Nikodem Iwin (252928)

27.03.2021

Projektowanie algorytmów i metod sztucznej inteligencji

Mgr inż. Marta Emirsajłow

Poniedziałek 13.15-15.00

Projekt1

Sprawozdanie

# Wstęp

Jako pierwszy projekt z przedmiotu Projektowanie algorytmów i metod sztucznej inteligencji należało zaimplementować odpowiednie algorytmy sortowania (przez kopcowanie, przez scalanie, Shella, quicksort) oraz dokonać analizy ich efektywności. W moim przypadku sortowanie jest od najmniejszego do największego elementu (sortowanie niemalejące).

Sortowaniom zostały poddane tablice typu całkowitego o następujących rozmiarach:

* 100 tablic o rozmiarze 10.000
* 100 tablic o rozmiarze 50.000
* 100 tablic o rozmiarze 100.000
* 100 tablic o rozmiarze 500.000
* 100 tablic o rozmiarze 1.000.000.

Elementy tablicy były tworzone w sposób losowy z zakresu od 0 do 1.000.000 i w zależności od warunku ich kolejność była następująca :

* Wszystkie elementy ułożone losowo.
* 25%, 50%, 75%, 95%, 99%, 99,7% początkowych elementów jest posortowane.
* Wszystkie elementy są uporządkowane od największego do najmniejszego.

# Opis algorytmów

## Algorytm sortowania przez kopcowanie (heap sort)

Algorytm bazuje na utworzeniu kopca binarnego. Założeniem kopca w przypadku sortowania rosnącego jest to by na samej górze kopca był element największy, a poniżej niego kolejno mniejsze. W przypadku gdy nad któryś elementem jest element większy niż element niżej elementy są zamieniane miejscami. Gdy uzyskamy część posortowaną, ta część jest przenoszona do tabeli jako część posortowana i nie jest już brana pod uwagę. Cała procedura jest powtarzana tak długo aż wszystkie elementy tablicy oryginalnej zostaną posortowane i trafią odpowiednio ułożone z powrotem do tablicy

## Algorytm sortowania przez scalanie (merge sort)

Sortowana tablica dzielona jest na mniejsze podtablice. Ta czynność powtarzana jest rekurencyjnie, aż do momentu otrzymania tablic jednowymiarowych. Następnie te podtablice są porównywane między sobą (lewa z prawą) i odpowiednio scalane. Scalone kawałki są łączone aż do uzyskania posortowanej tablicy oryginalnej.

## Algorytm sortowania Shella (Shell sort)

Algorytm bazuje na tablicy oryginalnej i polega na porównywaniu elementów oddalonych o określoną odległość. Wywołania sortowania następuje tyle razy ile założono w implementacji kodu. W moim przypadku ilość wywoływania algorytmu sortowania następuje zależnie od rozmiaru tablicy. Odległości są wyliczane w pętli for (int i = 10; i <= n; i=i\*10) , gdzie „n” jest rozmiarem tablicy, a odstępy w sortowaniu wyznaczone są wzorem „n/i”.

## Algorymt sortowania szybkiego (quicksort)

Sortowanie opiera się na porównywaniu elementów względem wybranego tzw. Piwot (element osiowy). Algorytm szuka dla jakiego indeksu po lewej stornie od piwota jest element większy (szukając od początku) i dla jakiego indeksu pop prawej stornie od piwota jest element mniejszy (szukając od końca). Gdy znajdzie takie elementy to zamienia je miejscami, o ile indeks lewy jest mniejszy lub równy indeksowi prawemu (warunek dla sortowania niemalejącego). Następnie przesuwamy naszą oś w zależności od tego jak wyglądały indeksy względem początkowej tablicy i powtarzamy operację. Algorytm powtarza się tak długo aż otrzymamy posortowaną oryginalną tablicę.

# Omówienie złożoności obliczeniowej algorytmów dla przypadków średnich i najgorszych

## Algorytm sortowania przez kopcowanie (heap sort)

Algorytm sortowania przez kopcowanie można przedstawić przy pomocy 4 kroków:

1. Zbuduj kopiec.
2. Zamień element wyżej z niższym jeżeli wyższy>niższy.
3. Wyciągnij posortowany element i przesuń elementy.
4. Wróć do kroku II dopóki nie skończą się elementy.

