**Algorytmy i Struktury Danych – Laboratoria**

Dr. Hab. Inż. Małgorzata Sterna

Informatyka, semestr 2, grupa i2, wtorki godz. 13.30

21.03.2017r.

**Ćwiczenie nr.1 – Algorytmy Sortowania**

Krzysztof Pasiewicz, 132302

Mikołaj Frankowski,

Zad. 2

Porównanie algorytmów sorotwania ze względu na czas ich wykonywania. SelectionSort, InsertionSort, BubbleSort, HeapSort, MergeSort, QuickSort, CountingSort.

Wyk. 1 – Sortowania Proste

Wykres 2 – Sortowania zaawansowane

|  |  |  |  |
| --- | --- | --- | --- |
| **Quant** | **SS** | **IS** | **BS** |
| 10000 | 0,12 | 0,093 | 0,233 |
| 20000 | 0,487 | 0,369 | 0,992 |
| 30000 | 1,086 | 0,824 | 2,262 |
| 40000 | 1,893 | 1,453 | 4,103 |
| 50000 | 2,96 | 2,266 | 6,492 |
| 60000 | 4,311 | 3,315 | 9,562 |
| 70000 | 5,876 | 4,626 | 13,24 |
| 80000 | 7,633 | 5,873 | 17,25 |
| 90000 | 9,64 | 7,471 | 21,78 |
| 100000 | 11,88 | 9,236 | 27 |

Tab. 1 – Czasy w sekundach dla zadanej ilości elementów.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Quant** | **HS** | **MS** | **QS** | **CS** |
| 60000 | 0,047 | 0,463 | 0,008 | 0,001 |
| 80000 | 0,054 | 0,809 | 0,011 | 0,001 |
| 100000 | 0,074 | 1,27 | 0,012 | 0,001 |
| 120000 | 0,087 | 1,817 | 0,016 | 0,002 |
| 140000 | 0,115 | 2,52 | 0,018 | 0,002 |
| 160000 | 0,124 | 3,23 | 0,021 | 0,003 |
| 180000 | 0,133 | 4,026 | 0,023 | 0,003 |
| 200000 | 0,146 | 5,064 | 0,025 | 0,004 |
| 220000 | 0,188 | 6,113 | 0,03 | 0,004 |
| 240000 | 0,192 | 7,341 | 0,031 | 0,004 |

Tab.2 – Czasy w sekundach dla zadanej ilości elementów

Zgodnie z przeiwdywaniami algorytmy sortowania SelectionSort, InsertionSort, BubbleSort okazały się być znacząco mniej efektywne od pozostałych sortowań. Prawidłowe zobrazowanie wyników wymagało przedstawnienia otrzymanych danych na osobnych wykresach. Należy jednak zauważyć,że wydajność powyższych algorytmów zależy nie tylko od samego algorytmu, ale także od zakresu danych wejściowych, oraz ich rozkładu. Na przykład dla liczb losowanych z zakresu [1;n\*100] wykres ten mógłby wyglądać zupełnie inaczej. Algormtymy sortowania możemy podzielić ze względu na złożoność, zajmowaną pamięć oraz wrażliwość na dane wejściowe.

Sortowania proste SS, IS, BS mają tą samą złożoność średnią wynoszącą O(n2). Wszystkie algorytmy działają w miejscu, a najlepszym spośród nich okazuje się być InsertionSort, którego złożoność obliczeniowe dla ciągu jużposortowanego wynosi O(n), natomiast SS i BS w tym przypadku nadal mają złożoność kwadratową. Dlatego też warto stosować IS do sortowania ciągów uprzednio częściowo posortowanych (np. Dodawanie elementów do ciągu posortowanego, z czego korzysta chociażby BucketSort). Zarówno BubbleSort jak i InsertionSort są sortowaniami stabilnymi czego, nie można powiedzieć o algorytmie sortowania przez wybór.

