|  |
| --- |
| WYŻSZA SZKOŁA MENEDŻERSKA  INSTYTUT NAUK O ZARZĄDZANIU I JAKOŚCI  INFORMATYKA |
| Algorytmy i złożoność |

**Praca zaliczeniowa**

**Sortowanie – zadanie 1**

|  |  |  |  |
| --- | --- | --- | --- |
| Skład zespołu/Student  Tomasz Jan Oksiędzki | **Prowadzący zajęcia:**  Marcin Paprzycki | | **Semestr**  **III** |
| **Grupa**  **49DR – A1** | | **Studia**  **Niestacjonarne** |
| Data wykonania  **2020-12-20** | | Data oddania:  2020-12-20 | |

Spis treści

[Wprowadzenie 3](#_Toc58680308)

[Przeprowadzenie eksperymentu 3](#_Toc58680309)

[Założenia 3](#_Toc58680310)

[Eksperyment - opis 3](#_Toc58680311)

[Środowisko 3](#_Toc58680312)

[Opis algorytmów 4](#_Toc58680313)

[Generowanie danych losowych 5](#_Toc58680314)

[Wyznaczenie punktów pomiarowych 5](#_Toc58680315)

[Tabela 3: Wyznaczone przedziału pomiarowe 6](#_Toc58680316)

[Procedura eksperymentu 6](#_Toc58680317)

[Wyniki z eksperymentu 6](#_Toc58680318)

[Wyniki finalne i wnioski 8](#_Toc58680319)

[Załączniki 9](#_Toc58680320)

[Pełne wyniki eksperymentu 9](#_Toc58680321)

[Kod programu 12](#_Toc58680322)

# Wprowadzenie

Przedmiotem tego opracowania jest analiza trzech rodzajów sortowanie i ich zachowania w zależności od zbioru danych wejściowych. Sortowanie, które będzie przedmiotem analizy jest bardzo kosztowną (jeżeli chodzi o złożoność algorytmiczną ale również i czasowo) operacją dość powszechnie wykorzystywaną w praktyce w różnych przedmiotach i zagadnieniach informatycznych. Ze względu na powyższy aspekt bardzo istotne jest dobranie efektywnej metody sortowania do danych tak, aby ograniczyć niepotrzebne zużycie zasobów.

Szczegółowym celem opracowania będzie sprawdzenia jak efektywnie sortowane są zbiory liczb losowych, posortowanych rosnąco oraz malejąco z zastosowaniem trzech algorytmów wyszukiwania: Selection sort, Insertion sort oraz Merge sort (w raporcie działanie opisywane jest jako eksperyment I). Dodatkowym aspektem jest sprawdzenie efektywności metod Insertion sort and Merge sort na posortowanym zbiorze poszerzonym o obserwacje losowe (w raporcie opisany jako eksperyment II).

Po zapoznaniu się z podstawowym opisem każdego z trzech algorytmów sortowania można oczekiwać, iż w eksperymencie I najbardziej efektywną metodą będzie w generalnie metoda merge sort. Aczkolwiek jak to się będzie przestawiało w zależności od danych wejściowych i długości analizowanych danych zostanie przedstawione w dalszej części raportu, zaś w eksperymencie II, można oczekiwać, iż wystarczy wydłużenie posortowanego wektora o niewielką część losowych liczb, co będzie skutkowało znacznym wydłużeniem czasu Insertion sort.

# Przeprowadzenie eksperymentu I

## Założenia

Do przeprowadzenia eksperymentu przyjęto następujące założenia:

* Zbiór danych składa się z losowo wygenerowanych nieposortowanych liczb zmiennoprzecinkowych z przedziału 0-1.
* Algorytmy sortujące mają za zadanie dokonać sortowania rosnąco zbioru wejściowego.
* Porównane zostaną trzy algorytmy sortowania:
  + Selection Sort
  + Insertion Sort
  + Merge Sort.
* Porównanie nastąpi na pięciu punktach pomiarowych wyznaczonych na podstawie analizy możliwej liczebności zbiorów wejściowych zapewniającej czasową mierzalność sortowania.
* Dla każdego z punktów pomiarowych zostanie przeprowadzonych 5 eksperymentów.
* Porównanie zachowania Insertion Sort oraz Merge Sort w zależności od zbioru wejściowego zostanie przeprowadzone poprzez dodawanie na końcu posortowanego wektora liczb losowych.

## Eksperyment - opis

### Środowisko

|  |  |
| --- | --- |
| Element środowiska | Parametry |
| Procesor | Inter® Core™ i7-7600 CPU @ 2.80 GHz 2.90 GHZ |
| RAM | 8 GB |
| System | Windows 10 Enterprise version 1809 |
| Dysk | SSD Samsung PM961 256 GB M.2 2280 PCI-E x4 Gen3 NVMe (MZVLW256HEHP-00000) |
| Język programowania | Python 3.8.2 32-bit |
| IDE | Microsoft Visual Studio Community 2019 Version 16.8.2 |

Tabela 1: Specyfikacja środowiska, na którym przeprowadzano eksperyment[[1]](#footnote-1)

### Opis algorytmów sortowania

Na potrzeby eksperymentu wykorzystano trzy metody Selection Sort, Insertion Sort oraz Merge Sort, które zostaną przedstawione w bardziej szczegółowy sposób w dalszej części opracowania:

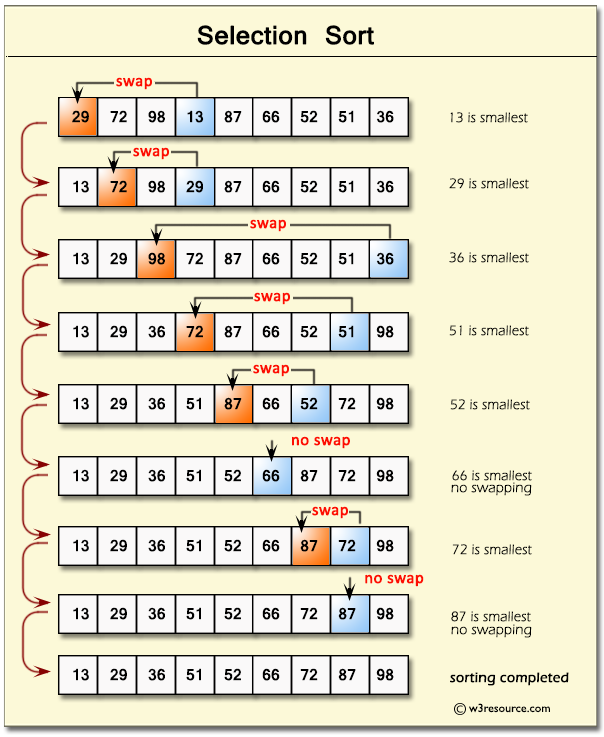
#### Selection sort

Selection sort[[2]](#footnote-2) w tłumaczeniu określane jest poprzez sortowanie przez wybór. Jest to jedna z prostszych metod sortowania, która cechuje się złożonością , gdzie określa liczbę elementów zbioru. Metoda sortowania przez wybieranie polega na wyszukaniu w zbiorze elementu, który powinien co do kolejności znajdować się na żądanej pozycji i zamienienie go z tym elementem, który obecnie ją zajmuje. Taką operację wykonuje się do momentu aż pozostały zbiór ma jeden element.

Przedstawiając tą metodę bardziej algorytmicznie:

1. Wyszukaj minimalną wartość zbioru spośród elementów od bieżącego do końca zbioru
2. Zamień wartość minimalną wyznaczoną w punkcie 1 z bieżącym elementem.

Metodę sortowania Selection Sort dla przejrzystości przedstawiamy graficznie:



Rysunek 1: Przykład sortowania z wykorzystaniem metody Selection sort[[3]](#footnote-3)

#### Insertion sort

Insertion sort[[4]](#footnote-4) w tłumaczeniu na język polski to metoda sortowania przez wstawianie. Jej nazwa odnosi się do sposobu układania kart podczas partii gry – tzn. układanie każdej karty w odpowiednie miejsce względem „wartości”. Sortowanie przez wstawianie cechuje się złożonością , gdzie określa liczbę elementów zbioru, czyli analogiczną do Selection sort. Insertion sort jest efektywny na małych zbiorach danych, zaś wraz ze wzrostem liczebności zbioru jego efektywność maleje. Warto zauważyć, iż jest to bardzo efektywna metoda w przypadku wykorzystania na zbiorze już posortowanym, gdyż wtedy wymagane jest tylko operacji porównania.

Przedstawiając tą metodę bardziej algorytmicznie:

1. Dzielimy zbiór na część posortowaną (na początku jest to pierwszy element zbioru) i nieposortowaną.
2. W każdym kolejnym kroku bierzemy pierwszy element z części nieposortowanej i wstawiamy we właściwe miejsce w części posortowanej (poprzez porównanie z ostatnim i poprzednimi elementami posortowanej części zbioru).

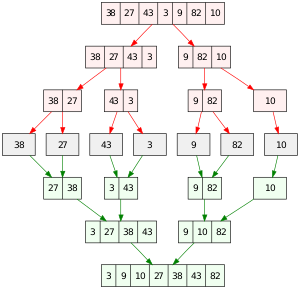
Metodę można również dla uproszczenia przedstawić graficznie:



Rysunek 2: Przykład sortowania z wykorzystaniem metody Insertion sort[[5]](#footnote-5).

#### Merge sort

Merge sort można przetłumaczyć, jako metodę sortowania przez złączanie, jest to w efektywniejszy algorytm sortowania od powyższych cechujący się złożonością czasową . Algorytm ten jest rekurencyjną metodą typu dziel i zwyciężaj, którego ideą jest podzielenie zbioru na mniejsze zbiory aż do uzyskania jednoelementowych zbiorów i następnie łączenie ich w posortowane liczniejsze zbiory.



Rysunek 3: Przykład sortowania z wykorzystaniem metody Merge sort[[6]](#footnote-6).

### Generowanie danych losowych

Jako dane do przeprowadzenia eksperymentu wykorzystano polecenie języka Python random.random(), które generuje liczby losowe z przedziału (0, 1) w formacie float.