Budowa kopca trwa *O(n)*. Krok III, czyli wyciagnięcie posortowanego elementu trwa *O(log n)* i jest wywoływane n razy czyli mamy *O(n log n).* Sumując budowanie i wyciąganie mamy:

*O(n + n log n)* = *O(n log n)*

Algorytm działa niezależnie od wartości liczb, gdyż notacja *O(n)* jest uwarunkowana od ich ilości.

Dlatego przypadek średni jak i najgorszy będą miały złożoność *O(n log n).*

## Algorytm sortowania przez scalanie (merge sort)

W wyniku algorytmu sortowania przez scalanie otrzymujemy drzewo binarne o wielkości *log n*.

By scalić poszczególne tablice potrzeba *n\*c* czasu, który zależy od zmiennej zależnej od komputera *c.*

Zatem z powyższego czas potrzebny na posortowanie n-elementowej tablicy to czas na scalenie dwóch ciągów n/2-elementowych oraz czas potrzebny na ich sortowanie czyli *n*. Z tego wynika że złożoność obliczeniowa wynosi *O(n log n).*

Przypadki najlepszy, średni i najgorszy rozróżnia tylko to jakie liczby chcemy sortować, jednakże złożoność będzie nadal taka sama.

## Algorytm sortowania Shella (Shell sort)

Algorytm sortowania Shella ma złożoność zależną od ciągu odstępów. W moim przypadku określenie złożoność obliczeniowej jest bardzo trudne. Można by powiedzieć, że shell sort to sortowanie przez wstawianie wykonane „x” razy na podciągach. Dlatego swoją złożoność określam jako złożoność obliczeniową *O(n2).* Przypadek średni jak i najgorszy mają tę samą złożoność, gdyż wykonujemy zawsze tyle samo porównań, bez względu na to jakie mamy dane. Nikt jeszcze nie wykazał złożoności sortowania Shella, ponieważ zależy ona od doboru odstępu między komórkami danych, które sortujemy. Optymalizacja algorytmu odbywa się poprzez dobieranie odpowiedniego odstępu zależnego od zbioru danych który chcemy posortować.

## Algorymt sortowania szybkiego (quicksort)

Przypadek średni mówi o tym że prawdopodobieństwo znalezienia liczby jest przedstawione za pomocą równomiernego rozkładu prawdopodobieństwa. Ponieważ algorytm opiera się głównie na jednej pętli i rekurencji to w notacji „Wielkie O” jego złożoność wynosi *O(n log n)*,

W przypadku najgorszym, czyli że trafiamy zawsze na element najmniejszy względem którego szukamy to złożoność obliczeniowa wynosi *O(n2).* Wynika to z faktu, iż otrzymujemy ciąg rekurencyjny *T(n)=n-1+T(n-1)* co nam daje T(n).

# Omówienie przebiegu eksperymentów

## Warunki przebiegu eksperymentów

Całość programu została napisana w Visual Studio Code. Eksperymenty zostały wykonane na moim prywatnym komputerze o procesorze Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz 3.90 GHz i 10,0 GB pamięci RAM.

Czas liczony był dzięki bibliotece zewnętrznej „chrono”. Pomiar odbywał się poprzez odmierzanie i następnie sumowanie odcinków czasowych potrzebnych na samo sortowanie tabeli. Jednostka czasu to milisekunda.

Każdy eksperyment odbył się na tym samym kodzie przy podobnym użyciu procesora i pamięci. Wszystkie warunki pracy programów były określane bezpośrednio z jego menu.

Poniżej w tabelach zaprezentowane są wyniki otrzymanych czasów (w milisekundach). Pierwsze tabele określają wyniki kolejnych eksperymentów( w narożniku oznaczono rozmiar tablicy), a ostania zbiór wyników średnich (średnie arytmetyczne) dla 100 tablic o określonych rozmiarach i zawartości:

* 0 – wszystkie elementy losowe
* 25% określa że początkowe 25% elementów jest posortowane (analogicznie reszta procentów)
* -100% oznacza że elementy są posortowane nierosnąco.