Jednak metody te nadal pozostają bardzo nieefektywne jeśli porównamy je z pozostałymi metodami sortowania. Sortowania MergeSort, HeapSort, QuickSort i CountingSort dla dużo większej liczby elementów nadal osiągają znacznie lepsze wyniki pod względem czasu działania. Najlepszym algorytmem okazuje się być CountSort o złożoności liniowej. Niestety do posortowania danych wymaga on dużych zasobów pamięci. Spośród algorytmów o złożoności O(nlog2n) najlepszym okazuje się QuickSort, lecz jest on wrażliwy na dane wejściowe i jego złożoność w przypadku pesymistycznym może spaść do O(n2). Następnym algorytmem okazuje się być HeapSort, który jednocześnie zawsze cechuje się złożonością logarytmiczną, lecz jest niestabilny i dość ciężki w implementacji. Najmniej efektywną metodą spośród algorytmów o złożoności logarytmicznej okazał się MergeSort, który potrzebuje również dodatkowych zasobów pamięciowych. Jednakże jest on niewrażliwy na dane wejściowe i stabilny dzięki czemu nigdy nie będzie miał złożoności gorszej niż O(nlog2n).

Zad. 4

Porównanie QuickSort’a (po środkowym elemencie) i CountingSort’a z zakresów: [1,n\*100] i [1,n/100].

Wykres – Zależność czasowa QS i CS w zakresie [1,n\*100]

Wykres – Porównanie QS i CS w zakresie [1,n/100]

|  |  |  |
| --- | --- | --- |
| **Quant** | **QS** | **CS** |
| 250000 | 0,036 | 0,108 |
| 500000 | 0,066 | 0,184 |
| 1000000 | 0,137 | 0,354 |
| 1500000 | 0,201 | 0,545 |
| 2000000 | 0,271 | 0,713 |
| 2500000 | 0,336 | 0,89 |
| 3000000 | 0,406 | 1,076 |
| 3500000 | 0,479 | 1,293 |
| 4000000 | 0,537 | 1,458 |
| 4500000 | 0,608 | 1,648 |

|  |  |  |
| --- | --- | --- |
| **Quant** | **QS** | **CS** |
| 250000 | 0,024 | 0,002 |
| 500000 | 0,058 | 0,003 |
| 1000000 | 0,104 | 0,007 |
| 1500000 | 0,155 | 0,01 |
| 2000000 | 0,211 | 0,018 |
| 2500000 | 0,268 | 0,016 |
| 3000000 | 0,318 | 0,019 |
| 3500000 | 0,368 | 0,022 |
| 4000000 | 0,414 | 0,025 |
| 4500000 | 0,462 | 0,028 |

Tab. – zakres [1,n\*100] Tab. – zakres [1,n/100]

Powyższe dane przedstawione graficznie na wykresach pokazuje jak pozornie najlepszy pod względem złożoności algorytm sortowania ( CountingSort o złożoności O(n) ) jest zależny od zakresu danych wejściowych i porównanie go z algorytmem sortowania szybkiego (QuickSort o złożoności średniej O(nlogn) ). Dla danych z zakresu [1;n/100] CountingSort jest zdecydowanie bardziej efektywny od QuickSort’a. Natomiast dla danych z zakresu znacząco przekraczającego ilość sortowanych elementów CountingSort staje się znacznie gorszy pod względem czasu, a QuickSort radzi sobie znacznie lepiej. Dodatkowo do wad sortowania przez zliczanie możemy zaliczyć jego duże zapotrzebowanie pamięciowe oraz ograniczenie danych wejściowych do liczb całkowitych dodatnich. Powyższe doświadczenie pokazuje jak ciężko znaleźć najlepszy możliwy w każdej instancji problemu sortowania algorytm. Stąd też, jeśli chcemy sortować dane w najszybszy możliwy sposób powiniśmy dobierać metodę sortowania do aktualnej instancji naszego problemu.