|  |  |  |
| --- | --- | --- |
| **PROJEKTOWANIE ALGORYTMÓW I METODY SZTUCZNEJ INTELIGENCJI** | | |
| Imię i nazwisko:  **Michał Filipowicz** | Nr albumu:  **226393** | Data:  **25.04.2017** |
| **Lista nr 5** | | |

1. Wprowadzenie  
     
   Głównym zadaniem do wykonania były testy implementacji poszczególnych sortowań dla różnych warunków początkowych.
2. Algorytmy oraz ich złożoność obliczeniowa  
     
   Zaimplementowane zostały trzy algorytmy (sześć wliczając bąbelkowe, introspektywne oraz dwa różne wybory pivota dla quicksort’a..) sortujące:  
     
   Quick Sort (Pol. Sortowanie szybkie) – wybierana jest oś podziału (tzw. Pivot). Często wybierany losowo lub poprzez określenie wartości środkowego elementu. Następnie kolejne elementy tablicy rozdzielane są do lewej lub prawej podtablicy zależnie od tego, czy są one większe od wybranego pivot’a.  
   W najgorszym przypadku złożoność obliczeniowa tego algorytmu to O(n2), oczekiwaną jest jednak złożoność (nlog(n)).  
     
   Merge Sort (Pol. Sortowanie przez scalanie) – tablica dzielona jest na dwie równe części rekurencyjnie, do czasu otrzymania elementu o najmniejszej (lub największej) wartości, następnie otrzymane części są „scalane” aż do otrzymania w pełni posortowanej tablicy.  
   Złożoność obliczeniowa algorytmu to (nlog2(n)).  
     
     
   Shell Sort (Pol. Sortowanie Shella) – polega na sortowaniu poprzez wstawianie elementów oddalonych o ustaloną liczbę miejsc (czyli tzw. Gap).  
   Złożoność algorytmu zależna jest od ciągu odstępów.
3. Wyniki eksperymentów  
     
   Przeprowadzony został szereg eksperymentów na zaimplementowanych i wymienionych powyżej algorytmach sortowania.  
   Dla każdego algorytmu ustalane były następujące warunki początkowe:  
     
   - Procent posortowania losowych tablic: 0, 25, 50, 75, 95, 99, 99.7.  
   - Ilość elementów tablicy: 10 000, 50 000, 100 000, 500 000, 1 000 000.  
   - Wykonanie algorytmu dla 1 lub 100 tablic.  
   - Wykonanie algorytmu dla odwrotnie posortowanej tablicy.  
     
   Na potrzeby eksperymentów w programie zaimplementowana została ręczna konfiguracja pozwalająca na ustalenie dowolnych z powyższych warunków początkowych.  
   Wszystkie pomiary wykonywane były WYŁĄCZNIE dla algorytmów sortowania, nie uwzględniały one czasu wykonania jakichkolwiek innych operacji.  
   Na potrzebę testów napisana została również funkcja sprawdzająca posortowanie pierwszej tablicy.  
     
   Wyniki pomiarów w millisekundach (pierw dla jednej, następnie dla stu tablic):  
     
     
     
     
   Quick Sort z losowym pivotem:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| QS | 0 | 25 | 50 | 75 | 95 | 99 | 99,7 | inverted |
| 10000 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1 |
| 50000 | 17 | 16 | 14 | 12 | 11 | 10 | 10 | 8 |
| 100000 | 39 | 36 | 30 | 24 | 22 | 21 | 21 | 17 |
| 500000 | 441 | 316 | 214 | 145 | 113 | 109 | 110 | 95 |
| 1000000 | 1473 | 961 | 582 | 336 | 228 | 227 | 222 | 191 |
|  | | | | | | | | |
| QS 100 | 0 | 25 | 50 | 75 | 95 | 99 | 99,7 | inverted |
| 10000 | 292 | 281 | 273 | 256 | 222 | 203 | 198 | 163 |
| 50000 | 1716 | 1649 | 1480 | 1269 | 1097 | 1051 | 1040 | 861 |
| 100000 | 3944 | 3584 | 3063 | 2545 | 2186 | 2118 | 2118 | 1784 |
| 500000 | 43524 | 31461 | 21335 | 14216 | 11275 | 10926 | 10995 | 9421 |
| 1000000 | 147168 | 96801 | 58023 | 32517 | 23006 | 22294 | 22208 | 19401 |