### Wyznaczenie punktów pomiarowych

Celem eksperymentu jest przeprowadzenie doświadczenia na określonych puntach pomiarowych i odpowiednich zbiorach. Do wyznaczenia pięciu punktów pomiarowych wyznaczono eksperymentalnie długości wektorów, dla których każda z trzech metod wyszukiwania trwa minimum jedną sekundę i około 5 minut dla każdego z trzech rodzajów danych wejściowych. Jak przedstawiają się wyniki eksperymentalnego wyznaczenia liczebności wektorów przedstawia poniższa tabela:

|  |  |  |  |
| --- | --- | --- | --- |
| Rodzaj danych wejściowych | Metoda sortowania | Długość wektora, żeby sortowanie trwało ok 1 sekundy | Długość wektora, żeby sortowanie trwało ok 5 minut |
| Wektor danych losowych | Selection Sort | 1 500 elementów | 25 000 elementów |
| Insertion Sort | 1 600 elementów | 26 00 elementów |
| Merge Sort | 24 000 elementów | 4 500 000 elementów |
| Wektor danych posortowanych rosnąco | Selection Sort | 1 700 elementów | 26 000 elementów |
| Insertion Sort | 500 000 elementów | -[[7]](#footnote-7) |
| Merge Sort | 24 000 elementów | 4 700 000 elementów |
| Wektor danych posortowanych malejąco | Selection Sort | 1 600 elementów | 23 000 elementów |
| Insertion Sort | 1 300 elementów | 20 000 elementów |
| Merge Sort | 24 000 elementów | 4 700 000 elementów |

Tabela 2: Wyniki pomiarów długości wektorów wejściowych w zależności od metody sortowania[[8]](#footnote-8)

Biorąc pod uwagę powyższe wyniki czasowe sortowania wektorów, przyjęto do dalszego eksperymentu pięciu punktów pomiarowych, które zostały wyznaczone na wektorze o długości 30000 elementów, dzięki czemu będzie możliwe przeprowadzenie obserwacji na zauważalnie mierzalnych danych. W ten sposób wyznaczono przedziały przedstawione w poniższej tabeli:

|  |  |  |
| --- | --- | --- |
| Numer przedziału | Pierwszy element | Ostatni element |
| 1 | 0 | 6 000 |
| 2 | 6 001 | 12 000 |
| 3 | 12 001 | 18 000 |
| 4 | 18 001 | 24 000 |
| 5 | 24 001 | 30 000 |

Tabela 3: Punkty pomiarowe i ich przedziały[[9]](#footnote-9)

### Procedura eksperymentu

Eksperyment polega na wykonaniu pięciokrotnego sortowania każdego z pięciu powyżej określonych odcinków każdym z trzech opisanych algorytmów. Powyższa procedura zostanie uruchomiona na 3 rodzajach danych wejściowych – wektor liczb losowych, wektor liczb posortowanych rosnąco oraz wektor liczb posortowanych malejąco. Procedura została uruchomiona poprzez Microsoft Visual Studio Community 2019. Następnie na podstawie otrzymanych wyników policzono średni czas algorytmu  oraz odchylenie standardowe wykorzystując poniższe wzory[[10]](#footnote-10):

### Wyniki z eksperymentu

Pełne wyniki eksperymentu zostały dołączone do pracy pod postacią jednego z załączników. Wyniki czasowe konkretnych metod na poszególnych odcinach pomiarowych zostały przedstawione poniżej w formie graficznej:

Rysunek 1: Wyniki Selection sort na danych losowych dla każdej z prób na odcinkach pomiarowych[[11]](#footnote-11)

Rysunek 2: Wyniki Selection sort na danych posortowanych rosnąco dla każdej z prób na odcinkach pomiarowych [[12]](#footnote-12)

Rysunek 3: Wyniki Selection sort na danych posortowanych malejąco dla każdej z prób na odcinkach pomiarowych [[13]](#footnote-13)

Z powyższych wykresów można zauważyć, iż sortowanie za pomocą metody Selection sort na wszystkich wektorach i odcinkach trwa od ok 14 do 20 sekund (jest jedna obserwacja odstająca ok 23 sekundy, co jednakże może wskazywać na chwilowe obciążenie zasobów maszyny testowej) i zachowuje się w miarę stabilnie. Warte podkreślenia jest to, iż wyniki na wektorze posortowanym malejąco cechują się najwyższą wizualnie średnią, co jest zgodne z odczuciami, co do tej metody sortującej. Również widać, iż wyniki pomiarowe metody Selection sort na danych losowych cechują się największą zmiennością.

Poniżej alogiczne graficzne przedstawienie dla wyników metody Insertion sort (wartości osi pionowych pomiędzy wykresami nie są konsystentne):

Rysunek 4: Wyniki Insertion sort na danych losowych dla każdej z prób na odcinkach pomiarowych[[14]](#footnote-14)

Rysunek 5: Wyniki Insertion sort na danych posortowanych rosnąco dla każdej z prób na odcinkach pomiarowych[[15]](#footnote-15)

Rysunek 6: Wyniki Insertion sort na danych posortowanych malejąco dla każdej z prób na odcinkach pomiarowych[[16]](#footnote-16)

Rysunek 7: Wyniki Merge sort na danych losowych dla każdej z prób na odcinkach pomiarowych[[17]](#footnote-17)

Rysunek 8: Wyniki Merge sort na danych posortowanych rosnąco dla każdej z prób na odcinkach pomiarowych[[18]](#footnote-18)

Rysunek 9: Wyniki Merge sort na danych posortowanych malejąco dla każdej z prób na odcinkach pomiarowych[[19]](#footnote-19)

### Wyniki finalne i wnioski

Na podstawie pełnych wyników wyznaczono wartości średnie oraz odchylenia standardowe dla każdego z zastosowanych algorytmów. Wyniki przedstawiają się następująco:

|  |  |  |
| --- | --- | --- |
| Algorytm | Średnia [s] | Odchylenie standardowe [s] |
| Podwójne | 1.123006 | 0.670374 |
| Pojedyncze | 0.721173 | 0.110435 |
| Turniej | 15.57815 | 6.365965 |

