Dokumentacja projektu ASD

Do projektu zaimplementowałem 4 różne algorytmy sortowania. Algorytmy, które zaimplementowałem to: HeapSort, QuickSort, InsertSort oraz MergeSort. Dla każdego z algorytmów sprawdziłem czas jego trwania. Każdy z nich przyjął tablicę o wielkości 100000 elementów. Wyniki przedstawiają się następująco. Dane podane są w sekundach.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Dane losowe | Dane posortowane | Dane odwrotnie posortowane |
| HeapSort | 0.94793701171875 | 0.9271266460418701 | 0.8645343780517578 |
| QuickSort | 0.3356471061706543 | Algorytm nie wykonał się | Algorytm nie wykonał się |
| InsertSort | 432.14152932167053 | 0.020102500915527344 | 879.4231886863708 |
| MergeSort | 0.5151631832122803 | 0.4424476623535156 | 0.46915245056152344 |

Wnioski jakie na podstawie powyższych wyników można uzyskać są następujące. Każdy z algorytmów ma swoje zastosowanie w pewnych przypadkach. Analizując po kolei.

HeapSort dla każdej porcji danych działa w podobnym czasie, więc nie ważne gdzie go użyjemy zadziała bardzo dobrze.

QuickSort najszybciej poradził sobie z posortowaniem tablicy, jednak dla danych posortowanych był na tyle zasobożerny, że interpreter pythona nie był w stanie go użyć i program nie kończył pracy. Dla danych odwrotnie posortowanych problem był taki sam. Zatem według powyższych danych można wywnioskować, że QuickSort jest najszybszy dla danych losowych jednak bardzo obciąża komputer.

InsertSort działał najgorzej ze wszystkich algorytmów dla liczb losowych, natomiast dla tablicy posortowanej zadziałał najszybciej ze wszystkich. Dla tablicy odwrotnie posortowanej algorytm wykonał się jeszcze gorzej niż dla liczb losowych. Na podstawie tych danych można wywnioskować, że InsertSort sprawdzi się idealnie do sprawdzenia czy tablica posortowana jest prawidłowo.

MergeSort działa bardzo podobnie jak HeapSort natomiast sporo szybciej, co sprawia, że wydaje się być najbardziej uniwersalny.