Quick Sort z pivotem po środku:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| QS | 0 | 25 | 50 | 75 | 95 | 99 | 99,7 | inverted |
| 10000 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 0 |
| 50000 | 15 | 15 | 12 | 10 | 8 | 7 | 7 | 4 |
| 100000 | 38 | 33 | 29 | 20 | 15 | 14 | 14 | 9 |
| 500000 | 431 | 294 | 191 | 128 | 85 | 75 | 76 | 54 |
| 1000000 | 1449 | 936 | 531 | 291 | 175 | 157 | 156 | 114 |
|  | | | | | | | | |
| QS 100 | 0 | 25 | 50 | 75 | 95 | 99 | 99,7 | inverted |
| 10000 | 255 | 246 | 231 | 206 | 166 | 140 | 137 | 85 |
| 50000 | 1605 | 1537 | 1296 | 1037 | 808 | 707 | 709 | 453 |
| 100000 | 3762 | 3333 | 2723 | 2121 | 1599 | 1450 | 1453 | 940 |
| 500000 | 42596 | 29981 | 19480 | 12527 | 8550 | 7557 | 7600 | 5514 |
| 1000000 | 144775 | 93949 | 54425 | 29235 | 17639 | 15747 | 15564 | 11339 |

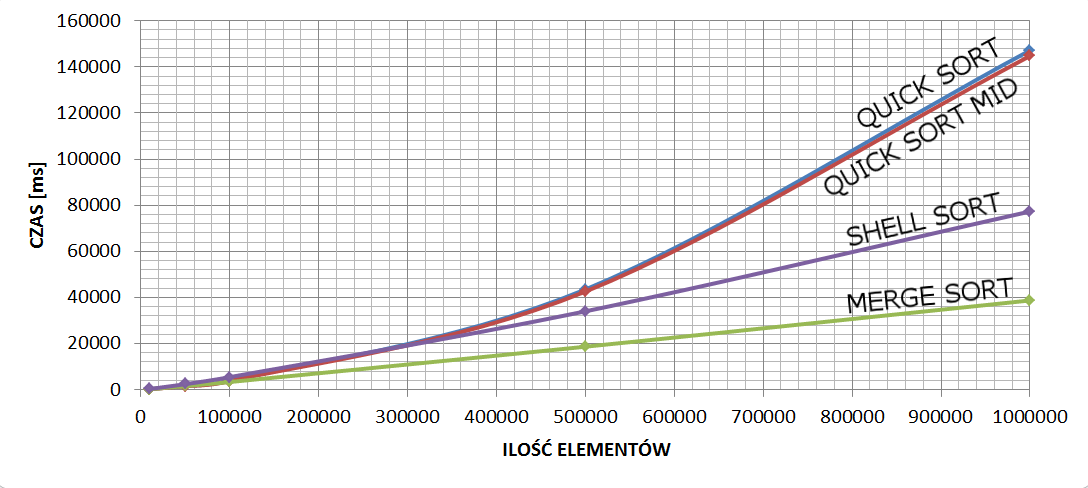
Merge Sort:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| MS | 0 | 25 | 50 | 75 | 95 | 99 | 99,7 | inverted |
| 10000 | 2 | 2 | 2 | 2 | 2 | 2 | 3 | 2 |
| 50000 | 16 | 15 | 13 | 12 | 11 | 11 | 11 | 11 |
| 100000 | 34 | 32 | 29 | 27 | 24 | 24 | 24 | 24 |
| 500000 | 189 | 174 | 161 | 148 | 137 | 135 | 134 | 135 |
| 1000000 | 389 | 361 | 334 | 316 | 286 | 281 | 279 | 281 |
|  | | | | | | | | |
| MS 100 | 0 | 25 | 50 | 75 | 95 | 99 | 99,7 | inverted |
| 10000 | 283 | 264 | 252 | 225 | 209 | 206 | 207 | 208 |
| 50000 | 1623 | 1516 | 1385 | 1271 | 1214 | 1157 | 1157 | 1158 |
| 100000 | 3401 | 3161 | 2896 | 2648 | 2463 | 2415 | 2407 | 2426 |
| 500000 | 18710 | 17331 | 15995 | 14671 | 13601 | 13396 | 13339 | 13394 |
| 1000000 | 38747 | 35865 | 33151 | 30525 | 28337 | 27910 | 27794 | 27954 |

