Politechnika Wrocławska

Katedra Informatyki Technicznej

Wydział Elektroniki

Projektowanie Algorytmów i Metody Sztucznej Inteligencji

Projekt 1: Sortowania

Prowadzący: dr inż. Andrzej Rusiecki

Autor: Julia Dorobisz

Termin: środa 18:55

Data oddania: 02.04.2019r.

1. **Wprowadzenie:**

   Celem projektu była samodzielna implementacja trzech wybranych algorytmów sortowania oraz przeprowadzenie testów ich efektywności. Opracowywane przeze mnie algorytmy to: sortowanie przez scalanie, sortowanie szybkie oraz sortowanie introspektywne.

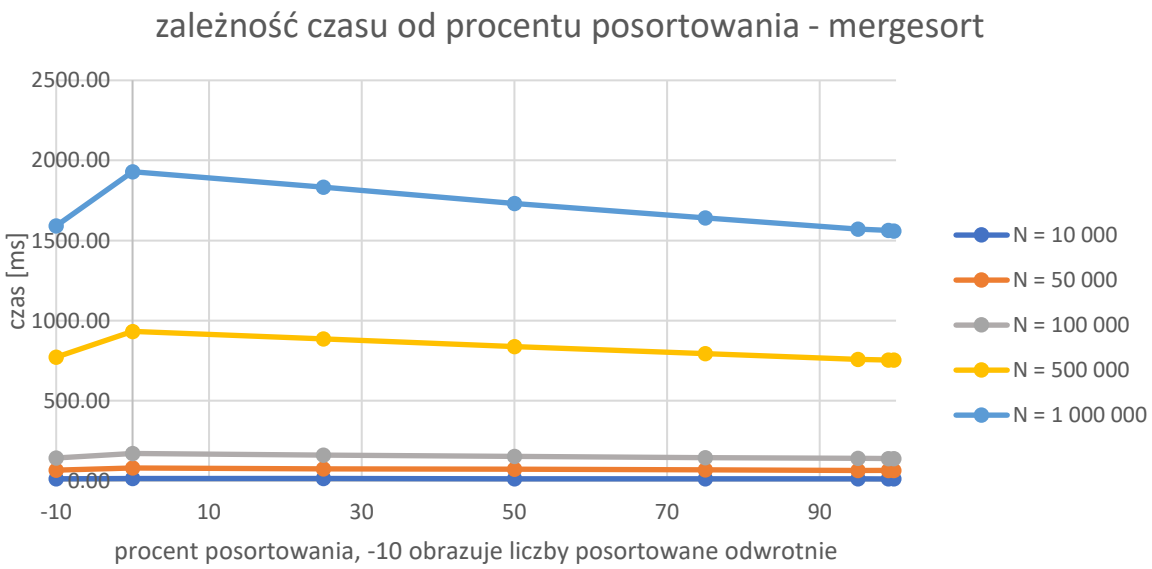
2. **Opis algorytmów:**

   a) **Sortowanie przez scalanie:**

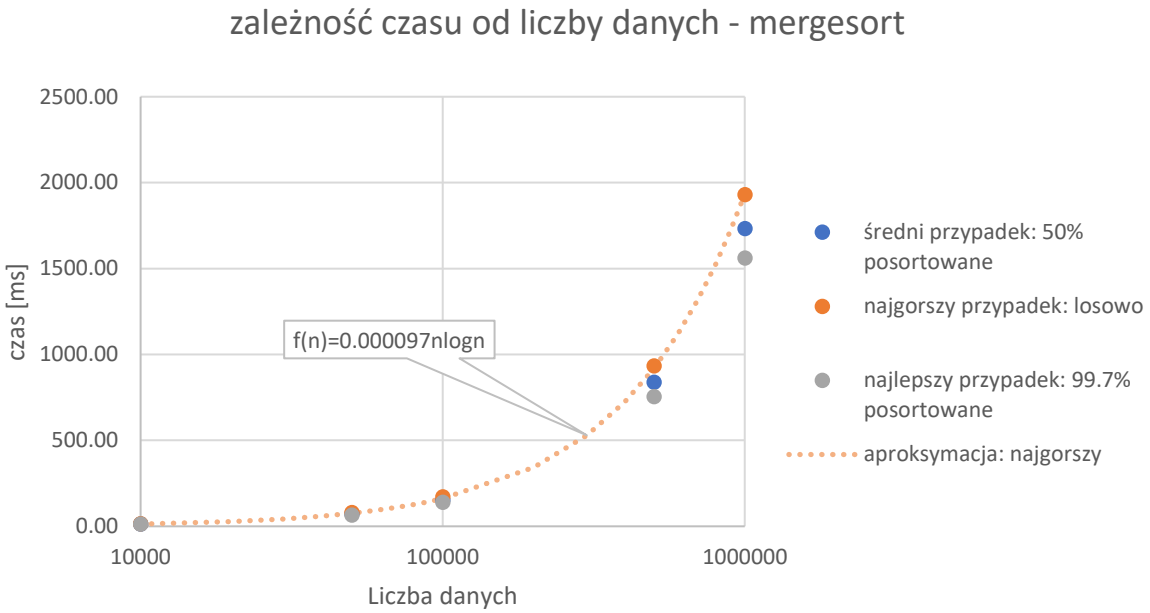
   Jest to algorytm rekurencyjny oparty na metodzie „dziel i zwyciężaj” W kolejnych krokach dzieli on problem na mniejsze podproblemy, aż dojdzie do pojedynczej liczby, która z definicji jest już posortowana. Sortowanie właściwe następuje w kolejnej fazie algorytmu czyli w trakcie scalania podproblemów. Wadą tego rozwiązania jest konieczność tworzenia dodatkowych tablic przechowujących podproblemy. Poniżej zamieszczam tabelę wraz z uśrednionymi wynikami badań 100 tablic każdego rodzaju:

N\stopień posortowania	odwrotnie:	0%	25%	50%	75%	95%	99%	99.7%
10000	12.54	14.27	13.80	12.95	12.23	11.88	11.62	11.88
50000	66.62	79.91	75.33	71.48	67.84	65.36	64.91	64.98
100000	142.64	170.50	161.06	152.48	144.93	139.53	138.95	139.15
500000	771.48	932.67	885.37	838.16	792.98	757.00	753.90	753.30
1000000	1591.04	1929.59	1833.53	1731.39	1641.18	1571.33	1562.61	1560.17

Algorytm sortowania przez scalanie jest algorytmem naturalnym, czyli takim, który posortuje tablicę tym szybciej, im więcej elementów jest już posortowanych. Własność tą można zaobserwować na poniższym wykresie:



Złożoność obliczeniowa algorytmu jest równa $O(nlog_2 n)$. Jest tak dlatego, że w fazie dzielenia tablicy na mniejsze powstaje drzewo o głębokości $log_2 n$. W fazie scalania przechodzimy po wszystkich poziomach drzewa scalając wszystkie występujące na danym poziomie ciągi liczb. Scalanie postępuje w czasie liniowym, zatem ostateczna złożoność w notacji O wynosi $nlog_2 n$ we wszystkich przypadkach, co potwierdza poniższy wykres z dopasowaną krzywą liniowo-logarytmiczną:

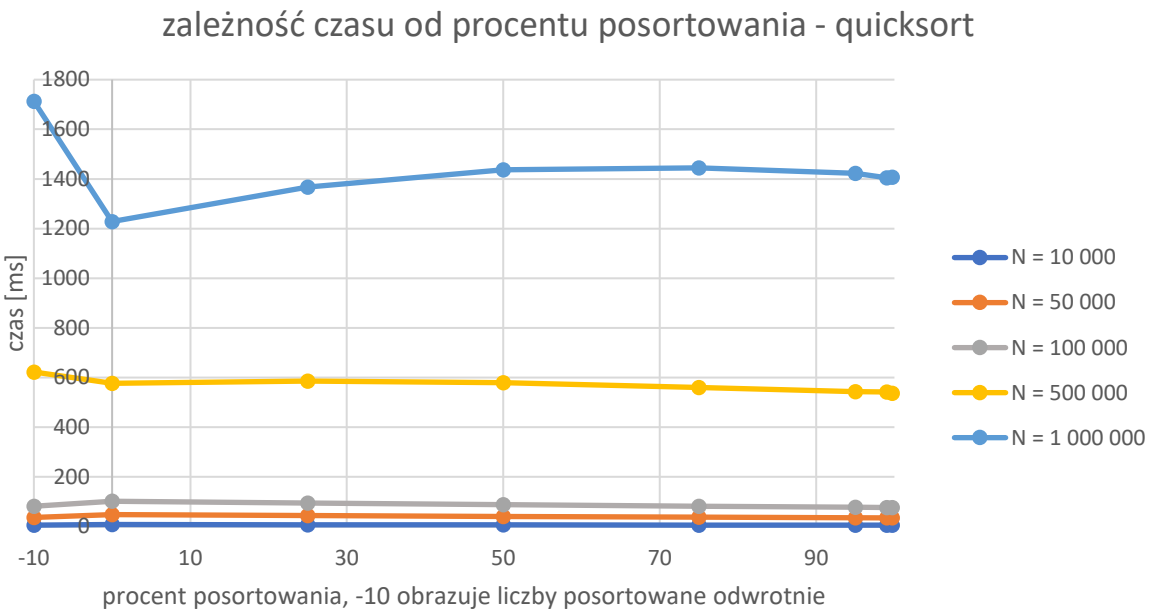


b) Sortowanie szybkie:

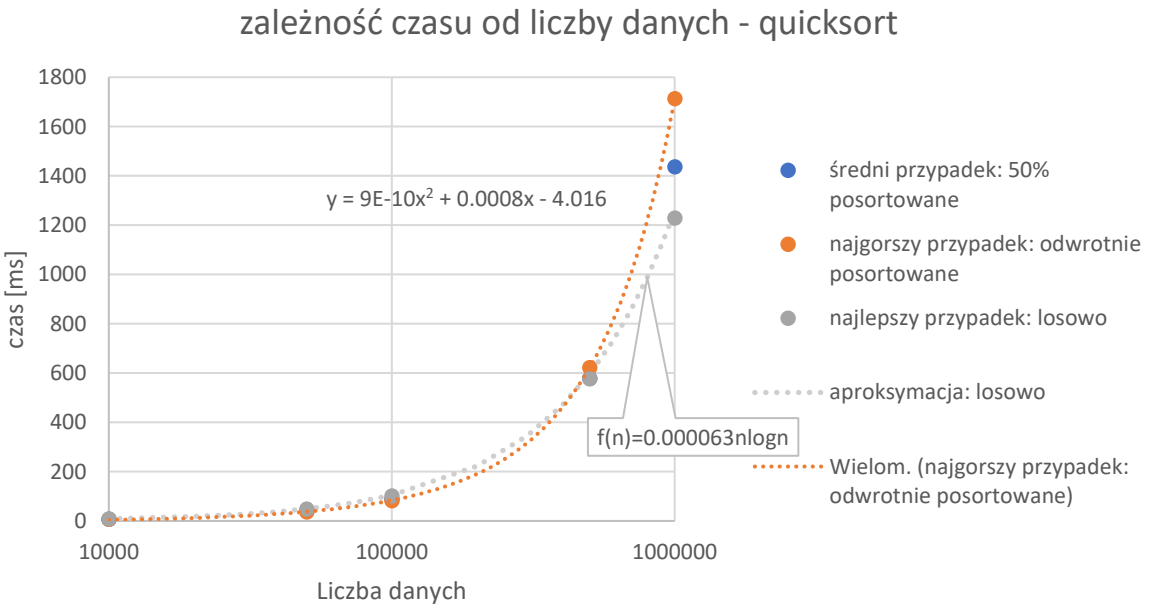
Algorytm zaprojektowany przez Charlesa Hoare'a jest jednym z najszybszych algorytmów. W przeciwności do sortowania przez scalanie nie wymaga tworzenia tablic, jednak również korzysta z metody „dziel i zwyciężaj". Dwa pierwsze i jedyne kroki polegają na: wybraniu pivota, w mojej implementacji jest to losowa liczba z sortowanego zakresu, a następnie podzielenia tablicy na dwie części: liczby mniejsze od pivota i liczby większe od pivota. Te kroki powtarzane są rekursywnie do momentu, w którym nasze podproblemy zawierają pojedyncze liczby. Cała tablica zawiera elementy w odpowiedniej kolejności. Poniżej wyniki badań:

N\stopień posortowania	odwrotnie:	0%	25%	50%	75%	95%	99%	99.7%
10000	6.08	8.10	7.27	6.83	6.08	5.81	5.72	5.71
50000	37.09	47.98	44.04	40.48	37.24	35.09	34.77	34.77
100000	81.50	101.67	94.49	88.14	81.54	77.35	76.53	76.49
500000	622.39	576.66	585.41	578.72	559.46	542.75	541.72	537.02
1000000	1712.84	1228.35	1366.90	1436.90	1444.56	1422.34	1404.92	1406.93

