**Algorytmy i Struktura Danych – Projekt I**

Filip Bianga s19329

gr. 44c

Wnioski:

|  |  |  |  |
| --- | --- | --- | --- |
| Tablica losowa | 300 000 | 400 000 | 500 000 |
| QuickSort | 64292 ms | 90207 ms | 133959 ms |
| MergeSort | 90797 ms | 122220 ms | -- |
| HeapSort | 128795 ms | 184506 ms | -- |
| InsertionSort | 56756334 ms | 106120365 ms | -- |

|  |  |  |  |
| --- | --- | --- | --- |
| 100 000  elementów | Tablica losowych liczb | Tablica posortowanych liczb | Tablica odwrotnie posortowanych liczb |
| QuickSort | 19115 ms | 26415146 ms | 25928730 ms |
| MergeSort | 27358 ms | 11529 ms | 11452 ms |
| HeapSort | 37651 ms | 22402 ms | 22165 ms |
| InsertionSort | 6829582 ms | 359 ms | 344 ms |

Zastosowałem 100 000 elementów to stworzenia powyższej tabeli, ponieważ przy minimalnej podanej w zadaniu 300 000 elementów program nie wyświetlał wyników.

InsertionSort dobrze sprawdza się przy małych danych/wartościach lecz przy sporo większych jak widać wyżej w tabeli, czas znacząco różni się od pozostałych algorytmów sortujących dlatego zachowuję się najgorzej i nie powinien być stosowany przy bardziej złożonych rozwiązaniach.

Najlepiej działającym algorytmem spośród 4 wyżej wymienionych sprawdził się QuickSort, jako jedyny przeszedł wszystkie trzy testy (pozostałe algorytmy nie są niczemu winne ponieważ, moc obliczeniowa mojego komputera jak i kompilatora online musiała być o wiele mniejsza, gdyż program po pewnym czasie się kończył i nie wyświetlał pozostałych rozwiązań)

Według mnie zgodne z moimi oczekiwaniami było to, że algorytm InsertionSort będzie najwolniejszy i nawet nie spodziewałem się, że zostanie wyświetlony mi wynik czasowy przy tak dużej liczbie elementów wylosowanej tablicy.