Testplan: sorteringsalgoritmer

|  |  |
| --- | --- |
| Datum: | 2025-05-22 |
| Testare: | Isak Lagerberg, Mattias Arvidsson & Joakim Sandström |
| Omfattning: | * Testning av sorteringsalgortimerna bubblesort, insertionsort, mergesort, quicksort, heapsort samt Selectionsort. * Prestandamätningar på dataset med varierande storlek (100 till 1 000 000 poster). * Analys av körtid. |
| Teststrategi: | Testningen utförs manuellt med hjälp av C#-programmet som utvecklats för laborationen. För varje algoritm kommer testfall skapas för att mäta körtid vid olika datamängder. Resultaten kommer dokumenteras och analyseras för att identifiera mönster och avvikelser. |
| Testmiljö: | * Operativsystem: Windows 11 Home ver. 24H2 * Utvecklingsmiljö: Visual Studio Enterprise 2022 (64-bit) ver. 17.13.6 * Programmeringsspråk: C#, .NET 9.0 |
| Acceptanskriterier: | * Algoritmerna ska sortera korrekt enligt specifikation. * Körtiden ska vara inom rimliga gränser för varje algoritm och datamängd. |
| Risker: | * Felaktig implementering av algoritmer kan leda till inkorrekta resultat. * Stora datamängder kan orsaka minnesbrist eller lång körtid. * Manuell testning kan vara tidskrävande och innefattar risk för mänskliga fel. |
| Tidsplan: | Vecka 1: implementering av sorteringalgoritmer och utförande av testfall för sortering. Vecka 2: analys och dokumentation av resultat. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test no.** | **Testnamn** | **Syfte med test** | **Testdata eller testsituation** | **Resultat** | **Kommentar** |
| 1 | *Bubblesort med slumpmässig data* | Testa algoritmen bubblesorts prestanda med slumpmässig inputdata | |  |  |  |  | | --- | --- | --- | --- | | **Algoritm:** Bubble sort | | | | | **Testfall** | **Element** | **Seed** | **Upprepningar** | | 1 | 100 | 123 | 100 | | 2 | 1000 | 123 | 100 | | 3 | 10 000 | 123 | 100 | | 4 | 100 000 | 123 | 100 | | 5 | 1 000 000 | 123 | 100 | | |  |  | | --- | --- | | **Testfall** | **Resultat (ms)** | | 1 | 0,00 | | 2 | 8,45 | | 3 | 871,52 | | 4 | 198306,05 | | 5 | N/A | | BubbleSort har mycket hög körtid på osorterad data. Eftersom algoritmen har komplexitet O(n²), växer tiden snabbt vid större datamängder. Körningen blev orealistiskt lång redan vid 100 000 element och omöjlig vid 1 000 000. |
| 2 | *Mergesort med slumpmässig data* | Testa algoritmen mergesort prestanda med slumpmässig inputdata | |  |  |  |  | | --- | --- | --- | --- | | **Algoritm:** Merge sort | | | | | **Testfall** | **Element** | **Seed** | **Upprepningar** | | 1 | 100 | 123 | 100 | | 2 | 1000 | 123 | 100 | | 3 | 10 000 | 123 | 100 | | 4 | 100 000 | 123 | 100 | | 5 | 1 000 000 | 123 | 100 | | |  |  | | --- | --- | | **Testfall** | **Resultat (ms)** | | 1 | 0,00 | | 2 | 0,00 | | 3 | 1,14 | | 4 | 27,24 | | 5 | 275,04 | | MergeSort är mycket effektiv även på stora mängder osorterad data. Tack vare sin rekursiva "divide and conquer"-strategi och O(n log n)-komplexitet är prestandan stabil oavsett indata. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test no.** | **Testnamn** | **Syfte med test** | **Testdata eller testsituation** | **Resultat** | **Kommentar** |
| **2** | *Insertion sort med slumpmässig data* | Testa algoritmen insertion sorts prestanda med slumpmässig inputdata | |  |  |  |  | | --- | --- | --- | --- | | **Algoritm**: Insertion sort | | | | | **Testfall** | **Element** | **Seed** | **Upprepningar** | | 1 | 100 | 123 | 100 | | 2 | 1000 | 123 | 100 | | 3 | 10 000 | 123 | 100 | | 4 | 100 000 | 123 | 100 | | 5 | 1 000 000 | 123 | 100 | | |  |  | | --- | --- | | **Testfall** | **Resultat (ms)** | | 1 | 0,00 | | 2 | 1,06 | | 3 | 156,67 | | 4 | 17 941 ,43 | | 5 | N/A | | InsertionSort presterar dåligt på stora, osorterade datamängder. Algoritmen är enkel att implementera men ineffektiv för stora N på grund av O(n²) i värsta fall. Den kraschar eller blir orimligt långsam vid 1 000 000 poster. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test no.** | **Testnamn** | **Syfte med test** | **Testdata eller testsituation** | **Resultat** | **Kommentar** |
