Projektowanie Algorytmów i Metody Sztucznej Inteligencji
Projekt 1

Prowadzący: Marta Emirsajłow

Maksymilian Kadukowski
248974

22 Marzec 2020

1 Wstęp

1.1 Cel ćwiczenia

Poznanie i zaimplementowanie różnych algorytmów sortowania, poznanie zasad notacji dużego O.

2 Badane algorytmy

2.1 Sortowanie przez scalanie

Sortowanie przez scalanie (z ang. *Merge sort*) jest algorytmem wymyślonym przez John'a von Neumann'a w 1945. Jest algorytmem działającym na zasadzie dziel-i-zwyciężaj - problem dzielony jest rekursywnie na mniejsze problemy takiego samego typu do momentu, kiedy rozwiązanie jest banalne (np. sortowanie tablicy o rozmiarze 1). Zaletą sortowania przez scalanie jest jego wydajność dla najgorszego przypadku - zarówno dla niego i najlepszego wynosi ona $O(n \log n)$.

Algorytm można przedstawić jako dwie procedury, odpowiednio sam *merge-sort* oraz *merge*. Procedura *merge-sort* dzieli tablicę na połowy, rekursywnie wywołuję siebie na nich oraz później wywołuje procedurę *merge* aby połączyć 2 posortowane tablice w jedną.

2.2 Sortowanie szybkie

Sortowanie szybkie (z ang. *Quicksort*) jest algorytmem wymyślonym przez Tony'ego Hoare w 1959. Podobnie jak sortowanie przez scalanie jest algorytmem działającym na zasadzie dziel-i-zwyciężaj. Zaletą szybkiego sortowania jest wydajność dla średniego przypadku - w przypadku dobrej implementacji jest on od 2 do 3 razy szybszy niż sortowania przez scalanie lub kopcowanie (mimo że dla wszystkich tych algorytmów notacja dużego O dla średniego przypadku wynosi tyle samo - $O(n \log n)$). Natomiast najgorszy przypadek daje wydajność na poziomie $O(n^2)$, lecz zdarza się to rzadko i jest uwarunkowany algorytmem wyboru klucza.

Algorytm można znowu przedstawić jako dwie procedury, samo *quick-sort* oraz *partition-with-key*. Procedura *partition-with-key* "dzieli" tablice na dwie podtablice zawierające odpowiednio mniejsze i większe elementy od danego klucza. Procedura *quick-sort* odpowiednio wybiera klucz, dzieli tablicę za pomocą *partition-with-key* i rekursywnie wywołuje siebie na tablicach wynikowych.

2.3 Sortowanie introspektywne

Sortowanie introspektywne (z ang. *Introsort*) jest algorytmem wymyślonym przez David'a Musser'a. Jest swoistą hybrydą pomiędzy sortowaniem szybkim oraz przez kopcowanie. Idea takiego sortowanie polega na usunięciu największej ilości wywołań rekursywnych dla quicksort'a występujących dla stosunkowo małych tablic - w sortowaniu introspektywnym występuję 'współczynnik maksymalnej rekursji' który wpływa na maksymalna ilość wywołań rekursywnych. Sortowanie introspektywne eliminuje najgorszy przypadek dla quicksort'a równocześnie nie usuwając jego lepszej wydajności, co w konsekwencji daje wydajność $O(n \log n))$ dla najlepszego i najgorszego przypadku. Tradycyjnie jako współczynnik największej rekursji używa się wyniku wyrażenia $\log n \cdot 2$, gdzie n to długość danej tablicy.

Algorytm, podobnie jak w przypadku szybkiego sortowania, używa procedur *partition-with-key*, *intro-sort* oraz *heap-sort*(która wykonuje sortowanie przez kopcowanie). Procedura *intro-sort* polega na :

- Jeśli nie przekroczyliśmy limitu rekursji ⇒ dzielimy daną tablicę na dwie używając *partition-with-key* i rekursywnie wykonując *intro-sort* na nich.
- Jeśli przekroczyliśmy limit rekursji ⇒ użyć *heap-sort* na danej tablicy.

3 Wyniki

Testy przeprowadzone zostały na komputerze prywatnym. Do mierzenia wydajności zastosowana została biblioteka `chrono` języka C++. Jako wynik testu podana została średnia czasu ze 100 sortowań.

Tablicę posortowaną w $X\%$ interpretuje jako tablicę, której początkowe $X\% \cdot size$ elementów jest posortowane, reszta tablicy jest całkowicie losowa.

Warto zaznaczyć, że w celu uniknięcia najgorszego przypadku dla QuickSort'a **klucz** wybierany jest jako środkowy element danej podtablicy.

