Projekt Algorytmy I złożoność obliczeniowa

Temat projektu: Badanie efektywności wybranych algorytmów sortowania ze względu na złożoność obliczeniową.

Prowadzący	Dr. Inż. Zbigniew Jerzy Buchalski
Imię, Nazwisko, Nr. indeksu	Kornel Uriasz 272967
Termin zajęć: dzień tygodnia (tydzień), godzina	Wtorek (Tydzien nieparzysty) 11:30 – 13:00
Data oddania sprawozdania:	30.04.2024 r.

Ocena końcowa

Zatwierdzam wyniki pomiarów.....

Data i podpis prowadzącego zajęcia.....…

Adnotacje dotyczące wymaganych poprawek oraz daty otrzymania poprawionego sprawozdania :

Spis treści

1. Wprowadzenie

Celem projektu jest wybranie i zaimplementowanie określonych algorytmów sortujących, a następnie przeprowadzenie analizy ich efektywności. Elementy tablicy powinny być liczby całkowite ze znakiem 4 bajtowe oraz w przypadku jednego z algorytmów liczbę z zmiennym przecinkiem 4 bajtową. Jako tablice posortowaną uważamy tablice elementów, w których element poprzedzający jest mniejszy od następującego po nim, inaczej uporządkowana rosnąco. W programie zostały zaimplementowane 4 algorytmy sortowania oraz funkcje je obsługującego. Algorytmami sortowania są:

1. Algorytm sortowania przez wstawianie (z ang. Insertion Sort) - prosty algorytm sortowania, który działa poprzez iteracyjne budowanie posortowanej sekwencji. Algorytm ten przegląda elementy wejściowej listy po kolei i wstawia każdy z nich na odpowiednie miejsce w już posortowanej części listy. Jest to jeden z najprostszych algorytmów sortowania, ale może być nieefektywny dla dużych zbiorów danych. Cechy sortowania przez wstawianie:

- Prostota implementacji: Algorytm ten jest łatwy do zrozumienia i zaimplementowania.
- Stabilność: Sortowanie przez wstawianie jest stabilne, co oznacza, że zachowuje kolejność równych elementów.
- Efektywność dla małych zbiorów danych: W przypadku małych zbiorów danych sortowanie przez wstawianie może być szybsze niż niektóre bardziej złożone algorytmy sortowania, takie jak quicksort czy mergesort.
- Niska wydajność dla dużych zbiorów danych: Dla dużych zbiorów danych sortowanie przez wstawianie może być nieefektywne, ponieważ jego złożoność czasowa wynosi O(n^2), gdzie n to liczba elementów w liście.
- Złożoność czasowa: O(n^2) w najgorszym i średnim przypadku, O(n) w najlepszym przypadku (gdy lista jest już posortowana).
- Złożoność pamięciowa: O(1), algorytm sortowania przez wstawianie nie wymaga dodatkowej pamięci poza samą listą wejściową.

2. Sortowanie przez kopcowanie (ang. heapsort) to algorytm sortowania, który wykorzystuje strukturę danych zwaną kopcem (ang. heap). Kopiec to specjalna struktura danych, która spełnia warunki kopca, czyli jest to drzewo binarne, w którym każdy węzeł ma wartość większą lub równą (w kopcu max-heap) lub mniejszą lub równą (w kopcu min-heap) od wartości swoich dzieci. Cechy sortowania przez kopcowanie:

- Stabilność: Sortowanie przez kopcowanie nie jest stabilne, co oznacza, że nie gwarantuje zachowania kolejności równych elementów.
- Brak konieczności dodatkowej pamięci: Algorytm sortowania przez kopcowanie działa na miejscu, co oznacza, że nie wymaga dodatkowej pamięci poza samą listą wejściową.
- Wydajność: Sortowanie przez kopcowanie ma złożoność czasową O(n log n) dla wszystkich przypadków (najlepszy, średni i najgorszy), co czyni go efektywnym dla dużych zbiorów danych.
- Niemożność wersji adaptacyjnej: Nie ma adaptacyjnej wersji sortowania przez kopcowanie, co oznacza, że jego wydajność nie zależy od stopnia uporządkowania danych wejściowych.