| **3** | *Selection sort med slumpmässig data* | Testa algoritmen selection sorts prestanda med slumpmässig inputdata | |  |  |  |  | | --- | --- | --- | --- | | **Algoritm**: Selection sort | | | | | **Testfall** | **Element** | **Seed** | **Upprepningar** | | 1 | 100 | 123 | 100 | | 2 | 1000 | 123 | 100 | | 3 | 10 000 | 123 | 100 | | 4 | 100 000 | 123 | 100 | | 5 | 1 000 000 | 123 | 100 | | |  |  | | --- | --- | | **Testfall** | **Resultat (ms)** | | 1 | 0,00 | | 2 | 3,01 | | 3 | 320,97 | | 4 | 35704,57 | | 5 | N/A | | SelectionSort har en konstant mönsterbeteende, alltid O(n²) oavsett indata. Det gör den stabil i tid men dåligt skalbar – vid 100 000 element tar det lång tid, och vid 1 000 000 blev det ohanterligt. |
| **4** | *Heapsort med slumpmässig data* | Testa algoritmen heapsorts prestanda med slumpmässig inputdata | |  |  |  |  | | --- | --- | --- | --- | | **Algoritm**: Heap sort | | | | | **Testfall** | **Element** | **Seed** | **Upprepningar** | | 1 | 100 | 123 | 100 | | 2 | 1000 | 123 | 100 | | 3 | 10 000 | 123 | 100 | | 4 | 100 000 | 123 | 100 | | 5 | 1 000 000 | 123 | 100 | | |  |  | | --- | --- | | **Testfall** | **Resultat (ms)** | | 1 | 0,00 | | 2 | 0,00 | | 3 | 3,27 | | 4 | 41,62 | | 5 | 554,22 | | HeapSort ger bra prestanda på osorterad data. Den är konsekvent snabb även på större dataset tack vare O(n log n)-komplexitet, även om den är något långsammare än QuickSort i detta fall. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test no.** | **Testnamn** | **Syfte med test** | **Testdata eller testsituation** | **Resultat** | **Kommentar** |
| **5** | *Quicksort med slumpmässig data* | Testa algoritmen quicksorts prestanda med slumpmässig inputdata | |  |  |  |  | | --- | --- | --- | --- | | **Algoritm**: Quick sort | | | | | **Testfall** | **Element** | **Seed** | **Upprepningar** | | 1 | 100 | 123 | 100 | | 2 | 1000 | 123 | 100 | | 3 | 10 000 | 123 | 100 | | 4 | 100 000 | 123 | 100 | | 5 | 1 000 000 | 123 | 100 | | |  |  | | --- | --- | | **Testfall** | **Resultat (ms)** | | 1 | 0,00 | | 2 | 0,00 | | 3 | 1,42 | | 4 | 22,22 | | 5 | 316,07 | | QuickSort presterar utmärkt på slumpmässig data och skalar effektivt. Den har i genomsnitt O(n log n)-komplexitet och är ofta den snabbaste praktiska algoritmen i dessa testfall. |
| **6** | *Bubblesort med sorterad data* | Testa algoritmen bubblesorts prestanda med sorterad inputdata | |  |  |  |  | | --- | --- | --- | --- | | **Algoritm:** Bubble sort | | | | | **Testfall** | **Element** | **Seed** | **Upprepningar** | | 1 | 100 | 123 | 100 | | 2 | 1000 | 123 | 100 | | 3 | 10 000 | 123 | 100 | | 4 | 100 000 | 123 | 100 | | 5 | 1 000 000 | 123 | 100 | | |  |  | | --- | --- | | **Testfall** | **Resultat (ms)** | | 1 | 0,00 | | 2 | 0,00 | | 3 | 0,00 | | 4 | 0,01 | | 5 | 6,06 | | BubbleSort är väldigt effektiv på redan sorterad data – särskilt om implementationen bryter tidigt när inga byten sker. Den når nära O(n) och presterar mycket bättre än i slumpfallet. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test no.** | **Testnamn** | **Syfte med test** | **Testdata eller testsituation** | **Resultat** | **Kommentar** |
