DV1625 Rapport Inlämningsuppgift A

Abdalrahman Mohammed, DVAMI22h

18 november 2023

1 Introduktion

Algoritmer utgör kärnan i att hantera komplexa problem och effektivt bearbeta samt analysera data inom olika tillämpningsområden. Den här rapporten inriktar sig på att implementera och testa tre olika sorteringsalgoritmer med syfte att utforska deras körtidsresultat. Genom att analysera dessa resultat kommer brytningspunkterna där algoritmerna skiftar i prestanda att identifieras.
Utifrån dessa brytningspunkter kommer två hybridsorteringsalgoritmer att konstrueras. Rapporten strävar slutligen efter att etablera en övergripande bevisning för att de utvecklade hybridsorteringsalgoritmerna generellt sett har snabbare körtider jämfört med originalalgoritmerna.

2 Bakgrund

Sortering syftar till att ordna element i en given sekvens. Ofta tillämpas sortering för att underlätta effektiv sökning, filtrering och åtkomst till data. Det finns olika sorteringsalgoritmer, där var och en har sina egna unika egenskaper och effektivitet i olika scenarier.

2.1 Insertionsort

Enligt information hämtad från kurslitteraturen (Thomas H Cormen, 2022, pp. 17-25) så bygger Insertionsort upp den slutliga sorterade sekvensen genom att successivt ta in element från den oordnade delen av listan och placera dem på rätt plats i den sorterade delen. Om ett element är mindre än eller lika med ett element i den sorterade delen, byts det ut med det element som finns i den nuvarande positionen. Algoritmen går igenom listan från början till slut och upprepar denna process tills hela listan är sorterad.
Tidskomplexitet för insertionsort i bästa fall är O(n) och detta sker när sekvensen redan är sorterad i stigande ordning. I medel fallet är Tidskomplexiteten $O(n^2)$ och detta sker när sekvensen är i omvänd ordning. I värsta fallet är Tidskomplexiteten $O(n^2)$ och detta sker när Sekvensen är slumpmässigt fördelade.
Fördelarna med denna algoritm inkluderar enkel implementation, effektivitet för små listor, stabil och adaptiv.
Nackdelarna med denna algoritm omfattar kvadratisk tidskomplexitet i genomsnittlig och värsta fall samt att den inte är optimerad för stora data set.

2.2 Quicksort

Enligt information hämtad från kurslitteraturen (Thomas H Cormen, 2022, pp. 182-190). Så väljer Quicksort ett pivotelement och partitionerar resten av elementen i två delar. Element mindre än pivoten placeras till vänster om den, medan de större placeras till höger. Algoritmen tillämpar sedan quicksort rekursivt på dessa två delar.
Tidskomplexitet för Quicksort i bästa fall är $O(n\log_2(n))$ och detta sker när partitioneringen alltid delar arrayen i två lika stora delar. I medel fallet är Tidskomplexiteten $O(n\log_2(n))$ och detta sker för att partitioneringen sannolikt kommer att variera och inte alltid leda till de extrema fallen som i det sämsta fallet. I värsta fallet är Tidskomplexiteten $O(n^2)$ och detta sker när partitioneringen alltid leder till att en del av arrayen är tom och en annan del innehåller alla elementen.
Fördelarna med denna algoritm inkluderar snabb prestanda på stora data set, in-place sortering och adaptiv.
Nackdelarna med denna algoritm omfattar att den inte är stabil, känslighet för pivotval och potentiell kvadratisk tidskomplexitet i värsta fall.

2.3 Mergesort

Enligt information hämtad från kurslitteraturen (Thomas H Cormen, 2022, pp. 34-43) så delar Mergesort upp listan i två lika stora delar tills varje lista har ett element. Därefter sorterar den varje del separat och kombinerar dem sedan i en stegvis ordnad sekvens.
Tidskomplexitet för mergesort i bästa fall, medel fall, värsta fall är $O(n\log_2(n))$ för att mergesort använder en konsekvent uppdelnings- och sammanslagningsstrategi som inte påverkas av initiala dataordningen, vilket resulterar i förutsägbar prestanda.
Fördelarna med denna algoritm inkluderar stabilitet, garanterad prestanda och effektivitet för länkade listor.
Nackdelarna med denna algoritm omfattar behovet av extra lagringsutrymme, en konstant faktor som kan påverka prestanda för små data set och brist på anpassning till delvis sorterade listor.