## Algorytm sortowania przez kopcowanie (heap sort)

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **10 000** | **1** | **2** | **3** | **4** | **5** |  | **50 000** | **1** | **2** | **3** | **4** | **5** |
| **0** | 138 | 135 | 144 | 129 | 145 |  | **0** | 815 | 824 | 813 | 822 | 818 |
| **25%** | 142 | 136 | 143 | 147 | 135 |  | **25%** | 773 | 765 | 771 | 773 | 764 |
| **50%** | 122 | 119 | 126 | 133 | 122 |  | **50%** | 725 | 723 | 720 | 719 | 713 |
| **75%** | 108 | 108 | 117 | 109 | 114 |  | **75%** | 646 | 656 | 648 | 651 | 648 |
| **95%** | 109 | 110 | 104 | 100 | 103 |  | **95%** | 589 | 585 | 579 | 580 | 588 |
| **99%** | 98 | 101 | 99 | 102 | 101 |  | **99%** | 579 | 570 | 574 | 561 | 569 |
| **99,7%** | 99 | 102 | 97 | 99 | 104 |  | **99,7%** | 564 | 569 | 574 | 571 | 566 |
| **-100%** | 145 | 146 | 148 | 143 | 151 |  | **-100%** | 844 | 841 | 842 | 838 | 850 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **100 000** | **1** | **2** | **3** | **4** | **5** |  | **500 000** | **1** | **2** | **3** | **4** | **5** |
| **0** | 1752 | 1736 | 1742 | 1749 | 1739 |  | **0** | 9270 | 9266 | 9300 | 9311 | 9272 |
| **25%** | 1640 | 1651 | 1642 | 1649 | 1647 |  | **25%** | 8528 | 8517 | 8557 | 8552 | 8528 |
| **50%** | 1522 | 1514 | 1530 | 1525 | 1531 |  | **50%** | 7580 | 7620 | 7590 | 7625 | 7538 |
| **75%** | 1390 | 1385 | 1392 | 1387 | 1372 |  | **75%** | 7287 | 7284 | 7139 | 7228 | 7240 |
| **95%** | 1213 | 1218 | 1213 | 1225 | 1233 |  | **95%** | 6436 | 6466 | 6647 | 6465 | 6404 |
| **99%** | 1193 | 1192 | 1186 | 1183 | 1189 |  | **99%** | 6232 | 6242 | 6192 | 6132 | 6232 |
| **99,7%** | 1198 | 1180 | 1189 | 1185 | 1193 |  | **99,7%** | 6092 | 6204 | 6091 | 6229 | 6042 |
| **-100%** | 1773 | 1772 | 1781 | 1787 | 1806 |  | **-100%** | 9377 | 9484 | 9438 | 9353 | 9520 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **1 000 000** | **1** | **2** | **3** | **4** | **5** |
| **0** | 20231 | 20138 | 20201 | 20285 | 20149 |
| **25%** | 18858 | 18893 | 18913 | 18926 | 18776 |
| **50%** | 17569 | 17713 | 17621 | 17587 | 17649 |
| **75%** | 15493 | 15524 | 15679 | 15538 | 15519 |
| **95%** | 13538 | 13359 | 13650 | 13363 | 13124 |
| **99%** | 13083 | 13099 | 13061 | 13177 | 13052 |
| **99,7%** | 12969 | 12760 | 12823 | 12899 | 12901 |
| **-100%** | 19879 | 19970 | 19973 | 19922 | 19872 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **heap** | **10 000** | **50 000** | **100 000** | **500 000** | **1 000 000** |
| **0** | 138 | 818 | 1 744 | 9 284 | 20 201 |
| **25%** | 140,6 | 769,2 | 1645,8 | 8536,4 | 18873,2 |
| **50%** | 124,4 | 720 | 1524,4 | 7590,6 | 17627,8 |
| **75%** | 111,2 | 649,8 | 1385,2 | 7235,6 | 15550,6 |
| **95%** | 105,2 | 584,2 | 1220,4 | 6483,6 | 13406,8 |
| **99%** | 100,2 | 570,6 | 1188,6 | 6206 | 13094,4 |
| **99,7%** | 100,2 | 568,8 | 1189 | 6131,6 | 12870,4 |
| **-100%** | 146,6 | 843 | 1783,8 | 9434,4 | 19923,2 |