Tabela 4: Średnia oraz odchylenie standardowe w zależności od zastosowanej metody[[20]](#footnote-20)

Jak pokazano w powyższej tablicy najszybszą metodą podczas analizy okazała się metoda pojedynczego przeszukiwania, która osiągnęła średni wynik poniżej 1 sekundy, drugą była metoda podwójnego przeszukiwania, której średni czas wyniósł ok 1.12 sekundy, zaś najwolniejsza była metoda turniejowa, która osiągała wyniki wręcz z innej skali uzyskując średnio 15.58 sekundy. Warte podkreślenia jest, iż wyniki uzyskane za pomocą pojedynczego przeszukiwania cechowały się najmniejszą zmiennością wyników (odchylenie standardowe wyniosło 0.11 sekundy), trochę większą zmienność uzyskała metoda podwójnego wyszukiwania (6-cio krotnie większe niż przy pojedynczej) oraz analogicznie jak przy wartości średniej metoda turniejowa osiągnęła wynik zmienności odchylenia standardowego nieporównywalny dla szybszych metod.

Zatem na podstawie eksperymentu należy stwierdzić, iż gdy celem przewodnim jest czas sugerowane jest wykorzystanie metody pojedynczej, natomiast, jeżeli z punktu widzenia przeprowadzającego działanie priorytet to zmienność czasu wyszukiwania – sugerowane jest wykorzystanie metody podwójnego przeszukiwania zbioru danych.

