Projektowanie Algorytmów i Metody Sztucznej Inteligencji	
Prowadzący *Mgr inż. Marta Emirsajłow*	**Termin** *Poniedziałek 15:15*
Imię, nazwisko, numer albumu,grupa *Mateusz Szlachetko 259370 Y03-51a*	**Data** *27 marca 2022*
Temat ćwiczenia i nr *Projekt 1*	

1 Wstęp

1.1 Cel ćwiczenia

Celem ćwiczenia było wybranie i zaimplementowanie 3 algorytmów sortowania, oraz przeanalizowanie ich efektywności poprzez wielokrotne sortowanie tablic o zadanych wymiarach i początkowym uporządkowaniu elementów.

Tabela 1: Wybrane algorytmy

Merge sort	**Quick sort**	**Introspective sort**

1.2 Problem sortowania

Sortowanie jest jednym z podstawowych problemów informatyki. W ogólności polega na uporządkowaniu danych ze zbioru względem przyjętego klucza. Dla naszego przypadku sprowadza się do układania kolejnych wartości liczbowych, od najmniejszego do największego elementu. Cele takiej operacji mogą być różne, dlatego w zależności od tego co potrzebujemy, jesteśmy w stanie dobrać odpowiedni algorytm sortowania. Różnią się one pomiędzy sobą złożonością obliczeniową, która przekłada się bezpośrednio na czas sortowania, ale i tym ile pamięci wykorzystują w danym procesie. Dodatkowo, algorytmy możemy klasyfikować jako stabilne, bądź niestabilne, w zależności od tego, czy elementy o tej samej wartości znajdujące się w tablicy wejściowej w danej kolejności, będą w takiej samej kolejności w tablicy wyjściowej.

2 Opis algorytmów

2.1 Quick sort

Działanie "sortowania szybkiego" jako algorytmu rekurencyjnego polega na wybraniu jednego elementu z tablicy (w moim przypadku był to ostatni element), który zostaje określony jako tzw. pivot. Następnie przechodząc przez zbiór do posortowania, kolejne elementy porównujemy z naszym pivotem i elementy mniejsze przenosimy do lewej części tablicy, a elementy większe do prawej(sortowanie w miejscu). Po tej operacji na naszych dwóch wydzielonych partycjach, lewej z mniejszymi elementami, prawej z większymi elementami, wywołujemy ponownie algorytm sortowania i rekurencyjnie sortujemy w ten sposób wszystkie elementy.

Przenoszenie elementów tablicy odbywa się w czasie liniowym. W zależności od tego ile wystąpi rekurencyjnych wywołań naszej funkcji, złożoność czasowa wyniesie optymistycznie = średnio- O(n logn) lub pesymistycznie O(n^2). Wynika to z tego, że w przypadku optymistycznym tablica zostanie podzielona na pół, wtedy głębokość drzewa wywołań określa zależność log n, gdzie n to liczba elementów tablicy(Udowodniono, że taki sam rząd złożoności wystąpi w przypadku średnim). Pesymistycznie jednak, gdy nasz pivot zawsze znajduje się na brzegu tablicy, przy każdym kolejnym wywołaniu liczba elementów do posortowania jest tylko o 1 mniejsza, co za tym idzie funkcja wywoła się n razy, gdzie n to liczba elementów tablicy.

2.2 Merge sort

Algorytm sortowania "przez scalanie" polega na podziale naszej wejściowej tablicy nieuporządkowanych elementów, aż do uzyskania tablic posortowanych, które następnie łączone są w tablice wyjściową, już z posortowanymi elementami. Najprostszą posortowaną tablicą, jest tablica złożona z jednego elementu. Algorytm rekurencyjnie rozbija kolejne podtablice na ich mniejsze, równe do co ilości elementów odpowiedniki, aż uzyskamy tablice złożone z samych posortowanych elementów, wtedy kolejno porównuje ze sobą elementy z sub tablic i wstawia je do naszej początkowej tablicy.

Złożoność obliczeniowa dla algorytmu merge sort wynosi O(n logn) dla każdego przypadku, niezależnie od danych wejściowych, wynika to z tego, że głębokość drzewa wywołań określa zależność log n, a ilość porównań elementów ma liniowy czas trwania n . Nie występuje w nim więc przypadek pesymistyczny taki jak w Quick sort.

