|  |  |  |
| --- | --- | --- |
| Prowadzący:  Dr inż. Łukasz Jeleń | Projektowanie Algorytmów i Metody  Sztucznej Inteligencji | Termin zajęć:  Śr 7:30 |
| Wojciech Gołębiowski  241477 | **„Porównanie algorytmów sortowania”** | Data oddania sprawozdania:  3.04.2019 r. |

1. **Wprowadzenie**

Program ma za zadanie porównać ze sobą czasy sortowania 100 tablic o różnych wymiarach za pomocą trzech algorytmów. Tablice do posortowania będą zawierały losowe elementy, częściowo posortowane lub posortowane odwrotnie. Na podstawie czasu wykonywania algorytmu będzie można wybrać najbardziej efektywny obliczeniowo.

1. **Opis badanych algorytmów**
   1. Sortowanie Shella (Shellsort)  
      Jest to uogólnienie sortowania przez wstawianie, sortuje on elementy tablicy położone od siebie o odległość równą n/2 , a następnie podwójnie zmniejsza odstęp miedzy sortowanymi elementami aż do uzyskania odległości mniejszej lub równej 1 Algorytm ten pracuje najefektywniej gdy tablica jest już częściowo posortowana. Jego złożoność obliczeniowa zależy od wybranych ciągów odstępów, w tym przypadku wynosi ona O(n2).
   2. Sortowanie przez scalanie (merge sort)  
      Jest to algorytm stosujący metodę dziel i zwyciężaj, czyli dzieli on tablice rekurencyjnie na dwie podtablice, sortuje je oddzielnie przez scalanie chyba że pozostał już tylko jeden element, a następnie łączy posortowane podtablice w jedną posortowaną. Złożoność obliczeniowa tego algorytmu wynosi O(nlogn).
   3. Sortowanie szybkie (quicksort)  
      Jest to jeden z najpopularniejszych algorytmów sortowania. Działa na zasadzie dziel i zwyciężaj. Na początku wybiera się element osiowy (w tym przypadku środek tablicy), po czym pierwszy element od początku tablicy większy od element osiowego zamienia się miejscami z pierwszym elementem od końca tablicy mniejszym od elementu osiowego. następnie sortuje się osobno lewą i prawą część tablicy. Złożoność obliczeniowa tego algorytmu wynosi O(nlogn), a w pesymistycznym przypadku O(n2).
2. **Omówienie przebiegu eksperymentów**

W poniższych tabelach zaprezentowano czasy w jakich dany algorytm w danym przypadku wykonał sortowanie.

|  |  |  |  |
| --- | --- | --- | --- |
| wszystkie elementy tablicy losowe | | | |
| Il. El. | Shell | Quick | Merge |
| 10000 | 0,21 | 0,546 | 3,351 |
| 50000 | 0,457 | 3,217 | 71,134 |
| 100000 | 0,981 | 6,794 | 337,465 |
| 500000 | 5,526 | 39,619 |  |
| 1000000 | 11,506 | 84,66 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 25% początkowych elementów tablicy jest już posortowanych, | | | |
| Il. El. | Shell | Quick | Merge |
| 10000 | 0,055 | 0,117 | 3,293 |
| 50000 | 0,323 | 0,67 | 71,688 |
| 100000 | 0,699 | 1,386 | 337,038 |
| 500000 | 4,101 | 6,04 |  |
| 1000000 | 8,499 | 12,196 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 50% początkowych elementów tablicy jest już posortowanych, | | | |
| Il. El. | Shell | Quick | Merge |
| 10000 | 0,055 | 0,117 | 3,375 |
| 50000 | 0,326 | 0,671 | 70,644 |
| 100000 | 0,697 | 1,368 | 333,544 |
| 500000 | 3,997 | 5,996 |  |
| 1000000 | 8,484 | 12,2 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 75% początkowych elementów tablicy jest już posortowanych, | | | |
| Il. El. | Shell [s] | Quick [s] | Merge [s] |
| 10000 | 0,055 | 0,117 | 3,261 |
| 50000 | 0,326 | 0,67 | 70,234 |
| 100000 | 0,695 | 1,366 | 329,899 |
| 500000 | 4,072 | 6,03 |  |
| 1000000 | 8,442 | 12,203 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 95% początkowych elementów tablicy jest już posortowanych, | | | |
| Il. El. | Shell [s] | Quick [s] | Merge [s] |
| 10000 | 0,055 | 0,118 | 3,248 |
| 50000 | 0,324 | 0,672 | 76,139 |
| 100000 | 0,696 | 1,37 | 329,442 |
| 500000 | 3,977 | 6,011 |  |
| 1000000 | 8,644 | 12,231 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 99% początkowych elementów tablicy jest już posortowanych, | | | |
| Il. El. | Shell [s] | Quick [s] | Merge [s] |
| 10000 | 0,055 | 0,118 | 3,246 |
| 50000 | 0,331 | 0,673 | 76,157 |
| 100000 | 0,703 | 1,36 | 346,98 |
| 500000 | 3,983 | 5,99 |  |
| 1000000 | 8,453 | 12,228 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 99,7% początkowych elementów tablicy jest już posortowanych, | | | |
| Il. El. | Shell [s] | Quick [s] | Merge [s] |
| 10000 | 0,055 | 0,117 | 3,255 |
| 50000 | 0,324 | 0,67 | 69,513 |
| 100000 | 0,695 | 1,372 | 329,81 |
| 500000 | 4,025 | 5,987 |  |
| 1000000 | 8,808 | 12,226 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Posortowane odwrotnie [s] | | | |
| Il. El. | Shell [s] | Quick [s] | Merge [s] |
| 10000 | 0,077 | 0,179 | 3,256 |
| 50000 | 0,46 | 0,981 | 69,028 |
| 100000 | 0,991 | 1,993 | 329,052 |
| 500000 | 5,459 | 9,093 |  |
| 1000000 | 11,711 | 18,399 |  |

Na poniższych wykresach zaprezentowano w jakim czasie dany algorytm posortował tablice w zależności od przypadku.

Na poniższych wykresach pokazano zależności czasu od ilości elementów.

1. **Wnioski**
   1. Z wykresu zależności czasu od posortowanych elementów widać, że najszybciej działającymi w każdej sytuacji algorytmami są Quicksort i Shellsort
   2. Algorytm Shella sortuje najszybciej ze wszystkich tablice częściowo posortowane, dzieje się tak dlatego, że stosujemy odstęp między sortowanymi elementami tablicy.
   3. Algorytm sortowania przez scalanie w tym przypadku wykonuje się najdłużej, na tyle długo, że nie udało się przeprowadzić pomiarów dla tablic większych niż 100000
2. **Literatura**

* <https://www.geeksforgeeks.org/shellsort/>
* <http://www.algorytm.org/algorytmy-sortowania/sortowanie-przez-scalanie-mergesort/merge-c.html>
* <https://pl.wikipedia.org/wiki/Sortowanie_przez_scalanie>
* <https://pl.wikipedia.org/wiki/Sortowanie_Shella>
* <https://pl.wikipedia.org/wiki/Sortowanie_szybkie>
* <http://www.algorytm.org/algorytmy-sortowania/sortowanie-szybkie-quicksort/quick-1-c.html>