- Odporność na pesymistyczne przypadki: Sortowanie przez kopcowanie jest odporną na pesymistyczne przypadki metodą sortowania, co oznacza, że jego wydajność nie drastycznie się pogarsza dla niekorzystnie uporządkowanych danych wejściowych.

3. Sortowanie szybkie (ang. quicksort) to popularny algorytm sortowania, który wykorzystuje strategię dziel i zwyciężaj. Algorytm ten opiera się na podziale listy na mniejsze podlisty, a następnie sortowaniu każdej z tych podlist. Sortowanie szybkie jest rekurencyjne, co oznacza, że sortuje mniejsze podlisty za pomocą siebie samego. Cechy sortowania szybkiego:

- Efektywność: Sortowanie szybkie jest jednym z najszybszych algorytmów sortowania dla typowych przypadków sortowania. Jego złożoność czasowa wynosi O(n log n) w przeciętnym i najlepszym przypadku, co czyni go bardzo efektywnym dla dużych zbiorów danych.
- Niepewność w najgorszym przypadku: Jedną z wad sortowania szybkiego jest to, że w najgorszym przypadku może mieć złożoność czasową O(n^2), co występuje, gdy lista jest już posortowana lub prawie posortowana, a pivot jest wybierany na skraju listy.
- Brak stabilności: Sortowanie szybkie nie jest stabilne, co oznacza, że nie zachowuje kolejności równych elementów.
- Brak dodatkowej pamięci: Algorytm sortowania szybkiego działa na miejscu, co oznacza, że nie wymaga dodatkowej pamięci poza samą listą wejściową.
- Możliwość optymalizacji: Istnieje wiele sposobów optymalizacji sortowania szybkiego, takich jak wybór optymalnego pivotu, wykorzystanie trzech median jako pivotu, czy też wykorzystanie sortowania przez wstawianie dla małych podlist. Te techniki mogą poprawić wydajność algorytmu w niektórych przypadkach.

4. Sortowanie Shella, znane również jako sortowanie wstawianie ze zmniejszającymi się odstępami, to algorytm sortowania, który jest ulepszoną wersją sortowania przez wstawianie. Algorytm ten został zaproponowany przez D.L. Shella w 1959 roku i polega na porównywaniu i przesuwaniu elementów o określonej odległości od siebie, a nie tylko sąsiadujących elementów. Cechy sortowania Shella:

- Efektywność: Sortowanie Shella łączy zalety sortowania przez wstawianie z wykorzystaniem odstępów, co może znacznie przyspieszyć proces sortowania, szczególnie dla dużych zbiorów danych.
- Brak stabilności: Podobnie jak sortowanie przez wstawianie, sortowanie Shella nie jest stabilne, co oznacza, że nie gwarantuje zachowania kolejności równych elementów.
- Złożoność czasowa: Złożoność czasowa sortowania Shella zależy od wybranej sekwencji odstępów. Dla niektórych sekwencji odstępów znaleziono algorytm o złożoności czasowej O(n^(4/3)), podczas gdy dla innych jest to O(n log n).
- Brak jednoznacznie najlepszej sekwencji odstępów: Nie ma jednoznacznie najlepszej sekwencji odstępów dla sortowania Shella. Wybór sekwencji odstępów może wpłynąć na wydajność algorytmu w zależności od danych wejściowych.
- Wrażliwość na wybór sekwencji odstępów: Efektywność sortowania Shella może być wrażliwa na wybór sekwencji odstępów. Niektóre sekwencje mogą działać lepiej dla niektórych zbiorów danych niż inne.

- Złożoność pamięciowa: Sortowanie Shella działa na miejscu, co oznacza, że nie wymaga dodatkowej pamięci poza samą listą wejściową.

W programie zostały

zaimplementowane dwie metody wyliczania odstępu. Jedna przedstawiona przez Pana Shella wynosząca N/2^k z zaokrągleniem w dół, której złożoność obliczeniowa w przypadku średnim wynosi O(n ^(3/2)), natomiast pesymistyczna to O(n^2). Druga przedstawiona przez Pana Curie wynosząca 2k −1 , której złożoność obliczeniowa w średnim oraz najgorszym przypadku O(n^(3/2)).