3.1 Tabele

Legenda dla nazwy testu - "`dlugosc_testu` - `stopien_posortowania`" (`rvrs` dla tablicy posortowanej na odwrot)

3.1.1 MergeSort

Test	Średnia [ms]
10000 - 0%	1.02
10000 - 25%	0.91
10000 - 50%	0.76
10000 - 75%	0.66
10000 - 90%	0.56
10000 - 99%	0.53
10000 - 99.7%	0.52
10000 - revrs	0.53
50000 - 0%	5.33
50000 - 25%	4.83
50000 - 50%	4.11
50000 - 75%	3.40
50000 - 90%	2.94
50000 - 99%	2.83
50000 - 99.7%	3.40
50000 - revrs	2.68
100000 - 0%	12.16
100000 - 25%	11.70
100000 - 50%	9.23
100000 - 75%	6.94
100000 - 90%	5.98
100000 - 99%	6.13
100000 - 99.7%	6.24
100000 - revrs	6.04
500000 - 0%	72.99
500000 - 25%	64.25
500000 - 50%	46.73
500000 - 75%	37.25
500000 - 90%	32.16
500000 - 99%	29.32
500000 - 99.7%	29.10
500000 - revrs	28.47
1000000 - 0%	139.64
1000000 - 25%	113.94
1000000 - 50%	95.24
1000000 - 75%	78.94
1000000 - 90%	65.80
1000000 - 99%	60.31
1000000 - 99.7%	59.37
1000000 - revrs	57.03