## Algorytm sortowania przez scalanie (merge sort)

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **10 000** | **1** | **2** | **3** | **4** | **5** |  | **50 000** | **1** | **2** | **3** | **4** | **5** |
| **0** | 228 | 215 | 213 | 226 | 223 |  | **0** | 1186 | 1198 | 1177 | 1184 | 1192 |
| **25%** | 208 | 207 | 207 | 220 | 213 |  | **25%** | 1100 | 1125 | 1100 | 1106 | 1119 |
| **50%** | 191 | 188 | 188 | 192 | 196 |  | **50%** | 1016 | 1012 | 1027 | 1018 | 1014 |
| **75%** | 176 | 173 | 176 | 173 | 175 |  | **75%** | 917 | 930 | 921 | 918 | 917 |
| **95%** | 159 | 163 | 169 | 163 | 166 |  | **95%** | 835 | 839 | 827 | 838 | 839 |
| **99%** | 161 | 164 | 157 | 159 | 161 |  | **99%** | 829 | 843 | 825 | 838 | 828 |
| **99,7%** | 162 | 163 | 160 | 155 | 163 |  | **99,7%** | 824 | 834 | 829 | 836 | 836 |
| **-100%** | 156 | 162 | 157 | 172 | 168 |  | **-100%** | 836 | 837 | 842 | 838 | 823 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **100 000** | **1** | **2** | **3** | **4** | **5** |  | **500 000** | **1** | **2** | **3** | **4** | **5** |
| **0** | 2433 | 2456 | 2446 | 2434 | 2443 |  | **0** | 11912 | 11905 | 11903 | 11981 | 11930 |
| **25%** | 2273 | 2269 | 2272 | 2262 | 2268 |  | **25%** | 10990 | 10969 | 11041 | 10853 | 10883 |
| **50%** | 2089 | 2098 | 2109 | 2073 | 2066 |  | **50%** | 9641 | 9875 | 9662 | 9942 | 9888 |
| **75%** | 1883 | 1884 | 1876 | 1882 | 1890 |  | **75%** | 9237 | 9193 | 9258 | 9312 | 9257 |
| **95%** | 1718 | 1730 | 1718 | 1729 | 1728 |  | **95%** | 8726 | 8650 | 8736 | 8667 | 8826 |
| **99%** | 1690 | 1688 | 1701 | 1688 | 1707 |  | **99%** | 8568 | 8501 | 8660 | 8555 | 8459 |
| **99,7%** | 1691 | 1684 | 1697 | 1700 | 1695 |  | **99,7%** | 8694 | 8525 | 8639 | 8618 | 8512 |
| **-100%** | 1712 | 1709 | 1714 | 1690 | 1720 |  | **-100%** | 8566 | 8610 | 8576 | 8488 | 8564 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **1 000 000** | **1** | **2** | **3** | **4** | **5** |
| **0** | 25061 | 25009 | 24955 | 24949 | 25107 |
| **25%** | 23289 | 23358 | 23370 | 23291 | 23303 |
| **50%** | 21879 | 21762 | 21779 | 21802 | 21818 |
| **75%** | 19647 | 19700 | 19711 | 19936 | 19704 |
| **95%** | 18201 | 18191 | 18235 | 18200 | 18206 |
| **99%** | 18069 | 18025 | 17996 | 17945 | 17825 |
| **99,7%** | 17905 | 17931 | 17847 | 17846 | 17881 |
| **-100%** | 17646 | 17613 | 17608 | 17581 | 17575 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **merge** | **10 000** | **50 000** | **100 000** | **500 000** | **1 000 000** |
| **0** | 221 | 1 187 | 2 442 | 11 926 | 25 016 |
| **25%** | 211 | 1110 | 2268,8 | 10947,2 | 23322,2 |
| **50%** | 191 | 1017,4 | 2087 | 9801,6 | 21808 |
| **75%** | 174,6 | 920,6 | 1883 | 9251,4 | 19739,6 |
| **95%** | 164 | 835,6 | 1724,6 | 8721 | 18206,6 |
| **99%** | 160,4 | 832,6 | 1694,8 | 8548,6 | 17972 |
| **99,7%** | 160,6 | 831,8 | 1693,4 | 8597,6 | 17882 |
| **-100%** | 163 | 835,2 | 1709 | 8560,8 | 17604,6 |