2. Plan eskperymentu

Testowanie algorytmów sortowania opierało się na sortowaniu danych generowanych dla 10 różnych rozmiarów tablicy. Dla każdego przypadku I rozmiaru w danej metodzie było wykonywanych 5 testów w celu minimalizacji błędów losowych. Dla danych przyjęto wielkości zbiorów : 10 000, 20 000, 30 000, 40 000, 50 000, 60 000, 70 000, 80 000, 90 000, 100 000. Przy każdej repetycji dane były losowane na nowo w celu uniknięcie wystąpienia przypadków skrajnych oraz uzyskania możliwie niezależnego średniego wyniku. Dane były losowane w oparciu o funkcję rand() z podstawowej biblioteki języka C. Dotyczy to liczb całkowitych z znakiem oraz liczb o zmiennym przecinku.

Pomiar czasu opierał się na odczytaniu zegara systemowego przed oraz po wywołaniu metody odpowiedzialnej za sortowanie zbioru danych. . Wyniki pomiaru w czasie działania programu są wyświetlane w oknie programu, plik tekstowy zostal wygenerowany poprzez przekierowanie standardowanego wyjścia na plik .txt . Wykresy zostały wygenerowane poprzez użycie skryptu w języku python.

Każdy z algorytmów miał przejść testy umieszczone w powyżej dla każdej możliwej konfiguracji. W szczególności dotyczy algorytmów sortowania Shella oraz szybkiego, które posiadały dodatkowe zmienne, które wszystkie zostały również przebadane. W przypadku algorytmu sortowania przez wstawianie, zostało zastosowane sortowanie na dwóch typach danych, 32-bitowych liczbach stało oraz zmienno-przecinkowych. W przypadku sortowania Shella konieczne bylo przetestowanie dwóch konfiguracji odstępów, natomiast w przypadku sortowania szybkiego testowane były konfiguracje wybieranie pivota. Konfiguracje wybierania pivota to : skrajnie lewy, skrajnie prawy, środkowy, oraz losowy.

3. Przebieg eksperymentu

Eksperyment przebiegły pomyślnie wykazując różnice pomiędzy wydajnością poszczególnych algorytmów oraz przyrostem czasu sortowania dla coraz większych biorów danych. Każdy z algorytmów przeszedł testy dla każdej wielkości tablicy w każdej możliwej konfiguracji. Wyniki pomiarów oraz wykresy zostały umieszczone poniżej.

Wielkość zbioru	Typ algorytmu sortującego [ms]	
	Sortowanie przez wstawianie uint32_t	Sortowanie przez wstawianie float
10 000	131	135
20 000	550	570
30 000	1150	1200
40 000	2160	2200
50 000	3305	3350
60 000	4850	4700
70 000	6667	6362
80 000	8550	8280
90 000	10500	11000
100 000	12750	14000