3 Metod och implementationsval

När insertionsort, quicksort och mergesort implementerades användes pseudokod från kurslitteraturen (Thomas H Cormen, 2022, pp. 30, 36, 39,183,184).
Quicksort implementerades med Lomuto-partitioneringen som partitioneringsmetod på grund av dess enkelhet och tydlighet vid implementation
Mergesort implementerades med top-down-ansatsen, vilket valdes för att underlätta förståelsen.
När c-konstanten beräknades används formeln $T(n) = O(f(n)) \rightarrow T(n) \leq c \cdot f(n)$, där vi antog medelfallet för respektive algoritm. Medelfallet för insertionsort är n^2 och medelfallet för quicksort och mergesort är $n\log_2(n)$.

4 Resultat och Analys

4.1 Sorteringsalgoritmer

Tabell 1 visar att resultatet för medelvärdet överensstämmer med förväntningarna, då insertionsort presterar snabbare än både quicksort och mergesort. Detta kan förklaras av att datafilen är liten och att insertionsort vanligtvis visar sig vara mer effektiv för mindre filstorlekar. I den första körtiden för insertionsort i Tabell 1 noterar vi en betydande ökning av körtiden, vilket kan härledas till att körtiden blir kvadratisk, vilket representerar det värsta fallet för insertionsort.
I Tabell 2 observerar vi en ökning av medelvärdet för insertionsort, vilket överstiger både quicksort och mergesort. Detta fenomen kan härledas till att filstorleken i Tabell 2 är större. Detta resultat är förväntat, eftersom mergesort och quicksort generellt sett är mer effektiva för hantering av större filer.
Senare i resten av tabellerna observerar vi en betydande ökning av körtiden för insertionsort, vilken är markant högre jämfört med mergesort och quicksort. Detta beror på att filstorlekarna i de återstående tabellerna bara fortsätter att växa. Dessutom observerar vi att quicksort presterar något bättre än mergesort. Det kan förklaras av att quicksort troligtvis har en mindre konstant faktor jämfört med mergesort.

Algoritm	Körtid (s/ms/µs)					
	1	2	3	4	5	Medelvärde
Insertionsort	5.400µs	3.000µs	2.800µs	2.700µs	2.700µs	3.320µs
Quicksort	12.500µs	5.300µs	5.000µs	4.800µs	4.800µs	6.480µs
Mergesort	15.300µs	9.000µs	8.800µs	8.800µs	8.700µs	10.120µs

Tabell 1, Sorteringsalgoritmernas körtid och medelvärde för filerna på storlek 10.

Algoritm	Körtid (s/ms/µs)					
	1	2	3	4	5	Medelvärde
Insertionsort	0.168ms	0.164ms	0.164ms	0.338ms	0.371ms	0.241ms
Quicksort	87.780µs	64.100µs	64.000µs	63.700µs	63.500µs	68.620µs
Mergesort	0.142ms	0.135ms	0.136ms	0.136ms	0.151ms	0.140ms

Tabell 2, Sorteringsalgoritmernas körtid och medelvärde för filerna på storlek 100.

Algoritm	Körtid (s/ms/µs)					
	1	2	3	4	5	Medelvärde
Insertionsort	19.218ms	19.231ms	20.141ms	19.657ms	19.097ms	19.469ms
Quicksort	1.082ms	1.041ms	1.030ms	1.062ms	1.029ms	1.049ms
Mergesort	3.227ms	1.662ms	1.651ms	1.624ms	1.634ms	1.960ms

Tabell 3, Sorteringsalgoritmernas körtid och medelvärde för filerna på storlek 1000.