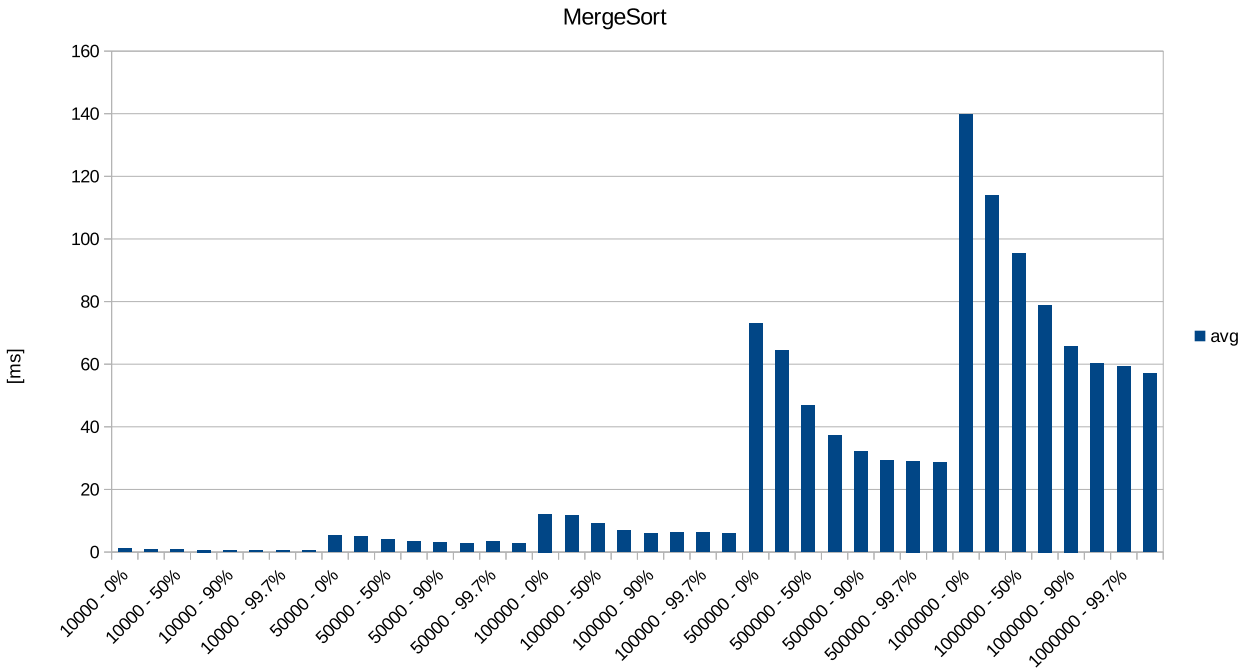
3.1.2 IntroSort

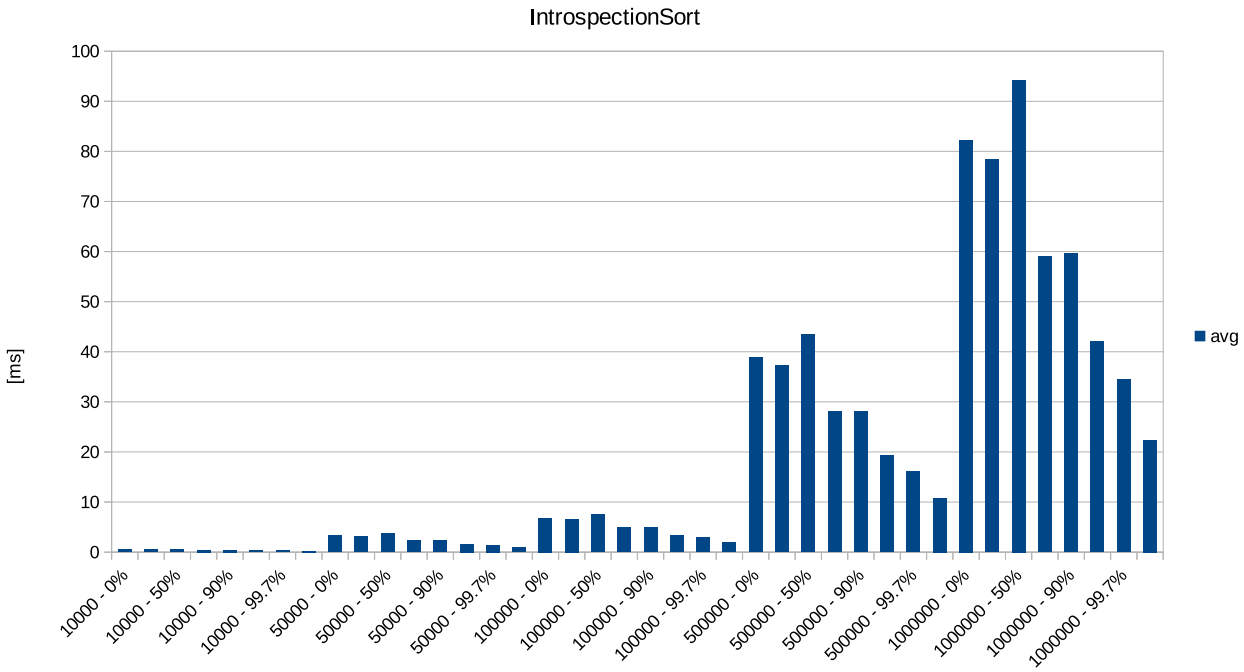
Test	Średnia [ms]
10000 - 0%	0.56
10000 - 25%	0.55
10000 - 50%	0.58
10000 - 75%	0.41
10000 - 90%	0.41
10000 - 99%	0.31
10000 - 99.7%	0.29
10000 - revrs	0.19
50000 - 0%	3.27
50000 - 25%	3.14
50000 - 50%	3.68
50000 - 75%	2.33
50000 - 90%	2.33
50000 - 99%	1.61
50000 - 99.7%	1.44
50000 - revrs	0.97
100000 - 0%	6.81
100000 - 25%	6.59
100000 - 50%	7.52
100000 - 75%	4.91
100000 - 90%	4.89
100000 - 99%	3.26
100000 - 99.7%	2.89
100000 - revrs	1.97
500000 - 0%	38.81
500000 - 25%	37.26
500000 - 50%	43.44
500000 - 75%	28.06
500000 - 90%	28.01
500000 - 99%	19.35
500000 - 99.7%	16.05
500000 - revrs	10.78
1000000 - 0%	82.25
1000000 - 25%	78.44
1000000 - 50%	94.26
1000000 - 75%	59.08
1000000 - 90%	59.57
1000000 - 99%	41.98
1000000 - 99.7%	34.51
1000000 - revrs	22.39

