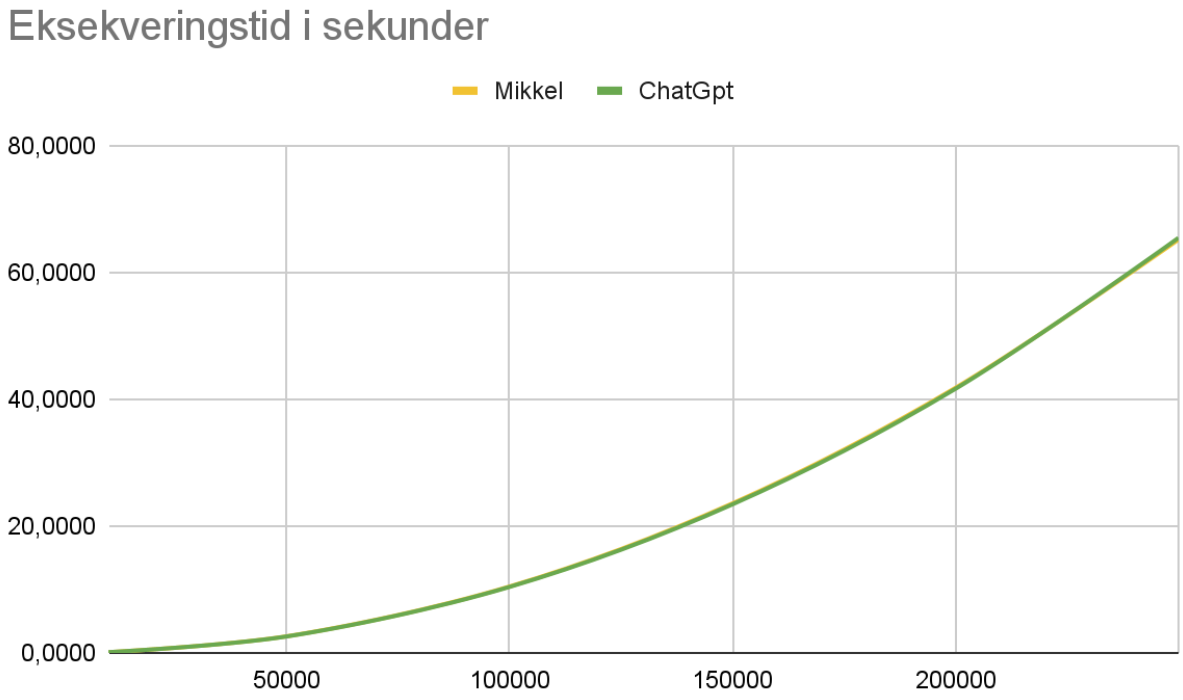
Selvom begge algoritmer er sorteringsalgoritmer og har samme mål - at arrangere tal i stigende rækkefølge - har de en forskellig tilgang og implementering.

Mikkels algoritme bruger pointeraritmetik og funktionsopkald (swap) for at sortere arrayet. Den arbejder ved gentagne gange at finde det mindste element i den ubehandlede del af arrayet og bytte det med det første ubehandlede element. Dette gentages, indtil hele arrayet er sorteret. Den Mikkels algoritme arbejder med en pointer til arrayelementerne.

ChatGPT's algoritme er mere direkte i sin tilgang og bruger en simpel variabel til at spore indekset for det mindste element. Den starter med at antage, at det første element i den ubehandlede del er det mindste, og søger derefter efter et mindre element i resten af arrayet. Hvis et mindre element findes, byttes det med det aktuelle element. ChatGPT's algoritme arbejder med arrayindeks i stedet for pointeraritmetik.

Selvom begge algoritmer udfører det samme grundlæggende trin med at finde det mindste element og bytte det med det aktuelle element, har de forskellige måder at opnå dette på. Mikkels algoritme bruger pointer og funktionsopkald, mens ChatGPT's bruger almindelige variable og indeks. Begge algoritmer er gyldige og giver det samme resultat, men de er implementeret på forskellige måder.