Algoritm	Körtid (s/ms/µs)					
	1	2	3	4	5	Medelvärde
Insertionsort	1.944s	1.938s	1.942s	1.973s	1.976s	1.954s
Quicksort	12.673ms	12.718ms	12.668ms	14.512ms	12.358ms	12.986ms
Mergesort	22.661ms	22.791ms	22.372ms	23.199ms	23.021ms	22.809ms

Tabell 4, Sorteringsalgoritmernas körtid och medelvärde för filerna på storlek 10 000.

Algoritm	Körtid (s/ms/µs)					
	1	2	3	4	5	Medelvärde
Insertionsort	259.434s	264.549s	255.830s	265.303s	241.899s	257.403s
Quicksort	0.184s	0.209s	0.190s	0.192s	0.198s	0.195s
Mergesort	0.323s	0.331s	0.308s	0.302s	0.303s	0.314s

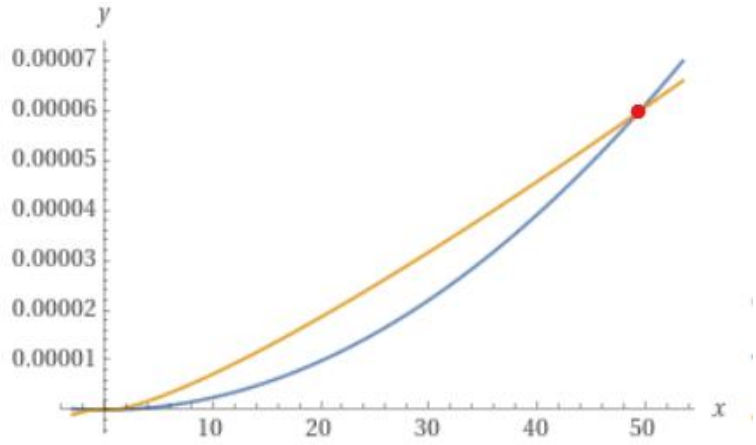
Tabell 5, Sorteringsalgoritmernas körtid och medelvärde för filerna på storlek 100 000.

Algoritm	c-konstanter					
	c_{10}	c_{100}	c_{1000}	c_{10000}	c_{100000}	$\bar{c}$
Insertionsort	$3.320 \cdot 10^{-8}$	$2.410 \cdot 10^{-8}$	$1.947 \cdot 10^{-8}$	$1.954 \cdot 10^{-8}$	$2.574 \cdot 10^{-8}$	$2.441 \cdot 10^{-8}$
Quicksort	$1.951 \cdot 10^{-7}$	$1.033 \cdot 10^{-7}$	$1.053 \cdot 10^{-7}$	$9.772 \cdot 10^{-8}$	$1.174 \cdot 10^{-7}$	$1.238 \cdot 10^{-7}$
Mergesort	$3.046 \cdot 10^{-7}$	$2.107 \cdot 10^{-7}$	$1.967 \cdot 10^{-7}$	$1.717 \cdot 10^{-7}$	$1.890 \cdot 10^{-7}$	$2.145 \cdot 10^{-7}$

Tabell 6, c-konstant för respektive sorteringsalgoritm samt medelvärdet av c-konstanterna, d.v.s. $\bar{c}$.

