**PROJEKTOWANIE ALGORYTMÓW I METODY SZTUCZNEJ INTELIGENCJI**

**SPRAWOZDANIE**

**PAWEŁ RAJKOWSKI**

**235216**

**PROJEKT NR. 2**

**NA OCENĘ 5**

W moim projekcie przeprowadziłem analizę 3 algorytmów sortowania:

* sortowania przez scalanie,
* sortowania szybkiego,
* sortowania introspektywnego.

Sortowanie przez scalanie polega na podziale sortowanej tablicy na coraz mniejsze fragmenty, aż do uzyskania tablic jednoelementowych, które są posortowane. Następnie fragmenty są łączone w większe dwójkami, przy zachowaniu kolejności liczb od najmniejszej do największej, aż nie otrzymamy początkowej tablicy, tylko że posortowanej.

Wysokość drzewa sortowania to O(log n), a na każdym poziomie drzewa wykonujemy O(n) operacji, zatem złożoność obliczeniowa sortowania przez scalanie to O(n log n).

Sortowanie szybkie polega na losowym wyborze piwota, po którego lewej stronie będą znajdować się elementy mniejsze od wylosowanego elementu, a po prawej – większe. Następnie dla fragmentów po lewej i prawej stronie piwota, wywoływane jest rekurencyjnie sortowanie szybkie.

Sortowanie to dla najgorszego przypadku (piwot często wylosowywany w pobliżu największego lub najmniejszego elementu tablicy, tzn. 90+ % elementów jest mniejszych/większych od piwota) ma złożoność O(n2), a dla przypadku średniego O(n log n).

Sortowanie introspektywne działa na zasadzie sortowania szybkiego. Jeśli sortowanie będzie wykonywało się dłużej niż powinno (przypadek pesymistyczny – więcej niż 2log2n wywołań rekurencyjnych), reszta fragmentów tablicy sortowana jest mniej złożonym algorytmem, który radzi sobie dobrze dla małych tablic (w moim przypadku jest to sortowanie przez kopcowanie).

Wykonałem testy dla tablic losowych liczb o rozmiarach (tablicy): 10 000, 50 000, 100 000, 500 000, 1 000 000 i różnych warunkach początkowych: 0%, 25%, 50%, 75%, 95%, 99%, 99.7% początkowych elementów posortowanych lub cała tablica posortowana odwrotnie.

Dla każdej z ww. tablic zmierzyłem czas z jakim sortuje się 100 tablic każdym z algorytmów sortowań (przez scalanie, szybkie, introspektywne).

Wyniki pomiarów:

SORTOWANIE PRZEZ SCALANIE

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0% | 25% | 50% | 75% | 95% | 99% | 99,70% | odwrócona |
| 10000 | 82 | 80 | 80 | 85 | 85 | 80 | 85 | 85 |
| 50000 | 485 | 485 | 485 | 485 | 502 | 485 | 485 | 480 |
| 100000 | 1026 | 1030 | 1030 | 1030 | 1030 | 1028 | 1031 | 1026 |
| 500000 | 5749 | 5602 | 5620 | 5622 | 5581 | 5642 | 5592 | 5582 |
| 1000000 | 11689 | 11669 | 11627 | 11612 | 11596 | 11550 | 11554 | 11560 |

Wykres kształtem przypomina krzywą *n log(n)*. Szybkość działania algorytmu zmienia się prawie niezauważalnie w zależności od tego, ile jest posortowanych początkowych elementów tablicy. W większości przypadków im więcej posortowanych początkowych elementów, tym szybciej działa algorytm sortowania. Co więcej, z pomiarów wynika, że odwrotne posortowanie tablicy, również, przyśpiesza sortowanie.