| **7** | *Mergesort med sorterad data* | Testa algoritmen mergesort prestanda med sorterad inputdata | |  |  |  |  | | --- | --- | --- | --- | | **Algoritm:** Merge sort | | | | | **Testfall** | **Element** | **Seed** | **Upprepningar** | | 1 | 100 | 123 | 100 | | 2 | 1000 | 123 | 100 | | 3 | 10 000 | 123 | 100 | | 4 | 100 000 | 123 | 100 | | 5 | 1 000 000 | 123 | 100 | | |  |  | | --- | --- | | **Testfall** | **Resultat (ms)** | | 1 | 0,00 | | 2 | 0,00 | | 3 | 0,67 | | 4 | 14,66 | | 5 | 161,70 | | MergeSorts prestanda påverkas knappt av om datan är sorterad eller inte, vilket visar hur robust algoritmen är. Tiderna är nästan identiska med osorterat fall. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test no.** | **Testnamn** | **Syfte med test** | **Testdata eller testsituation** | **Resultat** | **Kommentar** |
| **7** | *Insertion sort med sorterad data* | Testa algoritmen insertion sorts prestanda med sorterad inputdata | |  |  |  |  | | --- | --- | --- | --- | | **Algoritm**: Insertion sort | | | | | **Testfall** | **Element** | **Seed** | **Upprepningar** | | 1 | 100 | 123 | 100 | | 2 | 1000 | 123 | 100 | | 3 | 10 000 | 123 | 100 | | 4 | 100 000 | 123 | 100 | | 5 | 1 000 000 | 123 | 100 | | |  |  | | --- | --- | | **Testfall** | **Resultat (ms)** | | 1 | 0,00 | | 2 | 0,00 | | 3 | 0,00 | | 4 | 0,00 | | 5 | N/A | | InsertionSort presterar optimalt på redan sorterad data, med O(n) i bästa fall. Den går mycket snabbare än i det slumpmässiga fallet – även vid stora datamängder.  Konsolen kraschade vid körning utav 1 000 000 poster – p.g.a. stackoverflow. Algoritmens komplexitet O(n) gör att den inte är lämplig för så stora datamängder. |
| **8** | *Selection sort med sorterad data* | Testa algoritmen selection sorts prestanda med sorterad inputdata | |  |  |  |  | | --- | --- | --- | --- | | **Algoritm**: Selection sort | | | | | **Testfall** | **Element** | **Seed** | **Upprepningar** | | 1 | 100 | 123 | 100 | | 2 | 1000 | 123 | 100 | | 3 | 10 000 | 123 | 100 | | 4 | 100 000 | 123 | 100 | | 5 | 1 000 000 | 123 | 100 | | |  |  | | --- | --- | | **Testfall** | **Resultat (ms)** | | 1 | 0,00 | | 2 | 3,24 | | 3 | 345,62 | | 4 | 37098,55 | | 5 | N/A | | SelectionSort påverkas inte av att datan är försorterad. Algoritmen gör alltid lika många jämförelser och byten, vilket ger samma O(n²)-prestanda som med osorterad data. Vid 1 000 000 blev det ohanterligt. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test no.** | **Testnamn** | **Syfte med test** | **Testdata eller testsituation** | **Resultat** | **Kommentar** |
| **9** | *heapsort med sorterad data* | Testa algoritmen heapsorts prestanda med sorterad inputdata | |  |  |  |  | | --- | --- | --- | --- | | **Algoritm**: Heap sort | | | | | **Testfall** | **Element** | **Seed** | **Upprepningar** | | 1 | 100 | 123 | 100 | | 2 | 1000 | 123 | 100 | | 3 | 10 000 | 123 | 100 | | 4 | 100 000 | 123 | 100 | | 5 | 1 000 000 | 123 | 100 | | |  |  | | --- | --- | | **Testfall** | **Resultat (ms)** | | 1 | 0,00 | | 2 | 0,00 | | 3 | 2,92 | | 4 | 40,42 | | 5 | 748,84 | | HeapSorts prestanda förändras marginellt på sorterad data. Tiden förblir stabil tack vare algoritmens natur, men den är inte den snabbaste på redan sorterat. |
| **10** | *Quicksort med sorterad data* | Testa algoritmen quicksorts prestanda med sorterad inputdata | |  |  |  |  | | --- | --- | --- | --- | | **Algoritm**: Quick sort | | | | | **Testfall** | **Element** | **Seed** | **Upprepningar** | | 1 | 100 | 123 | 100 | | 2 | 1000 | 123 | 100 | | 3 | 10 000 | 123 | 100 | | 4 | 100 000 | 123 | 100 | | 5 | 1 000 000 | 123 | 100 | | |  |  | | --- | --- | | **Testfall** | **Resultat (ms)** | | 1 | 0,00 | | 2 | 5,47 | | 3 | 545,83 | | 4 | N/A | | 5 | N/A | | QuickSort riskerar att få sin sämsta fall-komplexitet O(n²) på sorterad data beroende på pivotval. Det märks i dina resultat där tiden snabbt ökar – och kraschar vid stora datamängder.  Konsolen kraschade vid körning utav 1 000 000 poster – p.g.a. stackoverflow. Algoritmens komplexitet O(n²) gör att den inte är lämplig för så stora datamängder. |