Array længde	Mikkels Execution Time In seconds	ChatGPTs Execution Time In seconds
10000	0,105000	0,109000
50000	2,618000	2,592000
100000	10,475000	10,406000
150000	23,575000	23,427000
200000	41,792000	41,638000
250000	65,149000	65,423000



Insertion Sort

Indsættelsessortering er en enkel og effektiv sorteringsalgoritme, der arbejder ved at opdele en given liste i to dele: en sorteret del og en usorteret del. Algoritmen starter med at betragte det første element som den sorterende del og de resterende elementer som den usorterede del. Derefter tages hvert element fra den usorterede del og placeres på sin korrekte position i den sorterende del. Dette gentages, indtil hele listen er sorteret.

For hvert element i den usorterede del sammenlignes det med elementerne i den sorterende del, indtil den korrekte position findes. Dette gøres ved at sammenligne elementet med de tidligere elementer i den sorterende del og flytte de større elementer en plads til højre for at give plads til det nye element. Denne proces fortsætter, indtil alle elementer er placeret korrekt, og listen er fuldt sorteret.

Indsættelsessortering er velegnet til små lister eller lister, der allerede er delvist sorteret, da den har en lineær tidskompleksitet i bedste tilfælde, men den kan blive langsom på store lister med omvendt sortering.

Forskellen mellem InsertionSort og SelectionSort

InsertSort og SelectionSort (Udvalgssortering) er to forskellige sorteringsalgoritmer med forskellige tilgange til at arrangere elementer. Her er nogle vigtige forskelle mellem dem:

Sorteringsmetode:

InsertSort: InsertSort arbejder ved at sammenligne hvert element med de tidligere elementer og placere det i den korrekte position i den sorterende del.

SelectionSort : SelectionSort arbejder ved gentagne gange at finde det mindste element i den usorterede del og bytte det med det første element i den usorterede del.

Arbejdstilgang:

InsertSort : InsertSort opbygger gradvist en sorteret del af listen ved at indsætte elementer fra den usorterede del i den korrekte rækkefølge.

SelectionSort : SelectionSort opbygger gradvist en sorteret del ved at vælge det mindste element fra den usorterede del og bytte det med det første usorterede element.

Bedste og værste tilfælde:

InsertSort : InsertSort har en bedste tidskompleksitet på O(n) i tilfælde af en allerede delvist sorteret liste og en værste tidskompleksitet på O(n^2).

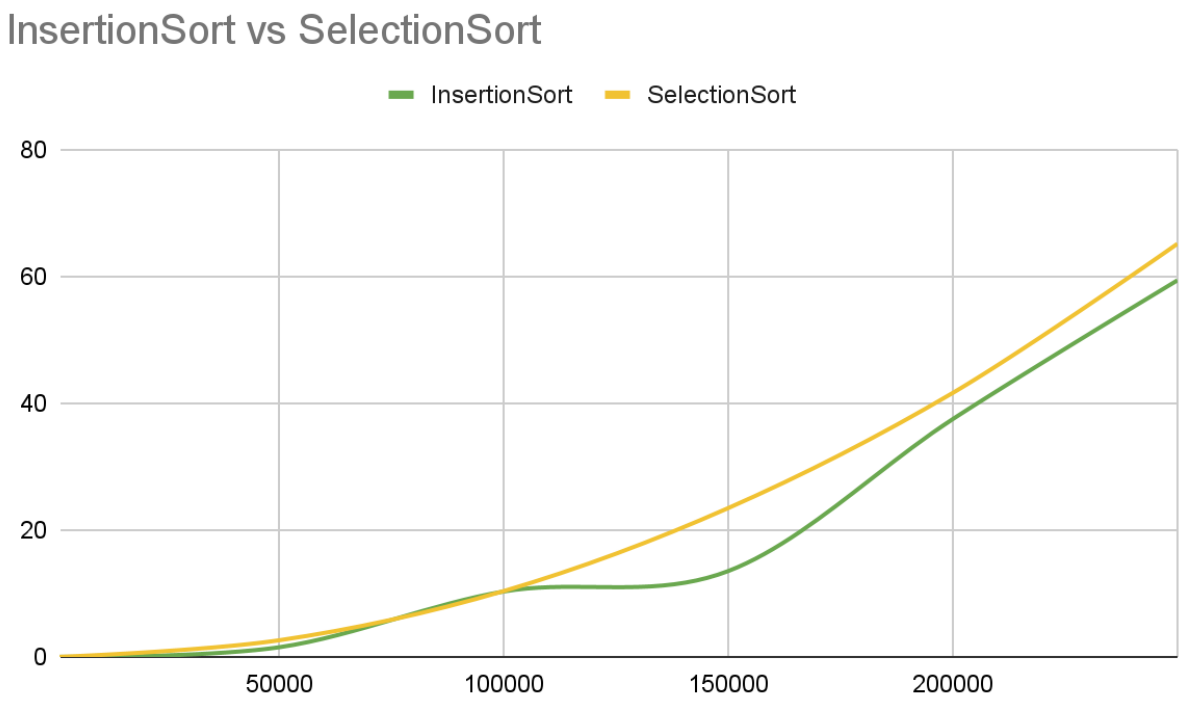
SelectionSort : SelectionSort har altid en tidskompleksitet på O(n^2), uanset inputdata.