SORTOWANIE SZYBKIE

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0% | 25% | 50% | 75% | 95% | 99% | 99,70% | odwrócona |
| 10000 | 85 | 90 | 90 | 85 | 90 | 89 | 90 | 89 |
| 50000 | 495 | 495 | 508 | 495 | 500 | 485 | 495 | 496 |
| 100000 | 1078 | 1091 | 1090 | 1060 | 1066 | 1071 | 1065 | 1081 |
| 500000 | 8191 | 8381 | 8257 | 8221 | 8253 | 8358 | 8304 | 8298 |
| 1000000 | 24898 | 24565 | 24754 | 24717 | 24993 | 24972 | 25063 | 24801 |

Wykres kształtem przypomina coś pomiędzy krzywą *n2* a krzywą *n log(n)*. Kształt przypomina częściowo *n2* z powodu istnienia najgorszego przypadku (o złożoności O(n2)). Jako, że losowanie piwota jest czysto losowe, to ilość posortowanych elementów i odwrotne posortowanie, nie mają wpływu na szybkość działania algorytmu.

SORTOWANIE INTROSPEKTYWNE

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0% | 25% | 50% | 75% | 95% | 99% | 99,70% | odwrócona |
| 10000 | 400 | 429 | 390 | 395 | 390 | 376 | 415 | 385 |
| 50000 | 1914 | 1874 | 1807 | 1835 | 1874 | 1896 | 1846 | 1855 |
| 100000 | 3547 | 3477 | 3674 | 3617 | 3668 | 3481 | 3503 | 3601 |
| 500000 | 13236 | 13229 | 13285 | 13271 | 13129 | 13159 | 13160 | 13306 |
| 1000000 | 27596 | 27840 | 27630 | 27610 | 27734 | 27609 | 28100 | 27673 |

Wykres na początku szybko rośnie jak proporcjonalnie do sortowania szybkiego, ale dla większych tablic algorytm zaczyna działać szybciej, ponieważ częściej występują i są eliminowane najgorsze przypadki. Dla małych tablic ma złożoność O(n2), a dla dużych –   
O(n log n). Tak samo jak dla sortowania szybkiego, losowy wybór piwota sprawia, że ilość posortowanych wstępnie elementów i odwrotne posortowanie, nie mają wpływu na działanie programu.

W porównaniu do funkcji sort(T, T+n) z biblioteki <algorithm> moje algorytmy sortowania działają bardzo wolno (ok. 100 razy wolniej). Może to być spowodowane długimi nazwami zmiennych, lub słabej optymalizacji kodu.

Mimo to, wyznaczone charakterystyki sortowań ilustrują ich cechy (złożoność, wpływ ilości posortowanych elementów).

Podczas pracy nad projektem wykorzystałem następujące źródła:

* <http://lukasz.jelen.staff.iiar.pwr.wroc.pl/downloads/files/wyklad5www.pdf>  
  (wykład dr inż. Łukasza Jelenia na temat sortowań)
* <http://andrzej.rusiecki.staff.ict.pwr.wroc.pl/pliki/PAMSI_proj2.pdf>  
  (instrukcja do wykonanie projektu nr 2)
* <https://pl.wikipedia.org/wiki/Sortowanie_przez_kopcowanie>  
  (Wikipedia – sortowanie przez kopcowanie)
* <https://pl.wikipedia.org/wiki/Sortowanie_przez_scalanie>  
  (Wikipedia – sortowanie przez scalanie)
* <https://pl.wikipedia.org/wiki/Sortowanie_szybkie>  
  (Wikipedia – sortowanie szybkie)
* <https://pl.wikipedia.org/wiki/Sortowanie_szybkie>  
  (Wikipedia – sortowanie introspektywne)
* <https://www.quora.com/How-would-you-explain-O-log-n-in-algorithms-to-1st-year-undergrad-student-Can-any-one-explain-it-with-mathematical-proof-for-log-n-complexity-by-taking-a-simple-example-like-Binary-search-and-simple-to-understand>  
  (zdjęcie do porównanie kształtów charakterystyk dla różnych złożoności)