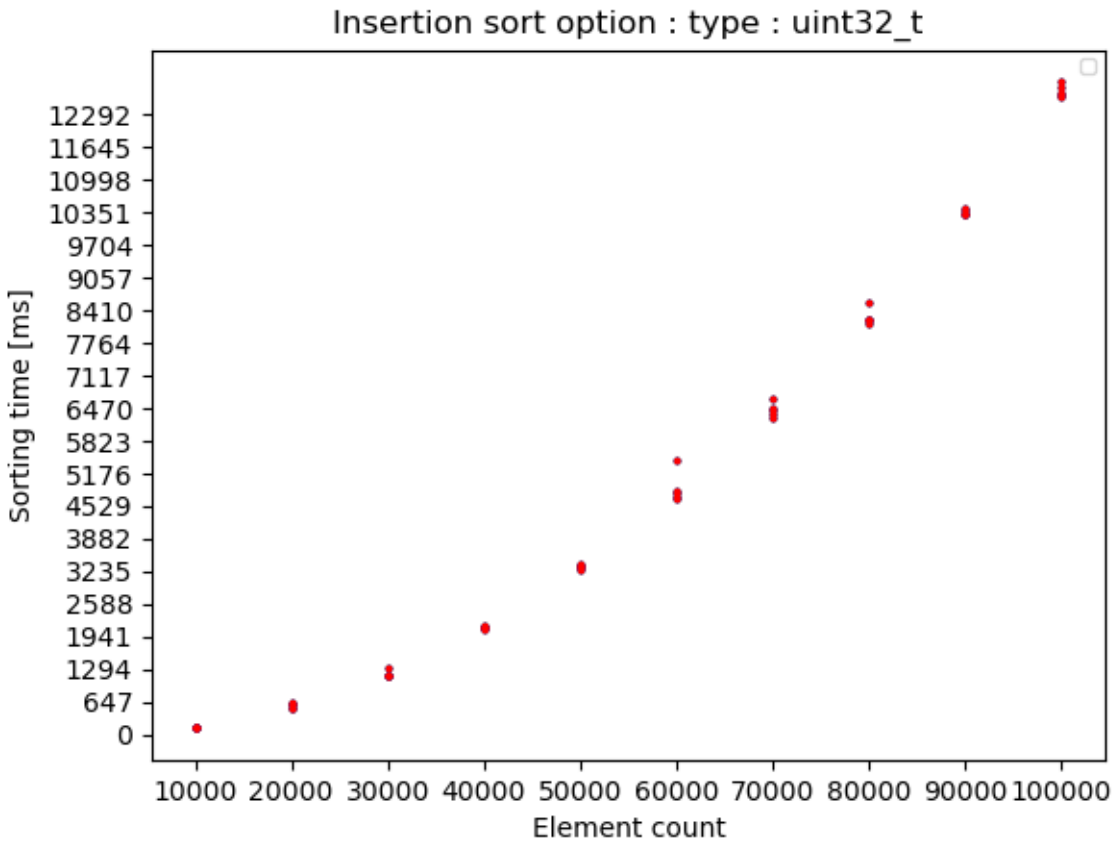
Wielkość zbioru	Typ algorytmu sortującego [ms]
	Sortowanie przez kopcowanie
10 000	3.50
20 000	7.5
30 000	11.4
40 000	17.0
50 000	21.4
60 000	25.0
70 000	30.4
80 000	34.7
90 000	39.5
100 000	44.2