Shell Sort:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| SS | 0 | 25 | 50 | 75 | 95 | 99 | 99,7 | inverted |
| 10000 | 3 | 3 | 3 | 3 | 2 | 2 | 1 | 1 |
| 50000 | 24 | 22 | 17 | 12 | 9 | 8 | 7 | 9 |
| 100000 | 55 | 45 | 33 | 22 | 18 | 15 | 14 | 19 |
| 500000 | 350 | 255 | 186 | 119 | 99 | 87 | 84 | 106 |
| 1000000 | 770 | 604 | 408 | 269 | 209 | 182 | 177 | 226 |
|  | | | | | | | | |
| SS 100 | 0 | 25 | 50 | 75 | 95 | 99 | 99,7 | inverted |
| 10000 | 359 | 337 | 325 | 300 | 233 | 161 | 133 | 157 |
| 50000 | 2477 | 2291 | 1737 | 1210 | 958 | 802 | 731 | 902 |
| 100000 | 5497 | 4682 | 3426 | 2296 | 1867 | 1561 | 1496 | 1932 |
| 500000 | 34056 | 26440 | 18695 | 11807 | 9915 | 8660 | 8439 | 10613 |
| 1000000 | 77313 | 59568 | 41176 | 25405 | 20989 | 18344 | 17696 | 22583 |

Wykres przedstawiający czas wykonywania sortowania całkowicie losowych stu tablic dla poszczególnych algorytmów:

  
  
  
  
  
  
IV. Wnioski

W ogólnym przypadku najszybszym algorytmem okazało się sortowanie przez scalanie. Sortowanie szybkie oraz Shell’a było od niego szybsze tylko przy odpowiednich warunkach początkowych (czyli częściowo posortowanej/odwrotnie posortowanej tablicy).  
Czasy wykonania sortowania przez scalanie zdawały się być „stabilne” względem reszty zaimplementowanych algorytmów, ponieważ czas ich wykonania zmieniał się nieznacznie przy zmianie warunków początkowych.  
  
Przy sortowaniu szybkim okazało się, że wybór pivot’a jako środka tablicy, we wszystkich wykonanych testach był wyborem szybszym od pivot’a losowego. Dodatkowo sortowanie to było najszybsze przy określonych warunkach początkowych (sortowanie odwrotnie posortowanej tablicy algorytmem Quick Sort dla miliona elementów trwało krócej, niż sortowanie tej samej tablicy algorytmem Merge Sort dla pół miliona elementów).  
  
Sortowanie Shella wykazywało dużą zależność od warunków początkowych, dodatkowo okazując się szybszym od sortowania szybkiego dla większej ilości elementów.  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
Źródła:  
  
wykłady dr.inż. Łukasza Jelenia  
C++ Kompendium Wiedzy Wydanie IV (2014), Bjarne’a Stroustrup’a  
https://pl.wikipedia.org/wiki/  
http://www.algorytm.org/  
http://www.cplusplus.com/  
https://stackoverflow.com/