# Załączniki

## Pełne wyniki eksperymentu I

|  |  |  |  |
| --- | --- | --- | --- |
| Metoda sortowania | Wektor wejściowy | Odcinek pomiarowy | Wynik [s] |
| Selection Sort | losowy | 1 | 14.123602 |
| Selection Sort | losowy | 1 | 15.587969 |
| Selection Sort | losowy | 1 | 17.307768 |
| Selection Sort | losowy | 1 | 16.243497 |
| Selection Sort | losowy | 1 | 15.710739 |
| Insertion Sort | losowy | 1 | 14.122369 |
| Insertion Sort | losowy | 1 | 13.466210 |
| Insertion Sort | losowy | 1 | 13.010858 |
| Insertion Sort | losowy | 1 | 14.918575 |
| Insertion Sort | losowy | 1 | 12.890246 |
| Merge Sort | losowy | 1 | 0.244002 |
| Merge Sort | losowy | 1 | 0.278005 |
| Merge Sort | losowy | 1 | 0.246597 |
| Merge Sort | losowy | 1 | 0.288558 |
| Merge Sort | losowy | 1 | 0.242509 |
| Selection Sort | posortowany rosnąco | 1 | 15.363341 |
| Selection Sort | posortowany rosnąco | 1 | 14.491281 |
| Selection Sort | posortowany rosnąco | 1 | 14.009255 |
| Selection Sort | posortowany rosnąco | 1 | 14.772845 |
| Selection Sort | posortowany rosnąco | 1 | 15.014056 |
| Insertion Sort | posortowany rosnąco | 1 | 0.013998 |
| Insertion Sort | posortowany rosnąco | 1 | 0.025419 |
| Insertion Sort | posortowany rosnąco | 1 | 0.020001 |
| Insertion Sort | posortowany rosnąco | 1 | 0.026999 |
| Insertion Sort | posortowany rosnąco | 1 | 0.020999 |
| Merge Sort | posortowany rosnąco | 1 | 0.267580 |
| Merge Sort | posortowany rosnąco | 1 | 0.382005 |
| Merge Sort | posortowany rosnąco | 1 | 0.375543 |
| Merge Sort | posortowany rosnąco | 1 | 0.272003 |
| Merge Sort | posortowany rosnąco | 1 | 0.251007 |
| Selection Sort | posortowany malejąco | 1 | 18.439076 |
| Selection Sort | posortowany malejąco | 1 | 18.139207 |
| Selection Sort | posortowany malejąco | 1 | 18.112045 |
| Selection Sort | posortowany malejąco | 1 | 18.206097 |
| Selection Sort | posortowany malejąco | 1 | 17.560518 |
| Insertion Sort | posortowany malejąco | 1 | 25.318010 |
| Insertion Sort | posortowany malejąco | 1 | 26.474244 |
| Insertion Sort | posortowany malejąco | 1 | 25.236140 |
| Insertion Sort | posortowany malejąco | 1 | 27.599443 |
| Insertion Sort | posortowany malejąco | 1 | 27.117572 |
| Merge Sort | posortowany malejąco | 1 | 0.232035 |
| Merge Sort | posortowany malejąco | 1 | 0.249005 |
| Merge Sort | posortowany malejąco | 1 | 0.266069 |
| Merge Sort | posortowany malejąco | 1 | 0.275002 |
| Merge Sort | posortowany malejąco | 1 | 0.238003 |
| Selection Sort | losowy | 2 | 16.065088 |
| Selection Sort | losowy | 2 | 14.744241 |
| Selection Sort | losowy | 2 | 15.477605 |
| Selection Sort | losowy | 2 | 14.825859 |
| Selection Sort | losowy | 2 | 14.237963 |
| Insertion Sort | losowy | 2 | 12.962677 |
| Insertion Sort | losowy | 2 | 12.435288 |
| Insertion Sort | losowy | 2 | 14.208402 |
| Insertion Sort | losowy | 2 | 15.578129 |
| Insertion Sort | losowy | 2 | 13.728486 |
| Merge Sort | losowy | 2 | 0.281567 |
| Merge Sort | losowy | 2 | 0.384004 |
| Merge Sort | losowy | 2 | 0.249571 |
| Merge Sort | losowy | 2 | 0.340001 |
| Merge Sort | losowy | 2 | 0.262007 |
| Selection Sort | posortowany rosnąco | 2 | 15.036231 |
| Selection Sort | posortowany rosnąco | 2 | 15.623363 |
| Selection Sort | posortowany rosnąco | 2 | 14.989795 |
| Selection Sort | posortowany rosnąco | 2 | 14.994445 |
| Selection Sort | posortowany rosnąco | 2 | 14.773172 |
| Insertion Sort | posortowany rosnąco | 2 | 0.012000 |
| Insertion Sort | posortowany rosnąco | 2 | 0.014000 |
| Insertion Sort | posortowany rosnąco | 2 | 0.013039 |
| Insertion Sort | posortowany rosnąco | 2 | 0.011967 |
| Insertion Sort | posortowany rosnąco | 2 | 0.015998 |
| Merge Sort | posortowany rosnąco | 2 | 0.230054 |
| Merge Sort | posortowany rosnąco | 2 | 0.227968 |
| Merge Sort | posortowany rosnąco | 2 | 0.224180 |
| Merge Sort | posortowany rosnąco | 2 | 0.221005 |
| Merge Sort | posortowany rosnąco | 2 | 0.285129 |
| Selection Sort | posortowany malejąco | 2 | 18.738145 |
| Selection Sort | posortowany malejąco | 2 | 18.998890 |
| Selection Sort | posortowany malejąco | 2 | 19.331222 |
| Selection Sort | posortowany malejąco | 2 | 19.503474 |
| Selection Sort | posortowany malejąco | 2 | 19.437879 |
| Insertion Sort | posortowany malejąco | 2 | 27.585631 |
| Insertion Sort | posortowany malejąco | 2 | 26.757496 |
| Insertion Sort | posortowany malejąco | 2 | 27.768525 |
| Insertion Sort | posortowany malejąco | 2 | 27.236309 |
| Insertion Sort | posortowany malejąco | 2 | 27.625418 |
| Merge Sort | posortowany malejąco | 2 | 0.241971 |
| Merge Sort | posortowany malejąco | 2 | 0.232034 |
| Merge Sort | posortowany malejąco | 2 | 0.255968 |
| Merge Sort | posortowany malejąco | 2 | 0.252030 |
| Merge Sort | posortowany malejąco | 2 | 0.243967 |
| Selection Sort | losowy | 3 | 15.255867 |
| Selection Sort | losowy | 3 | 16.369053 |
| Selection Sort | losowy | 3 | 23.594894 |
| Selection Sort | losowy | 3 | 16.540653 |
| Selection Sort | losowy | 3 | 19.031138 |
| Insertion Sort | losowy | 3 | 14.577385 |
| Insertion Sort | losowy | 3 | 15.783875 |
| Insertion Sort | losowy | 3 | 14.767660 |
| Insertion Sort | losowy | 3 | 12.541896 |
| Insertion Sort | losowy | 3 | 13.213298 |
| Merge Sort | losowy | 3 | 0.252000 |
| Merge Sort | losowy | 3 | 0.283000 |
| Merge Sort | losowy | 3 | 0.249550 |
| Merge Sort | losowy | 3 | 0.270003 |
| Merge Sort | losowy | 3 | 0.251008 |
| Selection Sort | posortowany rosnąco | 3 | 16.261928 |
| Selection Sort | posortowany rosnąco | 3 | 14.761496 |
| Selection Sort | posortowany rosnąco | 3 | 14.088903 |
| Selection Sort | posortowany rosnąco | 3 | 14.921989 |
| Selection Sort | posortowany rosnąco | 3 | 14.281313 |
| Insertion Sort | posortowany rosnąco | 3 | 0.011001 |
| Insertion Sort | posortowany rosnąco | 3 | 0.012000 |
| Insertion Sort | posortowany rosnąco | 3 | 0.012001 |