## Algorytm sortowania Shella (Shell sort)

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **10 000** | **1** | **2** | **3** | **4** | **5** |  | **50 000** | **1** | **2** | **3** | **4** | **5** |
| **0** | 164 | 170 | 176 | 159 | 166 |  | **0** | 1077 | 1072 | 1058 | 1080 | 1048 |
| **25%** | 154 | 157 | 149 | 163 | 162 |  | **25%** | 1036 | 1048 | 1039 | 1052 | 1042 |
| **50%** | 152 | 153 | 151 | 156 | 145 |  | **50%** | 977 | 975 | 969 | 978 | 984 |
| **75%** | 142 | 135 | 129 | 136 | 130 |  | **75%** | 893 | 896 | 889 | 897 | 892 |
| **95%** | 106 | 107 | 104 | 99 | 103 |  | **95%** | 722 | 716 | 725 | 712 | 719 |
| **99%** | 86 | 80 | 81 | 83 | 81 |  | **99%** | 567 | 565 | 564 | 575 | 570 |
| **99,7%** | 59 | 63 | 54 | 60 | 58 |  | **99,7%** | 407 | 424 | 424 | 410 | 413 |
| **-100%** | 61 | 62 | 63 | 58 | 65 |  | **-100%** | 368 | 371 | 363 | 368 | 365 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **100 000** | **1** | **2** | **3** | **4** | **5** |  | **500 000** | **1** | **2** | **3** | **4** | **5** |
| **0** | 2445 | 2433 | 2450 | 2439 | 2444 |  | **0** | 15214 | 15234 | 15210 | 15229 | 15229 |
| **25%** | 2312 | 2320 | 2321 | 2318 | 2307 |  | **25%** | 14600 | 14618 | 14625 | 14605 | 14622 |
| **50%** | 2163 | 2208 | 2194 | 2164 | 2202 |  | **50%** | 14033 | 14023 | 14015 | 13839 | 13885 |
| **75%** | 1958 | 2010 | 1943 | 2013 | 2189 |  | **75%** | 12628 | 12634 | 12598 | 12596 | 12560 |
| **95%** | 1594 | 1590 | 1609 | 1587 | 1601 |  | **95%** | 10389 | 10448 | 10744 | 10864 | 10925 |
| **99%** | 1286 | 1308 | 1308 | 1313 | 1303 |  | **99%** | 5523 | 5558 | 5574 | 5535 | 5589 |
| **99,7%** | 994 | 1008 | 1006 | 1003 | 999 |  | **99,7%** | 3633 | 3630 | 3613 | 3609 | 3635 |
| **-100%** | 810 | 825 | 776 | 801 | 794 |  | **-100%** | 4027 | 4142 | 4121 | 4057 | 4093 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **1 000 000** | **1** | **2** | **3** | **4** | **5** |
| **0** | 242530 | 242673 | 244063 | 242543 | 247361 |
| **25%** | 224389 | 224402 | 224357 | 224399 | 224457 |
| **50%** | 168152 | 168017 | 168234 | 167995 | 168026 |
| **75%** | 124696 | 124874 | 124651 | 124573 | 123739 |
| **95%** | 54907 | 54912 | 54923 | 54891 | 54872 |
| **99%** | 29881 | 29625 | 29698 | 29699 | 29714 |
| **99,7%** | 15912 | 15915 | 15910 | 15893 | 15918 |
| **-100%** | 9064 | 9076 | 9073 | 9101 | 9084 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **shell** | **10 000** | **50 000** | **100 000** | **500 000** | **1 000 000** |
| **0** | 167 | 1 067 | 2 442 | 15 223 | 243 834 |
| **25%** | 157 | 1043,4 | 2315,6 | 14614 | 224400,8 |
| **50%** | 151,4 | 976,6 | 2186,2 | 13959 | 168084,8 |
| **75%** | 134,4 | 893,4 | 2022,6 | 12603,2 | 124506,6 |
| **95%** | 103,8 | 718,8 | 1596,2 | 10674 | 54901 |
| **99%** | 82,2 | 568,2 | 1303,6 | 5555,8 | 29723,4 |
| **99,7%** | 58,8 | 415,6 | 1002 | 3624 | 15909,6 |
| -100% | 61,8 | 367 | 801,2 | 4088 | 9079,6 |