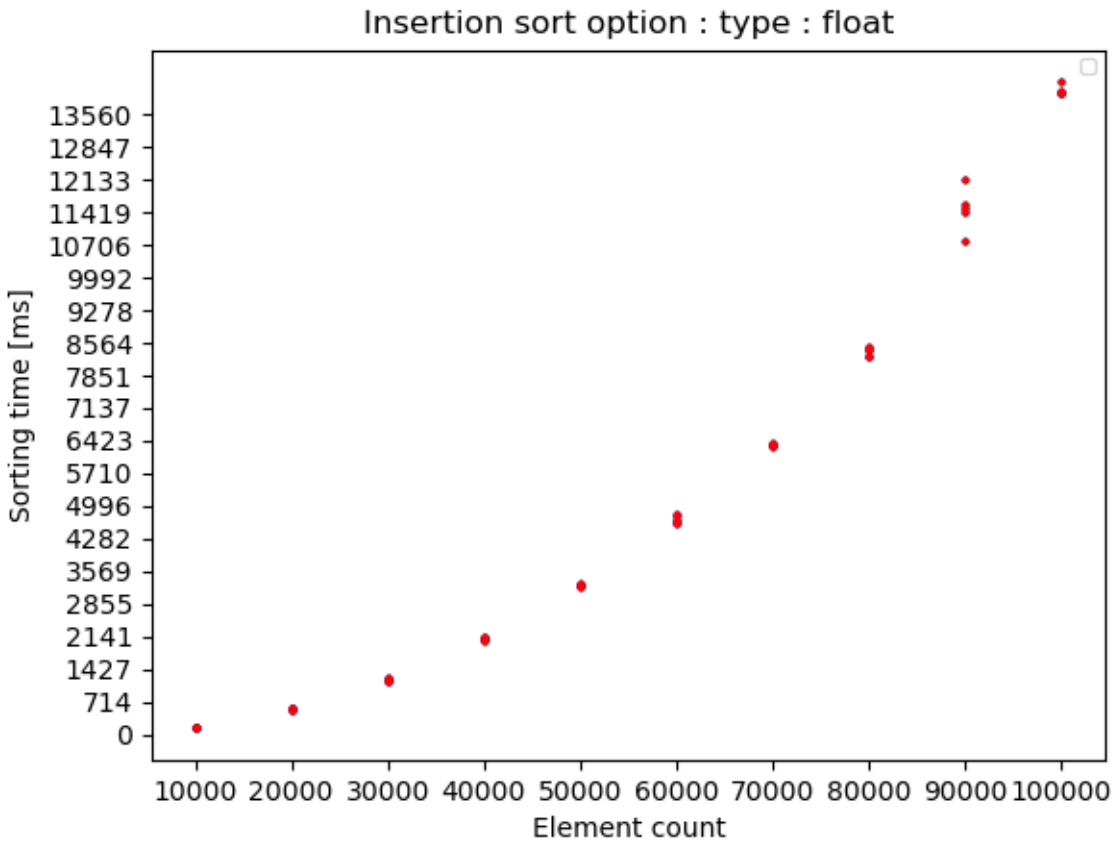
Wielkość zbioru	Typ algorytmu sortującego [ms]	
	Sortowanie shella, potęga 2	Sortowanie shella, odstępy Curie
10 000	3.3	3.2
20 000	7.2	7.0
30 000	10.7	13.2
40 000	16.2	18.5
50 000	22.3	25.8
60 000	25.3	34.2
70 000	30.2	52.7
80 000	38.2	55.1
90 000	43.7	65.0
100 000	50.1	80.4

Wielkość zbioru	Typ algorytmu sortującego [ms]			
	Sortowanie szybkie, pivot lewy	Sortowanie szybkie, pivot środkowy	Sortowanie szybkie, pivot prawy	Sortowanie szybkie, pivot losowy
10 000	1.65	1.35	1.59	1.79
20 000	3.43	3.07	3.60	3.58
30 000	5.67	4.63	5.42	5.54
40 000	7.20	6.74	7.67	7.93
50 000	9.82	8.54	9.29	9.21
60 000	11.33	10.53	11.70	11.5
70 000	13.82	12.32	14.7	13.32
80 000	15.52	14.90	15.83	15.43
90 000	18.67	15.78	18.20	17.45
100 000	20.65	17.42	20.92	19.45