2.3 Intro sort

"Sortowanie introspektywne" jest hybrydowym algorytmem sortowania, który składa się z 3 algorytmów. Sortowania szybkiego, przez kopcowanie i przez wstawianie. Takie połączenie pozwala zachować średnią złożoność czasową na poziomie O(n logn) przy braku wystąpienia pesymistycznej wersji złożoności charakterystycznej dla Quick sort. Przy założeniu maksymalnej ilości rekurencyjnych wywołań dla quicksort, po zmniejszeniu ilości przejść do 0, algorytm zaczyna sortować używając algorytmu heapsort, i od pewnej ustalonej ilości elementów używa introsort. Przy implementacji, bez użycia introsort, algorytm stawał się nieefektywny, dlatego potrzebna jest kombinacja właśnie tych 3 algorytmów. Jego zaletą jest zachowanie właściwości quicksort, przy czym nie jest tak wrażliwy na

uporządkowanie danych na wejściu i jego czas wywołania nie zmienia się znacząco dla uporządkowanych zbiorów.

3 Eksperyment

Przebieg eksperymentu polegał na wywołaniu każdego z wybranych algorytmów sortowania, dla 100 tablic o różnym stopniu (procentowym) początkowego uporządkowania elementów, lub kolejności w jakiej elementy były początkowo posortowane (malejąco). Całe badanie polegało na zliczaniu czasu potrzebnego na posortowanie każdej kolejnej tablicy(ze 100 tablic), zsumowaniu wszystkich pojedynczych przejść i wyświetleniu wyników. Po wywołaniu wszystkich potrzebnych kombinacji, wyniki dla każdego z algorytmów prezentują się następująco:

Tabela 2: Czas potrzebny na posortowanie elementów z początkowym uporządkowaniem: 0%

Algorytm[czas ms]/Ilość elementów	**QuickSort**	**MergeSort**	**IntroSort**
10 000	119	100	100
50 000	868	702	774
100 000	1797	1501	1673
500 000	10294	8525	10273
1 000 000	21593	17926	22051

Tabela 3: Czas potrzebny na posortowanie elementów z początkowym uporządkowaniem: 25%

Algorytm[czas ms]/Ilość elementów	**QuickSort**	**MergeSort**	**IntroSort**
10 000	101	100	101
50 000	815	603	857
100 000	1825	1326	1886
500 000	10653	7766	11174
1 000 000	22444	16157	25978

Tabela 4: Czas potrzebny na posortowanie elementów z początkowym uporządkowaniem: 50%

Algorytm[czas ms]/Ilość elementów	**QuickSort**	**MergeSort**	**IntroSort**
10 000	140	100	120
50 000	1141	659	1006
100 000	2151	1810	2247
500 000	14098	8420	14268
1 000 000	29838	17215	28287

Tabela 5: Czas potrzebny na posortowanie elementów z początkowym uporządkowaniem: 75%

Algorytm[czas ms]/Ilość elementów	**QuickSort**	**MergeSort**	**IntroSort**
10 000	103	73	212
50 000	1032	506	1462
100 000	2215	1135	2840
500 000	13656	6531	20809
1 000 000	32595	13447	34298

Tabela 6: Czas potrzebny na posortowanie elementów z początkowym uporządkowaniem: 95%

Algorytm[czas ms]/Ilość elementów	**QuickSort**	**MergeSort**	**IntroSort**
10 000	307	49	210
50 000	1874	402	1452
100 000	3961	902	3261
500 000	24147	5124	18173
1 000 000	53983	11228	38405

Tabela 7: Czas potrzebny na posortowanie elementów z początkowym uporządkowaniem: 99%

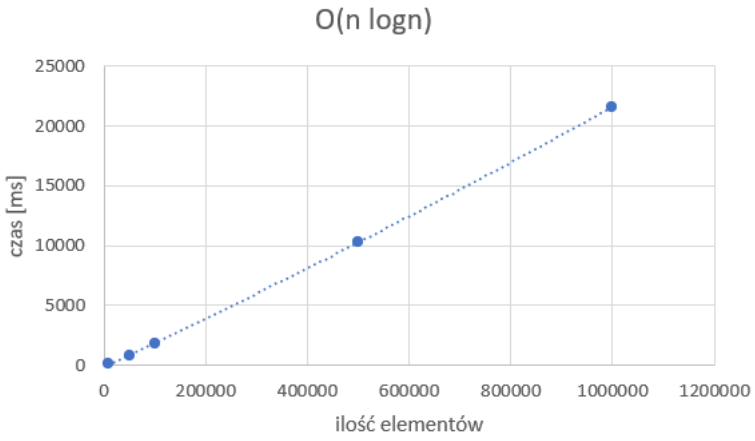
Algorytm[czas ms]/Ilość elementów	**QuickSort**	**MergeSort**	**IntroSort**
10 000	391	55	210
50 000	4699	458	1749
100 000	13271	1050	3920
500 000	85780	6127	19396
1 000 000	162541	12786	47850