| Insertion Sort | posortowany rosnąco | 3 | 0.011965 |
| Insertion Sort | posortowany rosnąco | 3 | 0.012999 |
| Merge Sort | posortowany rosnąco | 3 | 00.292030 |
| Merge Sort | posortowany rosnąco | 3 | 00.261999 |
| Merge Sort | posortowany rosnąco | 3 | 00.236135 |
| Merge Sort | posortowany rosnąco | 3 | 00.279997 |
| Merge Sort | posortowany rosnąco | 3 | 00.295003 |
| Selection Sort | posortowany malejąco | 3 | 18.688270 |
| Selection Sort | posortowany malejąco | 3 | 18.819427 |
| Selection Sort | posortowany malejąco | 3 | 18.470397 |
| Selection Sort | posortowany malejąco | 3 | 18.269580 |
| Selection Sort | posortowany malejąco | 3 | 17.400322 |
| Insertion Sort | posortowany malejąco | 3 | 25.251942 |
| Insertion Sort | posortowany malejąco | 3 | 26.166021 |
| Insertion Sort | posortowany malejąco | 3 | 24.872305 |
| Insertion Sort | posortowany malejąco | 3 | 25.479310 |
| Insertion Sort | posortowany malejąco | 3 | 25.624953 |
| Merge Sort | posortowany malejąco | 3 | 0.242513 |
| Merge Sort | posortowany malejąco | 3 | 0.234002 |
| Merge Sort | posortowany malejąco | 3 | 0.230002 |
| Merge Sort | posortowany malejąco | 3 | 0.240000 |
| Merge Sort | posortowany malejąco | 3 | 0.226997 |
| Selection Sort | losowy | 4 | 14.210971 |
| Selection Sort | losowy | 4 | 14.652501 |
| Selection Sort | losowy | 4 | 14.098933 |
| Selection Sort | losowy | 4 | 14.951873 |
| Selection Sort | losowy | 4 | 15.771264 |
| Insertion Sort | losowy | 4 | 13.579989 |
| Insertion Sort | losowy | 4 | 13.169060 |
| Insertion Sort | losowy | 4 | 13.889991 |
| Insertion Sort | losowy | 4 | 12.823024 |
| Insertion Sort | losowy | 4 | 13.300231 |
| Merge Sort | losowy | 4 | 0.256032 |
| Merge Sort | losowy | 4 | 0.264969 |
| Merge Sort | losowy | 4 | 0.247967 |
| Merge Sort | losowy | 4 | 0.251008 |
| Merge Sort | losowy | 4 | 0.253964 |
| Selection Sort | posortowany rosnąco | 4 | 15.252064 |
| Selection Sort | posortowany rosnąco | 4 | 14.218985 |
| Selection Sort | posortowany rosnąco | 4 | 15.634654 |
| Selection Sort | posortowany rosnąco | 4 | 14.470994 |
| Selection Sort | posortowany rosnąco | 4 | 16.312050 |
| Insertion Sort | posortowany rosnąco | 4 | 0.013995 |
| Insertion Sort | posortowany rosnąco | 4 | 0.012031 |
| Insertion Sort | posortowany rosnąco | 4 | 0.011962 |
| Insertion Sort | posortowany rosnąco | 4 | 0.013003 |
| Insertion Sort | posortowany rosnąco | 4 | 0.014001 |
| Merge Sort | posortowany rosnąco | 4 | 0.306997 |
| Merge Sort | posortowany rosnąco | 4 | 0.281998 |
| Merge Sort | posortowany rosnąco | 4 | 0.247999 |
| Merge Sort | posortowany rosnąco | 4 | 0.255002 |
| Merge Sort | posortowany rosnąco | 4 | 0.260999 |
| Selection Sort | posortowany malejąco | 4 | 18.166628 |
| Selection Sort | posortowany malejąco | 4 | 18.797067 |
| Selection Sort | posortowany malejąco | 4 | 19.791148 |
| Selection Sort | posortowany malejąco | 4 | 19.244034 |
| Selection Sort | posortowany malejąco | 4 | 19.245033 |
| Insertion Sort | posortowany malejąco | 4 | 25.717183 |
| Insertion Sort | posortowany malejąco | 4 | 26.056865 |
| Insertion Sort | posortowany malejąco | 4 | 26.069569 |
| Insertion Sort | posortowany malejąco | 4 | 32.666281 |
| Insertion Sort | posortowany malejąco | 4 | 25.796408 |
| Merge Sort | posortowany malejąco | 4 | 0.236035 |
| Merge Sort | posortowany malejąco | 4 | 0.234964 |
| Merge Sort | posortowany malejąco | 4 | 0.271003 |
| Merge Sort | posortowany malejąco | 4 | 0.234995 |
| Merge Sort | posortowany malejąco | 4 | 0.244003 |
| Selection Sort | losowy | 5 | 14.829015 |
| Selection Sort | losowy | 5 | 15.468017 |
| Selection Sort | losowy | 5 | 14.347019 |
| Selection Sort | losowy | 5 | 15.353044 |
| Selection Sort | losowy | 5 | 14.790039 |
| Insertion Sort | losowy | 5 | 12.912029 |
| Insertion Sort | losowy | 5 | 13.391788 |
| Insertion Sort | losowy | 5 | 12.543915 |
| Insertion Sort | losowy | 5 | 14.325279 |
| Insertion Sort | losowy | 5 | 12.570121 |
| Merge Sort | losowy | 5 | 0.239999 |
| Merge Sort | losowy | 5 | 0.256966 |
| Merge Sort | losowy | 5 | 0.244038 |
| Merge Sort | losowy | 5 | 0.246481 |
| Merge Sort | losowy | 5 | 0.239035 |
| Selection Sort | posortowany rosnąco | 5 | 14.440980 |
| Selection Sort | posortowany rosnąco | 5 | 15.627907 |
| Selection Sort | posortowany rosnąco | 5 | 16.081218 |
| Selection Sort | posortowany rosnąco | 5 | 14.321544 |
| Selection Sort | posortowany rosnąco | 5 | 14.324187 |
| Insertion Sort | posortowany rosnąco | 5 | 0.017001 |
| Insertion Sort | posortowany rosnąco | 5 | 0.013278 |
| Insertion Sort | posortowany rosnąco | 5 | 0.012000 |
| Insertion Sort | posortowany rosnąco | 5 | 0.016996 |
| Insertion Sort | posortowany rosnąco | 5 | 0.013001 |
| Merge Sort | posortowany rosnąco | 5 | 0.425995 |
| Merge Sort | posortowany rosnąco | 5 | 0.332518 |
| Merge Sort | posortowany rosnąco | 5 | 0.309568 |
| Merge Sort | posortowany rosnąco | 5 | 0.336048 |
| Merge Sort | posortowany rosnąco | 5 | 0.393532 |
| Selection Sort | posortowany malejąco | 5 | 18.925271 |
| Selection Sort | posortowany malejąco | 5 | 19.004820 |
| Selection Sort | posortowany malejąco | 5 | 17.507545 |
| Selection Sort | posortowany malejąco | 5 | 18.143203 |
| Selection Sort | posortowany malejąco | 5 | 17.812326 |
| Insertion Sort | posortowany malejąco | 5 | 34.862508 |
| Insertion Sort | posortowany malejąco | 5 | 28.343228 |
| Insertion Sort | posortowany malejąco | 5 | 30.137019 |
| Insertion Sort | posortowany malejąco | 5 | 33.502852 |
| Insertion Sort | posortowany malejąco | 5 | 30.514033 |
| Merge Sort | posortowany malejąco | 5 | 0.266965 |
| Merge Sort | posortowany malejąco | 5 | 0.243002 |
| Merge Sort | posortowany malejąco | 5 | 0.316044 |
| Merge Sort | posortowany malejąco | 5 | 0.314569 |
| Merge Sort | posortowany malejąco | 5 | 0.226031 |