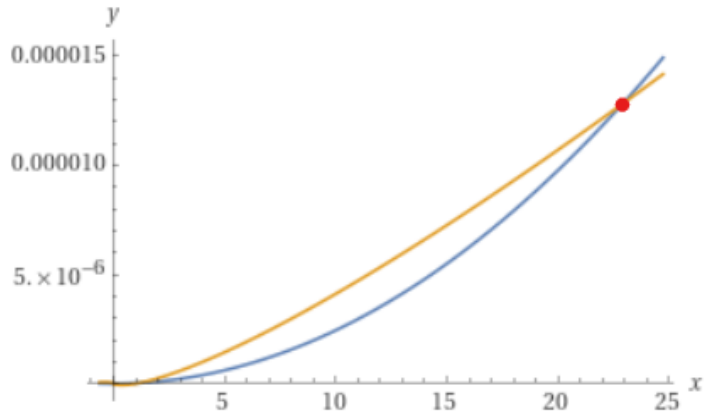
4.2 Identifiering av brytpunkter

Brytpunkterna mellan insertionsort samt både mergesort och quicksort identifierades med hjälp av ett grafverktyg som beräknade var kurvorna korsade varandra. Formeln som används $T(n) = O(f(n)) \rightarrow T(n) \leq \bar{c} \cdot f(n)$, där vi antog medelfallet för respektive algoritm (x = antal element, y = Körtid).



$x = 49.456$ and $y = 0.0000597044$

Insertionsort
Mergesort



$x = 22.915$ and $y = 0.0000128177$

Insertionsort
Quicksort

4.3 Hybridsorteringsalgoritmer

I Tabell 7 märker vi att hybridsorteringen med quicksort uppvisar ett högre medelvärde för körtiden när det gäller datafilen med 10 element. Detta kan möjligen härledas till att den första körningen på datafilen med 10 element resulterade i en ovanligt hög körtid, möjligen på grund av att värsta fallet för insertionsort inträffade.
Å andra sidan observerar vi en minskning av medelvärdet för körtiden i övriga datafiler som ökar i storlek, jämfört med hybridsorteringen med mergesort. Denna förändring kan förklaras av att hybridsorteringen med quicksort förmodligen har en mindre konstant faktor jämfört med hybridsorteringen med mergesort.

Filstorlek	Algoritm	Körtid (s/ms/µs)					
		1	2	3	4	5	Medelvärde
10	$Hybrid_Q$	7.900µs	3.300µs	3.000µs	2.900µs	2.900µs	4.000µs
	$Hybrid_M$	5.700µs	3.400µs	3.100µs	2.800µs	2.800µs	3.560µs
100	$Hybrid_Q$	60.800µs	55.900µs	52.800µs	52.500µs	52.700µs	54.940µs
	$Hybrid_M$	$0.112ms$	$0.105ms$	$0.104ms$	$0.104ms$	$0.103ms$	$0.106ms$
1000	$Hybrid_Q$	$0.958ms$	$0.955ms$	$0.960ms$	$0.957ms$	$0.983ms$	$0.963ms$
	$Hybrid_M$	$1.998ms$	$1.278ms$	$1.298ms$	$1.299ms$	$1.306ms$	$1.436ms$
10 000	$Hybrid_Q$	$11.562ms$	$11.689ms$	$11.704ms$	$11.819ms$	$11.885ms$	$11.732ms$
	$Hybrid_M$	$18.691ms$	$19.201ms$	$19.523ms$	$18.828ms$	$18.981ms$	$19.045ms$
100 000	$Hybrid_Q$	$0.170s$	$0.165s$	$0.156s$	$0.162s$	$0.169s$	$0.164s$
	$Hybrid_M$	$0.270s$	$0.261s$	$0.268s$	$0.264s$	$0.291s$	$0.271s$

Tabell 7, Körtid och medelvärde för hybridsorteringsalgoritmerna.

5 Slutsatser

Till slut kan man se tydligt att hybridsorteringsalgoritmerna är snabbare än standard sorteringsalgoritmerna och visar på fördelarna med att kombinera olika sorteringsmetoder för att optimera prestanda. Genom att välja lämpliga brytpunkter för övergången mellan insertionsort och antingen mergesort eller quicksort kan hybridsorteringsalgoritmerna dra nytta av de specifika styrkorna hos varje enskild algoritm. Detta resulterar i att de skapade hybrider är mer anpassade för olika scenarier och data set.

6 Bibliografi

Thomas H Cormen, C. E. (2022). *Introduction to Algorithms.* : MIT press.

(Thomas H Cormen, 2022)