Praktisk anvendelse:

InsertSort : InsertSort er velegnet til små lister eller lister, der allerede er delvist sorteret.

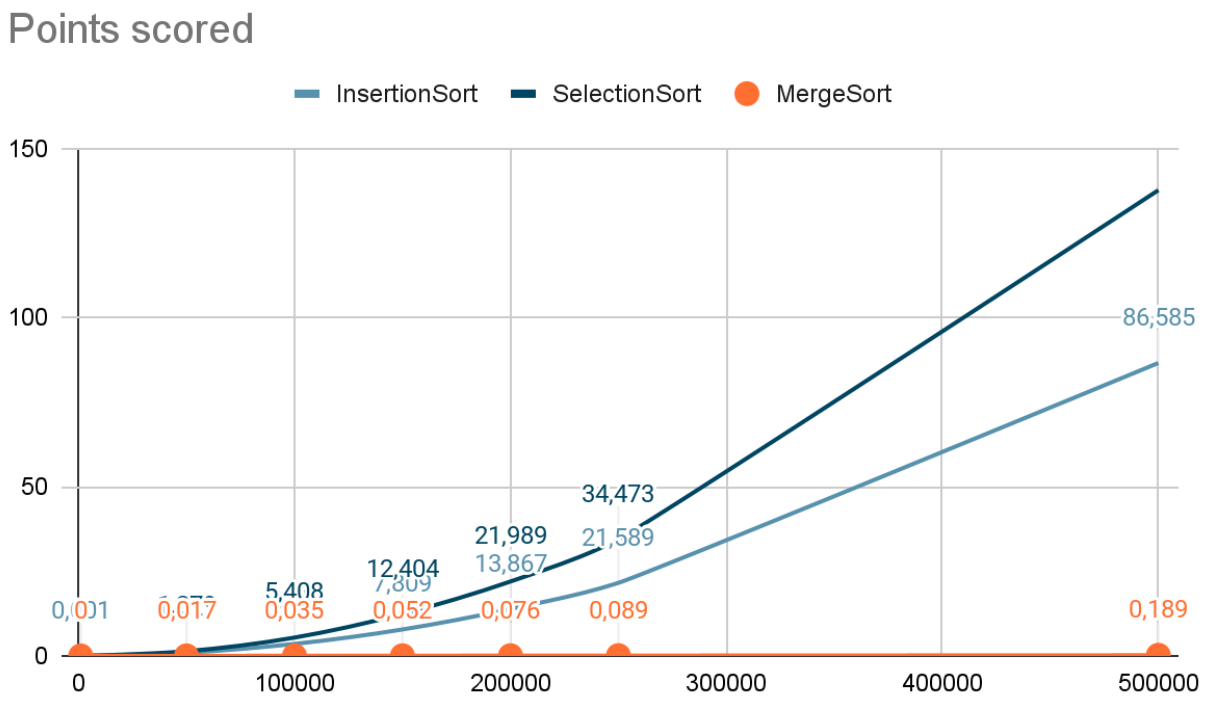
SelectionSort : SelectionSort kan være mere effektiv end InsertSort i nogle tilfælde, men den er generelt mindre effektiv end mere avancerede sorteringsalgoritmer som f.eks. quicksort eller mergesort.