## Kod programu

Poniżej załączono kod programu (dostępny również na platformie github: https://github.com/oksiedz/Python/tree/master/Projects/Project1), którym przeprowadzono eksperyment. Wyznaczanie krańców przedziału odbywało się poprzez zmianę wartości zmiennej NoOfNumbers znajdującego się w piątej linii kodu.

|  |
| --- |
| import random  import datetime  import math  NoOfNumbers = 7000000  randomlist = []  print("Rozpoczecie generowania ciagu liczb.")  for i in range(0, NoOfNumbers):  randomlist.append(random.random())  print("Zakonczenie generowania ciagu liczb.")  listofresults = []  #wyszukanie najwyższych wartości podwójnym przeszukaniem  def double\_search(startnumber, endnumber):  #print("Podwojne przeszukanie wszystkich elementow")  max\_1 = 0  max\_2 = 0  listofnumbers = randomlist[startnumber:endnumber]  starttime = datetime.datetime.now()  for i in range(0, len(listofnumbers)):  if listofnumbers[i] > max\_1:  max\_1 = listofnumbers[i]  for i in range(0, len(listofnumbers)):  if (listofnumbers[i] > max\_2 and listofnumbers[i] < max\_1):  max\_2 = listofnumbers[i]  endtime = datetime.datetime.now()  #print("Maksymalna liczba jest rowna: ", max\_1, ", druga co do wielkosci jest rowna: ", max\_2, ", czas wyszukiwania wynosi: ", endtime - starttime)  listofresults.append("Podwójne"+str(max\_1)+";"+str(max\_2)+";"+str(endtime-starttime))  #wyszukanie najwyższych wartości pojedynczym przeszukaniem  def single\_search(startnumber, endnumber):  #print("Pojedyncze przeszukanie wszystkich elementow")  max\_1 = 0  max\_2 = 0  listofnumbers = randomlist[startnumber:endnumber]  starttime = datetime.datetime.now()  for i in range(0, len(listofnumbers)):  if listofnumbers[i] > max\_2 and listofnumbers[i] < max\_1:  max\_2 = listofnumbers[i]  if listofnumbers[i] > max\_2 and listofnumbers[i] > max\_1:  max\_2 = max\_1  max\_1 = listofnumbers[i]  endtime = datetime.datetime.now()  #print("Maksymalna liczba jest rowna: ", max\_1, ", druga co do wielkosci jest rowna: ", max\_2, ", czas wyszukiwania wynosi: ", endtime - starttime)  listofresults.append("Pojedyncze"+str(max\_1)+";"+str(max\_2)+";"+str(endtime-starttime))  #wyszukanie najwyższych wartości metodą turniejową  def tournament\_search(startnumber, endnumber):  #print("Przeszukanie wszystkich elementow metodą turniejową")  #start clearing and defining parameters  listofloserpairs = []  listofnumbers = randomlist[startnumber:endnumber]  koniec\_szukania = 0  listofwinners = []  change\_the\_listofnumbers = 0  round = 1  #end clearing and defining parameters  starttime = datetime.datetime.now()  #start of loop  while (koniec\_szukania == 0):  if change\_the\_listofnumbers == 1:  listofnumbers = listofwinners  listofwinners = []  n = int(math.ceil(len(listofnumbers) / 2))    if n == 1:  koniec\_szukania = 1  if float(listofnumbers[0]) > float(listofnumbers[1]):  if round == 1:  max\_1 = listofnumbers[0]  if round == 2:  max\_2 = listofnumbers[0]  listofloserpairs.append(str(listofnumbers[0])+";"+str(listofnumbers[1]))  else:  if round == 1:  max\_1 = listofnumbers[1]  if round == 2:  max\_2 = listofnumbers[1]  listofloserpairs.append(str(listofnumbers[1])+";"+str(listofnumbers[0]))  else:  for i in range(0, n):  if 2\*i+1<len(listofnumbers):  if float(listofnumbers[2\*i]) > float(listofnumbers[2\*i+1]):  listofwinners.append(listofnumbers[2\*i])  listofloserpairs.append(str(listofnumbers[2\*i])+";"+str(listofnumbers[2\*i+1]))  else:  listofwinners.append(listofnumbers[2\*i+1])  listofloserpairs.append(str(listofnumbers[2\*i+1])+";"+str(listofnumbers[2\*i]))  else:  listofwinners.append(str(listofnumbers[2\*i]))  change\_the\_listofnumbers = 1  #end of loop  #start clear the parameter to provide new players for the tournament  listofnumbers = []  #end clear the parameter to provide new players for the tournament  #start creation of new players  for i in range(0, len(listofloserpairs)):  pairforcomparison = listofloserpairs[i].split(";")  if float(pairforcomparison[0]) == float(max\_1):  listofnumbers.append(pairforcomparison[1])  #end creation of new players  ####start of second tournament for the losers with the winner  listofloserpairs = []  koniec\_szukania = 0  listofwinners = []  change\_the\_listofnumbers = 0  round = 2  #start of loop  while (koniec\_szukania == 0):  if change\_the\_listofnumbers == 1:  listofnumbers = listofwinners  listofwinners = []  n = int(math.ceil(len(listofnumbers) / 2))    if n == 1:  koniec\_szukania = 1  if float(listofnumbers[0]) > float(listofnumbers[1]):  if round == 1:  max\_1 = listofnumbers[0]  if round == 2:  max\_2 = listofnumbers[0]  # listofloserpairs.append(str(listofnumbers[0])+";"+str(listofnumbers[1]))  else:  if round == 1:  max\_1 = listofnumbers[1]  if round == 2:  max\_2 = listofnumbers[1]  # listofloserpairs.append(str(listofnumbers[1])+";"+str(listofnumbers[0]))  else:  for i in range(0, n):  if 2\*i+1<len(listofnumbers):  if listofnumbers[2\*i] > listofnumbers[2\*i+1]:  listofwinners.append(listofnumbers[2\*i])  # listofloserpairs.append(str(listofnumbers[2\*i])+";"+str(listofnumbers[2\*i+1]))  else:  listofwinners.append(listofnumbers[2\*i+1])  # listofloserpairs.append(str(listofnumbers[2\*i+1])+";"+str(listofnumbers[2\*i]))  else:  listofwinners.append(str(listofnumbers[2\*i]))  change\_the\_listofnumbers = 1  #end of loop  endtime = datetime.datetime.now()  #print("Maksymalna liczba jest rowna: ", max\_1, ", druga co do wielkosci jest rowna: ", max\_2, ", czas wyszukiwania wynosi: ", endtime - starttime)  listofresults.append("Turniej"+str(max\_1)+";"+str(max\_2)+";"+str(endtime-starttime))  print("1. Double search")  for i in range(0,10):  double\_search(1000000, 2200000)  print("2. Double search")  for i in range(0,10):  double\_search(2200000, 3400000)  print("3. Double search")  for i in range(0,10):  double\_search(3400000, 4600000)  print("4. Double search")  for i in range(0,10):  double\_search(4600000, 5800000)  print("5. Double search")  for i in range(0,10):  double\_search(5800000, 7000000)  print("1. single search")  for i in range(0,10):  single\_search(1000000, 2200000)  print("2. single search")  for i in range(0,10):  single\_search(2200000, 3400000)  print("3. single search")  for i in range(0,10):  single\_search(3400000, 4600000)  print("4. single search")  for i in range(0,10):  single\_search(4600000, 5800000)  print("5. single search")  for i in range(0,10):  single\_search(5800000, 7000000)  print("1. tournament search")  for i in range(0,10):  tournament\_search(1000000, 2200000)  print("2. tournament search")  for i in range(0,10):  tournament\_search(2200000, 3400000)  print("3. tournament search")  for i in range(0,10):  tournament\_search(3400000, 4600000)  print("4. tournament search")  for i in range(0,10):  tournament\_search(4600000, 5800000)  print("5. tournament search")  for i in range(0,10):  tournament\_search(5800000, 7000000)  for i in range(0, len(listofresults)):  print(listofresults[i]) |