## Algorymt sortowania szybkiego (quicksort)

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **10 000** | **1** | **2** | **3** | **4** | **5** |  | **50 000** | **1** | **2** | **3** | **4** | **5** |
| **0** | 123 | 125 | 122 | 121 | 127 |  | **0** | 718 | 717 | 723 | 719 | 718 |
| **25%** | 125 | 114 | 117 | 123 | 131 |  | **25%** | 699 | 697 | 701 | 700 | 690 |
| **50%** | 156 | 154 | 153 | 155 | 165 |  | **50%** | 1155 | 1159 | 1154 | 1153 | 1145 |
| **75%** | 98 | 101 | 94 | 97 | 100 |  | **75%** | 564 | 563 | 564 | 566 | 559 |
| **95%** | 86 | 81 | 80 | 79 | 84 |  | **95%** | 460 | 455 | 459 | 460 | 453 |
| **99%** | 52 | 65 | 66 | 63 | 60 |  | **99%** | 363 | 356 | 374 | 365 | 352 |
| **99,7%** | 61 | 65 | 54 | 57 | 51 |  | **99,7%** | 341 | 336 | 327 | 338 | 329 |
| **-100%** | 42 | 29 | 35 | 40 | 32 |  | **-100%** | 210 | 217 | 218 | 219 | 217 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **100 000** | **1** | **2** | **3** | **4** | **5** |  | **500 000** | **1** | **2** | **3** | **4** | **5** |
| **0** | 1504 | 1501 | 1507 | 1512 | 1509 |  | **0** | 7649 | 7313 | 7608 | 7524 | 7484 |
| **25%** | 1463 | 1458 | 1458 | 1464 | 1452 |  | **25%** | 7070 | 7065 | 7116 | 7060 | 7160 |
| **50%** | 2507 | 2517 | 2497 | 2476 | 2495 |  | **50%** | 19260 | 19190 | 19189 | 19281 | 19249 |
| **75%** | 1189 | 1184 | 1187 | 1191 | 1198 |  | **75%** | 6092 | 6192 | 6172 | 6149 | 6176 |
| **95%** | 982 | 972 | 977 | 977 | 1007 |  | **95%** | 4860 | 5571 | 5103 | 4915 | 5134 |
| **99%** | 767 | 774 | 766 | 764 | 750 |  | **99%** | 4232 | 4243 | 4258 | 4101 | 4237 |
| **99,7%** | 693 | 696 | 699 | 694 | 714 |  | **99,7%** | 3802 | 3699 | 3735 | 3752 | 3715 |
| **-100%** | 467 | 460 | 471 | 471 | 450 |  | **-100%** | 2555 | 2624 | 2608 | 2547 | 2630 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **1 000 000** | **1** | **2** | **3** | **4** | **5** |
| **0** | 15875 | 15746 | 15783 | 15769 | 15784 |
| **25%** | 15463 | 15556 | 15495 | 15495 | 15519 |
| **50%** | 43588 | 43757 | 43971 | 43711 | 43649 |
| **75%** | 13457 | 13564 | 13479 | 13428 | 13079 |
| **95%** | 10965 | 10936 | 10873 | 10956 | 11031 |
| **99%** | 9107 | 9176 | 9068 | 9058 | 9056 |
| **99,7%** | 8129 | 8192 | 8199 | 8046 | 8088 |
| **-100%** | 5758 | 5726 | 5820 | 5838 | 5921 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **quick** | **10 000** | **50 000** | **100 000** | **500 000** | **1 000 000** |
| **0** | 124 | 719 | 1 507 | 7 516 | 15 791 |
| **25%** | 122 | 697,4 | 1459 | 7094,2 | 15505,6 |
| **50%** | 156,6 | 1153,2 | 2498,4 | 19233,8 | 43735,2 |
| **75%** | 98 | 563,2 | 1189,8 | 6156,2 | 13401,4 |
| **95%** | 82 | 457,4 | 983 | 5116,6 | 10952,2 |
| **99%** | 61,2 | 362 | 764,2 | 4214,2 | 9093 |
| **99,7%** | 57,6 | 334,2 | 699,2 | 3740,6 | 8130,8 |
| **-100%** | 35,6 | 216,2 | 463,8 | 2592,8 | 5812,6 |