Array længde	InsertionSort	SelectionSort
1000	0,000000	0,002000
50000	1,528000	2,629000
100000	10,318000	10,391000
150000	13,546000	23,479000
200000	37,509000	41,633000
250000	59,366000	65,126000



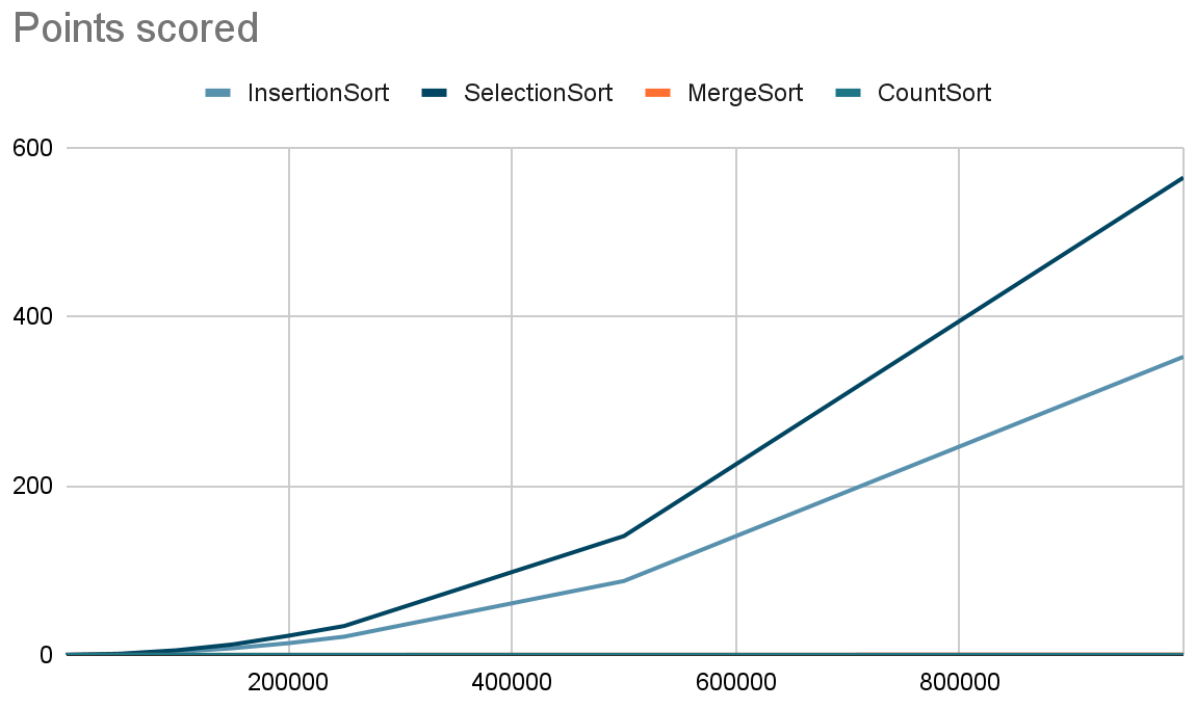
Merge Sort

Array længde	InsertionSort	SelectionSort	MergeSort
1000	0,001000	0,000000	0,000000
50000	0,928000	1,379000	0,017000
100000	3,541000	5,408000	0,035000
150000	7,809000	12,404000	0,052000
200000	13,867000	21,989000	0,076000
250000	21,589000	34,473000	0,089000
500000	86,585000	137,669000	0,189000



Count Sort

Array længde	InsertionSort	SelectionSort	MergeSort	CountSort
1000	0,001000	0,000000	0,001000	0,000000
50000	0,965000	1,381000	0,017000	0,000000
100000	3,505000	5,409000	0,033000	0,001000
150000	7,815000	12,284000	0,055000	0,001000
200000	13,817000	22,605000	0,092000	0,002000
250000	21,532000	34,138000	0,084000	0,002000
500000	87,373000	140,532000	0,190000	0,003000
1000000	352,444000	564,402000	0,361000	0,006000



Merge vs Counter sort

Array længde	MergeSort	CountSort
1000	0,001000	0,000000
50000	0,017000	0,000000
100000	0,033000	0,001000
150000	0,055000	0,001000
200000	0,092000	0,002000
250000	0,084000	0,002000
500000	0,190000	0,003000
1000000	0,361000	0,006000