Algorytm ten nie jest naturalny, zbiory zawierające posortowane elementy generalnie sortują się dłużej od losowych:



Ogólna złożoność obliczeniowa algorytmu jest równa $O(nlog_2 n)$, jednak w najgorszym przypadku (-10 na wykresie powyżej), kiedy tablica jest posortowana odwrotnie algorytm przy każdym możliwym porównaniu będzie zamieniał miejscami liczby osiągając złożoność $O(n^2)$ jak widać na wykresie poniżej:



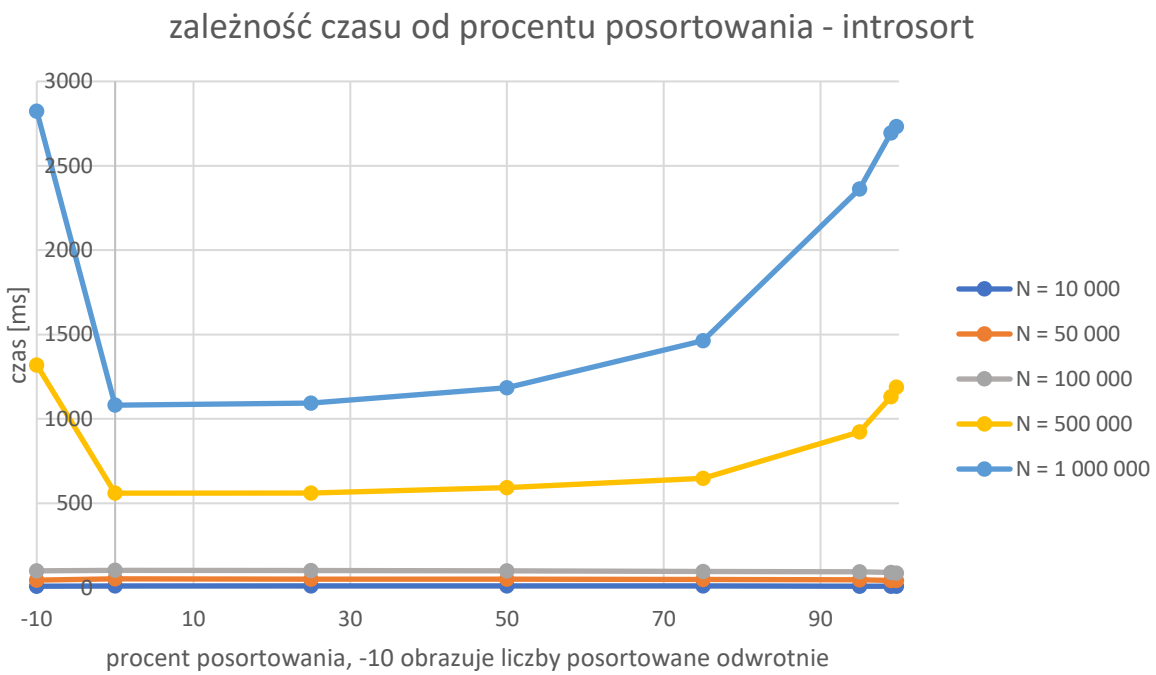
Oprócz faktu posiadania zestawu pesymistycznego, algorytm sortowania szybkiego jest niestabilny, co oznacza, że dwie wartości o tym samym kluczu mogą po posortowaniu zmienić swoją kolejność.

c) Sortowanie introspektywne:

W algorytmie sortowania szybkiego istnieje problem rozmiaru stosu przy dużych wielkościach tablic wejściowych. Zbyt dużo wywołań rekurencyjnych może generować błędy z powodu braku miejsca w pamięci przeznaczonej na stos. Znakomita większość tych wywołań dotyczy małych zbiorów, powiedzmy kilkunastu liczb, z których wszystkie znajdują się bardzo blisko swoich prawidłowych miejsc w zbiorze nadrzędnym. Dla takich podzbiorów wywoływanie sortowania szybkiego jest kosztowne i warto zastąpić je innym wolniejszym algorytmem, który nie jest rekurencyjny. W przypadku sortowania introspektywnego używa się pomocniczego algorytmu sortowania stogowego. Wyniki badań nad algorytmem sortowania introspektywnego podano w poniższej tabeli:

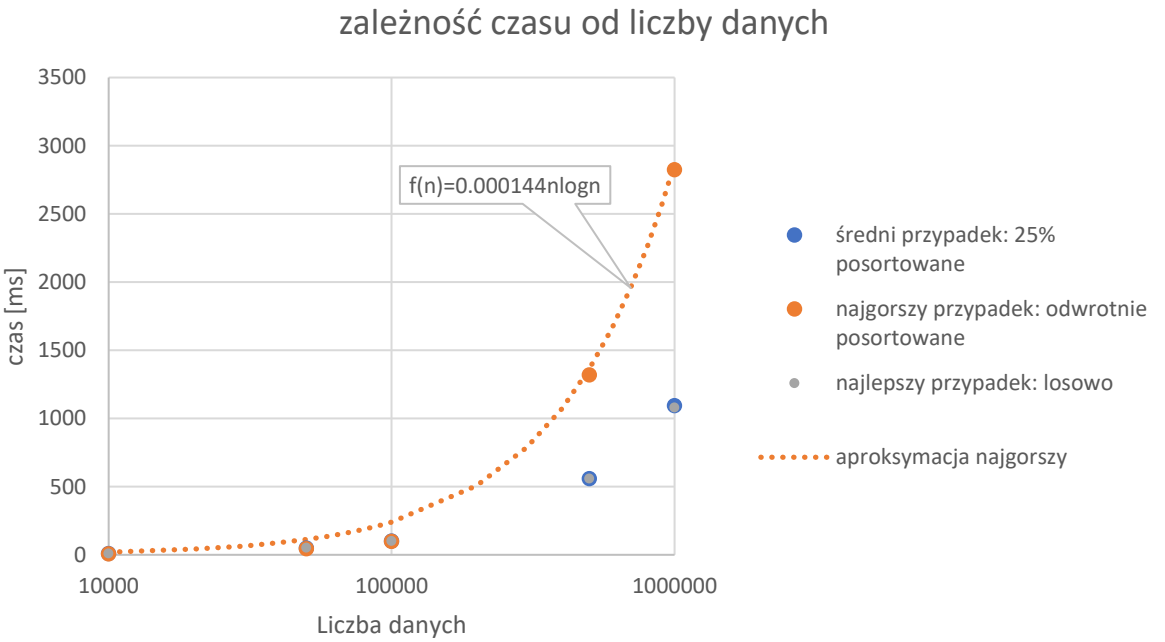
N\stopień posortowania	odwrotnie:	0%	25%	50%	75%	95%	99%	99.7%
10000	7.97	9.18	9.11	9.03	8.81	8.19	7.61	7.53
50000	43.74	51.39	50.41	49.17	47.26	45.04	41.77	41.08
100000	98.84	101.73	100.09	97.94	95.95	93.13	88.62	85.42
500000	1319.17	559.39	559.58	591.88	646.88	922.03	1131.13	1188.49
1000000	2824.66	1080.91	1093.42	1184.69	1463.45	2363.39	2695.28	2733.72

Algorytm introspektywny jest nienaturalny, co obrazuje poniższy wykres:



Widać tutaj wyraźne wydłużenie czasu w przypadku odwrotnie posortowanych liczb (przypadek pesymistyczny), jednak utrzymuje się ono na poziomie wydłużenia w przypadku prawidłowo posortowanej tablicy.

Zaletą algorytmu sortowania introspektywnego jest wyeliminowanie złożoności kwadratowej w najgorszym przypadku. Poprzez ograniczenie współczynnikiem $M = 2log_2 n$ mamy pewność, że algorytm wykona tylko tyle rekurencyjnych wywołań quicksort, a dla pozostałych podzbiorów uruchomi sortowanie stogowe, co daje nam ostateczną złożoność obliczeniową na poziomie $O(nlog_2 n)$ nawet w najgorszym przypadku. Powyższy wywód obrazuje wykres:



3. Literatura/źródła:

a) pl.wikipedia.org
b) pl.khanacademy.org/computing/computer-science/algorithms/
c) „Algorytm Quicksort” Agnieszka Miśkowiec, Tomasz Skucha - Akademia Górniczo-Hutnicza
d) www. algorytm.edu.pl/algorytmy-maturalne
e) eduinf.waw.pl/inf/alg
f) Playlista Youtube „algorytmy” użytkownika kakaboc
g) Prezentacje dr inż. Łukasza Jelenia do kursu Projektowanie algorytmów i metody sztucznej inteligencji