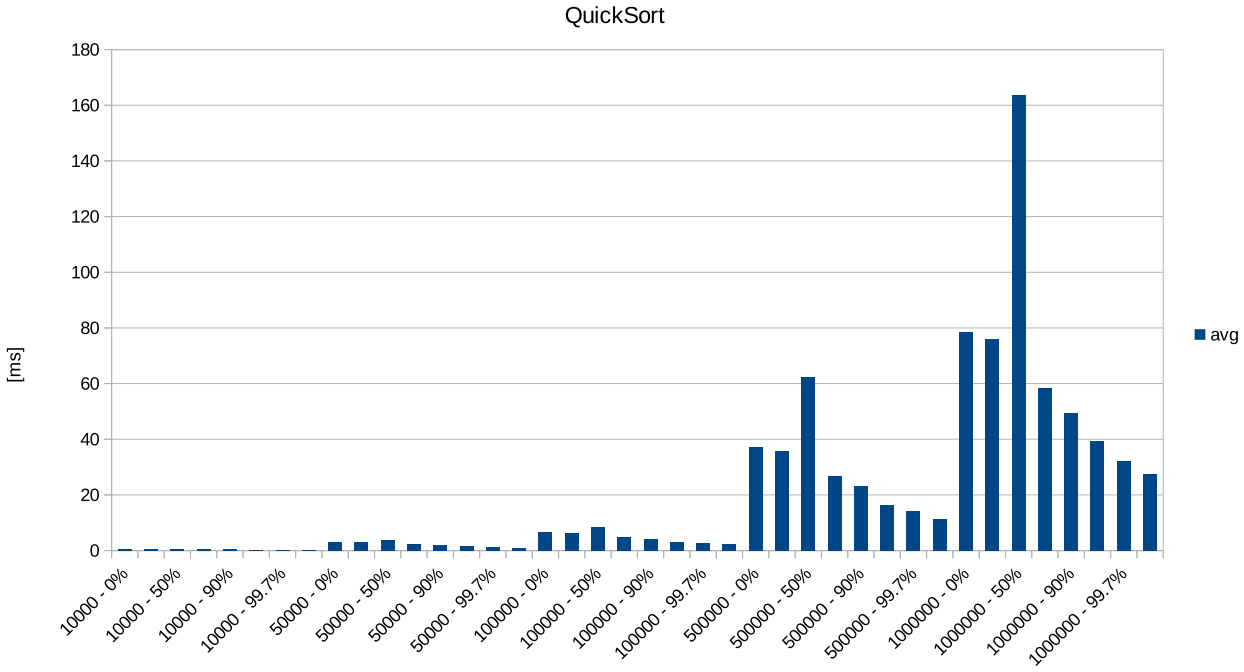
3.1.3 QuickSort

Test	Średnia [ms]
10000 - 0%	0.55
10000 - 25%	0.53
10000 - 50%	0.55
10000 - 75%	0.42
10000 - 90%	0.37
10000 - 99%	0.31
10000 - 99.7%	0.30
10000 - revrs	0.21
50000 - 0%	3.12
50000 - 25%	3.01
50000 - 50%	3.68
50000 - 75%	2.32
50000 - 90%	2.01
50000 - 99%	1.47
50000 - 99.7%	1.41
50000 - revrs	1.05
100000 - 0%	6.60
100000 - 25%	6.38
100000 - 50%	8.56
100000 - 75%	4.77
100000 - 90%	4.18
100000 - 99%	2.95
100000 - 99.7%	2.75
100000 - revrs	2.18
500000 - 0%	37.20
500000 - 25%	35.93
500000 - 50%	62.28
500000 - 75%	26.89
500000 - 90%	23.31
500000 - 99%	16.22
500000 - 99.7%	14.29
500000 - revrs	11.44
1000000 - 0%	78.63
1000000 - 25%	75.86
1000000 - 50%	163.75
1000000 - 75%	58.55
1000000 - 90%	49.53
1000000 - 99%	39.27
1000000 - 99.7%	32.31
1000000 - revrs	27.34

3.2 Wykresy


MergeSort
[ms]
160
140
120
100
80
60
40
20
0
10000 - 0%
10000 - 50%
10000 - 90%
10000 - 99.7%
50000 - 0%
50000 - 50%
50000 - 90%
50000 - 99.7%
100000 - 0%
100000 - 50%
100000 - 90%
100000 - 99.7%
500000 - 0%
500000 - 50%
500000 - 90%
500000 - 99.7%
1000000 - 0%
1000000 - 50%
1000000 - 90%
1000000 - 99.7%
avg


IntrospectionSort
[ms]
100
90
80
70
60
50
40
30
20
10
0
10000 - 0%
10000 - 50%
10000 - 90%
10000 - 99.7%
50000 - 0%
50000 - 50%
50000 - 90%
50000 - 99.7%
100000 - 0%
100000 - 50%
100000 - 90%
100000 - 99.7%
500000 - 0%
500000 - 50%
500000 - 90%
500000 - 99.7%
1000000 - 0%
1000000 - 50%
1000000 - 90%
1000000 - 99.7%
avg



4 Wnioski

Wyniki testów w większości sprawdzają się z przewidywaniami. Warto zaznaczyć, że dla IntroSort'a i QuickSort'a posortowane w 50% - możliwą przyczyną jest sposób wybrania klucza, które te dwa algorytmy dzielą. Jako klucz w pierwszym podziale wybrany najpewniej został ostatni element posortowanej części, który jest na pewno blisko maksymalnemu elementowi całego zbioru. Prowadzi to do bardzo nierównego pierwszego podziału, co z kolei przedłuża działanie algorytmu.

Bibliografia

[1] Geeks for geeks: Algorithms,
`https://www.geeksforgeeks.org/fundamentals-of-algorithms/`

[2] Strony Wikipedii na temat sortowań,
`https://en.wikipedia.org/wiki/Merge_sort`
`https://en.wikipedia.org/wiki/Introsort`
`https://en.wikipedia.org/wiki/Quicksort`