1. Opracowanie własne [↑](#footnote-ref-1)
2. https://www.tutorialspoint.com/data\_structures\_algorithms/selection\_sort\_algorithm.htm [↑](#footnote-ref-2)
3. https://www.w3resource.com/php-exercises/searching-and-sorting-algorithm/searching-and-sorting-algorithm-exercise-4.php [↑](#footnote-ref-3)
4. http://www.algorytm.edu.pl/algorytmy-maturalne/sortowanie-przez-wstawianie.html [↑](#footnote-ref-4)
5. https://www.geeksforgeeks.org/insertion-sort/ [↑](#footnote-ref-5)
6. https://en.wikipedia.org/wiki/Merge\_sort [↑](#footnote-ref-6)
7. Przy 4 500 000 elementach czas sortowania Insertion Sort zajmował ok 11 sekund – nie weryfikowano dalej. [↑](#footnote-ref-7)
8. Opracowanie własne [↑](#footnote-ref-8)
9. Opracowanie własne [↑](#footnote-ref-9)
10. Średnia arytmetyczna: <https://www.matemaks.pl/srednia-arytmetyczna.html>, odchylenie standardowe: https://www.matemaks.pl/odchylenie-standardowe.html [↑](#footnote-ref-10)
11. Opracowanie własne [↑](#footnote-ref-11)
12. Opracowanie własne [↑](#footnote-ref-12)
13. Opracowanie własne [↑](#footnote-ref-13)
14. Opracowanie własne [↑](#footnote-ref-14)
15. Opracowanie własne [↑](#footnote-ref-15)
16. Opracowanie własne [↑](#footnote-ref-16)
17. Opracowanie własne [↑](#footnote-ref-17)
18. Opracowanie własne [↑](#footnote-ref-18)
19. Opracowanie własne [↑](#footnote-ref-19)
20. Opracowanie własne [↑](#footnote-ref-20)