## Porównanie czasów średnich

Poniżej znajdują się wykresy porównanych czasów średnich. Na osi pionowej oznaczono czas w milisekundach, a na osi poziomej przedstawiono za pomocą liczb warunek w jakim były ułożone liczby:

* 1 - wszystkie elementy losowe
* 2 – 25% początkowych elementów jest posortowane
* 3 – 50% początkowych elementów jest posortowane
* 4 – 75% początkowych elementów jest posortowane
* 5 – 95% początkowych elementów jest posortowane
* 6 – 99% początkowych elementów jest posortowane
* 7 – 99,7% początkowych elementów jest posortowane
* 8 – elementy są posortowane nierosnąco

# Podsumowanie i wnioski

## Sortowanie Shella – dobór parametru odległości

Zakłada się że sortowanie Shella powinno być szybsze niż p. sortowanie przez scalanie .Z powyższych wykresów widać że dla większości przypadków mój algorytm Shella jest nieefektywny. Powodem takiego działania jest złe określenie odległości między porównywalnymi elementami. W moim algorytmie odległość wynosi „n/i”, gdzie „i” na początku przyjmuje wartość 10 i zwiększa się o pomnożenie siebie przez 10 za każde wywołanie porównania elementów, aż do otrzymania posortowanej oryginalnej tablicy. Gdyby zamienić obecną formę na przykładowo: na początku „i” ma wartość 5 i zwiększa się o pomnożenie przez siebie liczby 2, to program byłby o wiele bardziej efektywny.

Powyższy algorytm został tak zaimplementowany specjalnie by pokazać, że ogólnie przyjęty Shell sort nie musi być szybszy niż sortowanie przez scalanie czy kopcowanie jeżeli zostanie źle dobrany parametr odległości.

**Uwaga**: Dane pomiarowe dla tablic o wielkości 1.000.000 mogą się różnić od rzeczywistych, co jest powodowane procesorem i programami, które mogły wymuszać na nim pracę (np. programy systemu operacyjnego).

## Sortowanie przez scalanie i kopcowanie

Są to dwa wolniejsze algorytmy sortowania, pomijając nieoptymalnie zaimplementowany algorytm sortowania Shella. Implementacja obu sortowań jest trudniejsza niż np. quicksort.

Sortowanie przez scalanie wymaga tworzeniu tablic pomocniczych, które źle zaimplementowane mogą zapełnić bufor pamięci – należy pamiętać o usunięciu ich po wyciagnięciu elementów.

Sortowanie przez kopcowanie jest najtrudniejsze do implementacji gdyż działa na zasadzie drzewa binarnego. Według powyższych tabel i wykresów sortowanie przez kopcowanie trwa dłużej dla tablic posortowanych nierosnąco niż sortowanie przez scalanie. DLACZEGO

## Najlepsze rozwiązanie

Najoptymalniejszym algorytmem jest sortowanie szybkie, gdyż było najszybsze dla większości przypadków. Jest ono również stosunkowo proste do implementacji dlatego gdy chcemy coś posortować to najlepszym wyborem będzie quicksort.

# Literatura

* Instrukcja do Projektu1 zamieszczona na Microsoft Teams w zespole przedmiotowym
* https://pl.wikipedia.org/wiki/Sortowanie\_szybkie
* https://pl.wikipedia.org/wiki/Sortowanie\_Shella
* https://pl.wikipedia.org/wiki/Sortowanie\_przez\_scalanie
* https://pl.wikipedia.org/wiki/Sortowanie\_przez\_kopcowanie
* https://www.geeksforgeeks.org/measure-execution-time-function-cpp/
* Wizualizacja i porównanie algorytmów - https://www.youtube.com/watch?v=ZZuD6iUe3Pc
* https://www.geeksforgeeks.org/merge-sort/
* http://www.algorytm.edu.pl/algorytmy-maturalne/quick-sort.html
* https://www.geeksforgeeks.org/shellsort/
* https://www.geeksforgeeks.org/heap-sort/
* prezentacje Pana Dr inż. Łukasza Jelenia z przedmiotu PAMSI wykład zamieszczane na ePortal-u PWR
* https://pl.wikipedia.org/wiki/Sortowanie\_przez\_wstawianie
* https://dsp.krzaq.cc/post/180/nie-uzywaj-rand-cxx-ma-random/