Tabela 8: Czas potrzebny na posortowanie elementów z początkowym uporządkowaniem: 99.7%

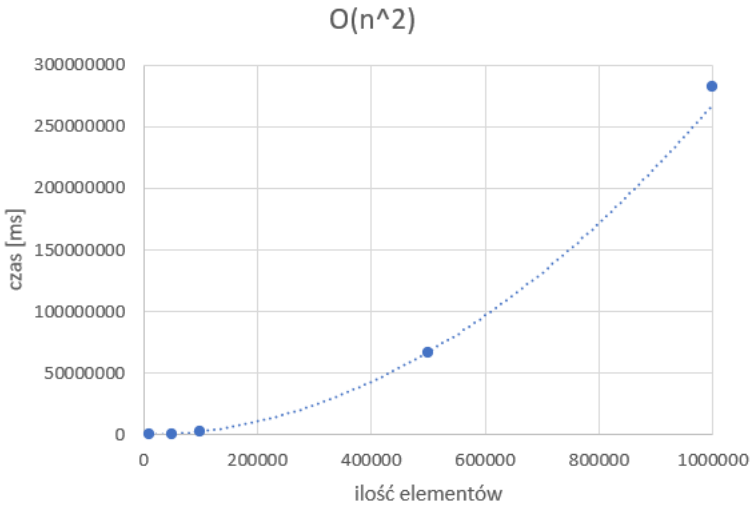
Algorytm[czas ms]/Ilość elementów	**QuickSort**	**MergeSort**	**IntroSort**
10 000	831	40	201
50 000	5611	402	1374
100 000	15135	873	3009
500 000	173656	4984	17257
1 000 000	463007	10438	36293

Tabela 9: Czas potrzebny na posortowanie elementów z początkowym uporządkowaniem: 100% Kolejność: Malejąca

Algorytm[czas ms]/Ilość elementów	**QuickSort**	**MergeSort**	**IntroSort**
10 000	28800	10	301
50 000	650,6 [s]	405	1717
100 000	42,915 [min]	907	3764
500 000	18,3971 [godz]	5188	21556
1 000 000	78,53277 [godz]	10897	44878



Rysunek 1: Wykres t(n) obrazujący średni przypadek dla każdego z algorytmów



Rysunek 2: Wykres zależności t(n) wywołania algorytmu quicksort z uprządkowaną tablicą

4 Wnioski

- Wybrane algorytmy sortowania, dla przypadku z losowymi danymi zachowują się podobnie.
- Wraz ze wzrostem uporządkowania najgorzej zaczyna zachowywać się quicksort.
- Merge sort nie wykazuje znaczącej wrażliwości na dane w tablicach, a introsort spowalnia się analogicznie jak quicksort dla większego uporządkowania, przy czym w granicznych przypadkach, zaczyna już funkcjonować jego mechanizm wykluczający przypadek pesymistyczny.
- Dla całkowitego uporządkowania Quicksort osiąga pesymistyczną czasową złożoność obliczeniową $O(n^2)$ co obrazuje wykres(2)
- Dla zadanych w ćwiczeniu problemów najlepiej sprawdził się algorytm Mergesort. Pozostałe dwa algorytmy przez swoją wrażliwość na uporządkowanie danych, traciły na szybkości, przy czym introsort jest odporny na przypadek graniczny. Był on jednak trudniejszy w implementacji, a Mergesort wykorzystuje więcej pamięci, dlatego końcowy wybór zawsze musi być uzależniony od naszego celu.
- Wszystkie wyniki są charakterystyczne dla danego komputera na którym były wykonywane pomiary i od sposobu wywoływania funkcji. W zależności ile zasobów przeznaczymy dla wykonywanego programu, dostaniemy różną prędkość wykonywania.W ogólności czasy sortowania różnią się z każdym wywołaniem z racji na losowość ułożenia elementów w tablicy, ale uśrednione wyniki pokrywają się z przewidywaniami teoretycznymi.