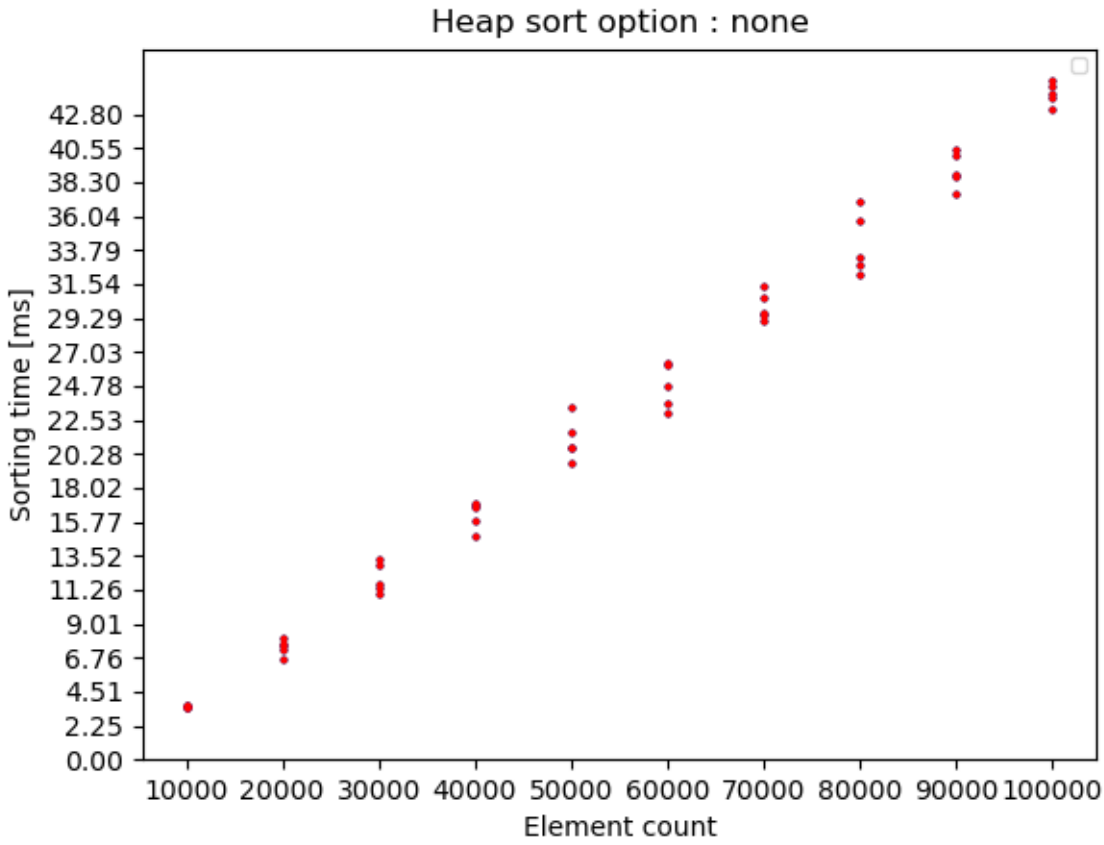
b. Wykresy reprezentujące wyniki pomiarów

Wykresy dla sortowania przez wstawianie:

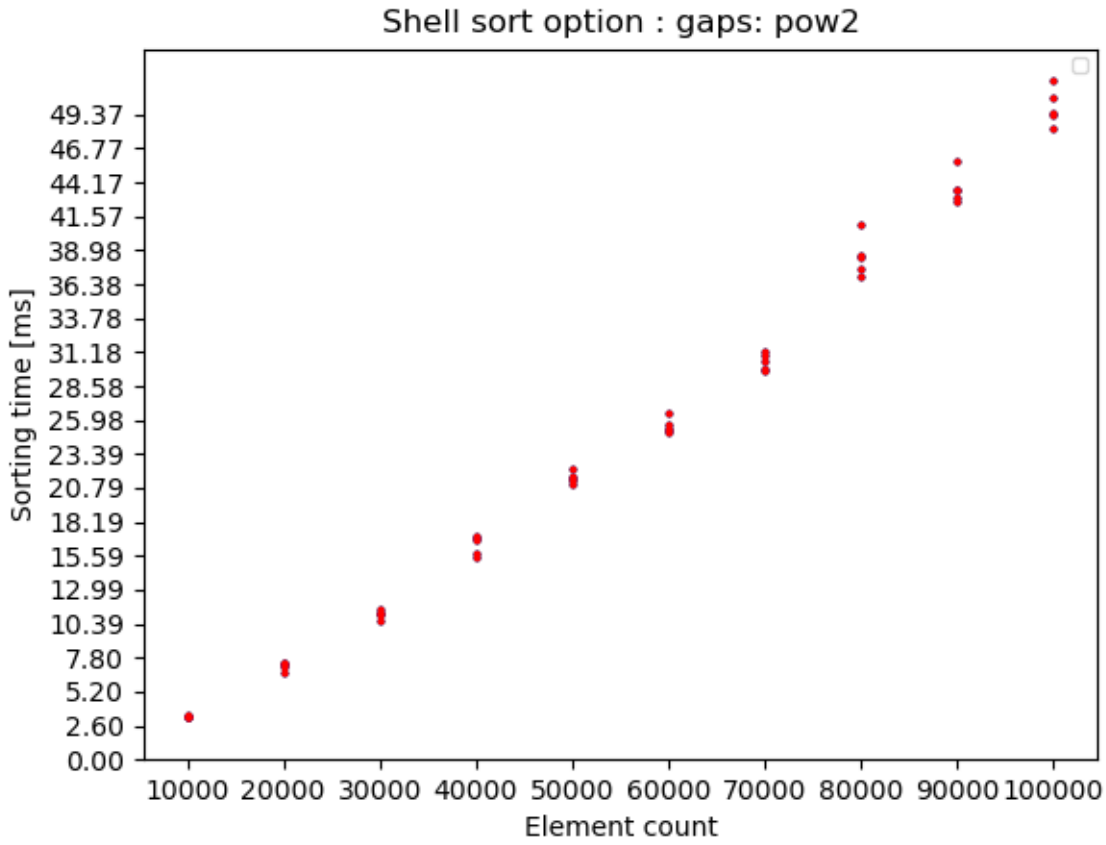


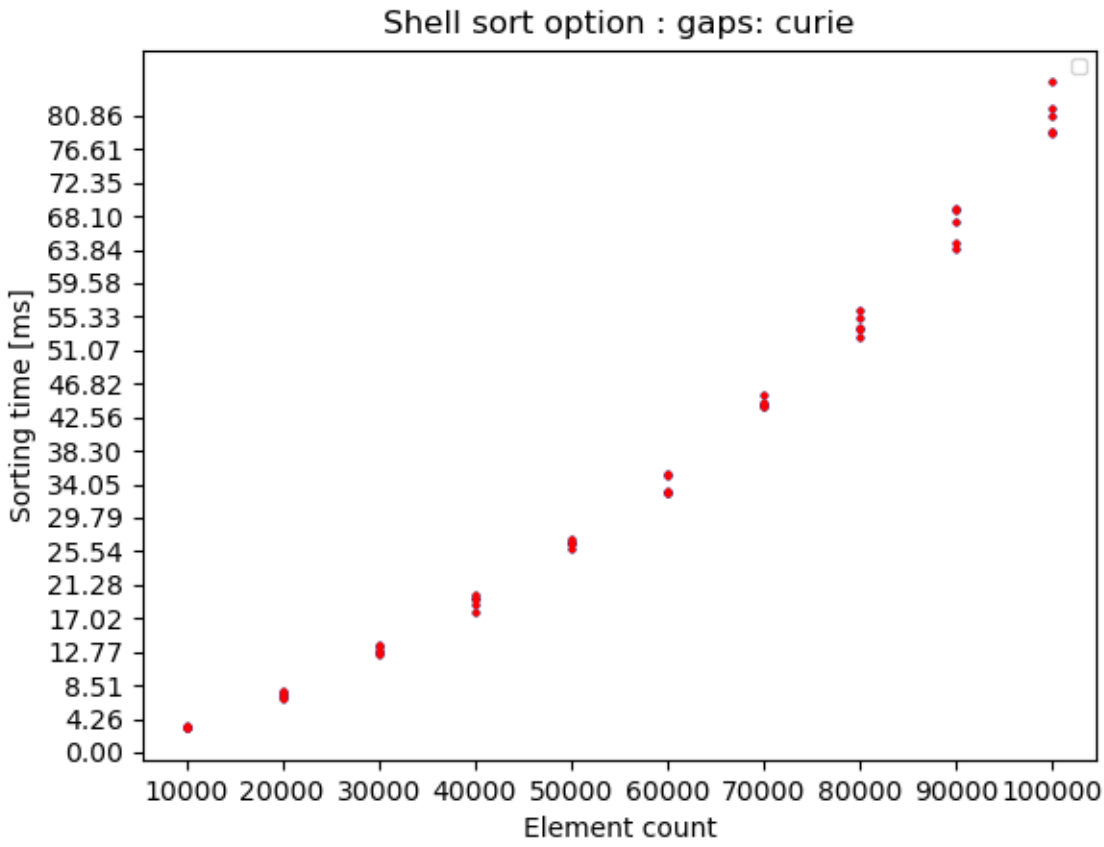


Wykresy dla sortowania przez kopcowanie:

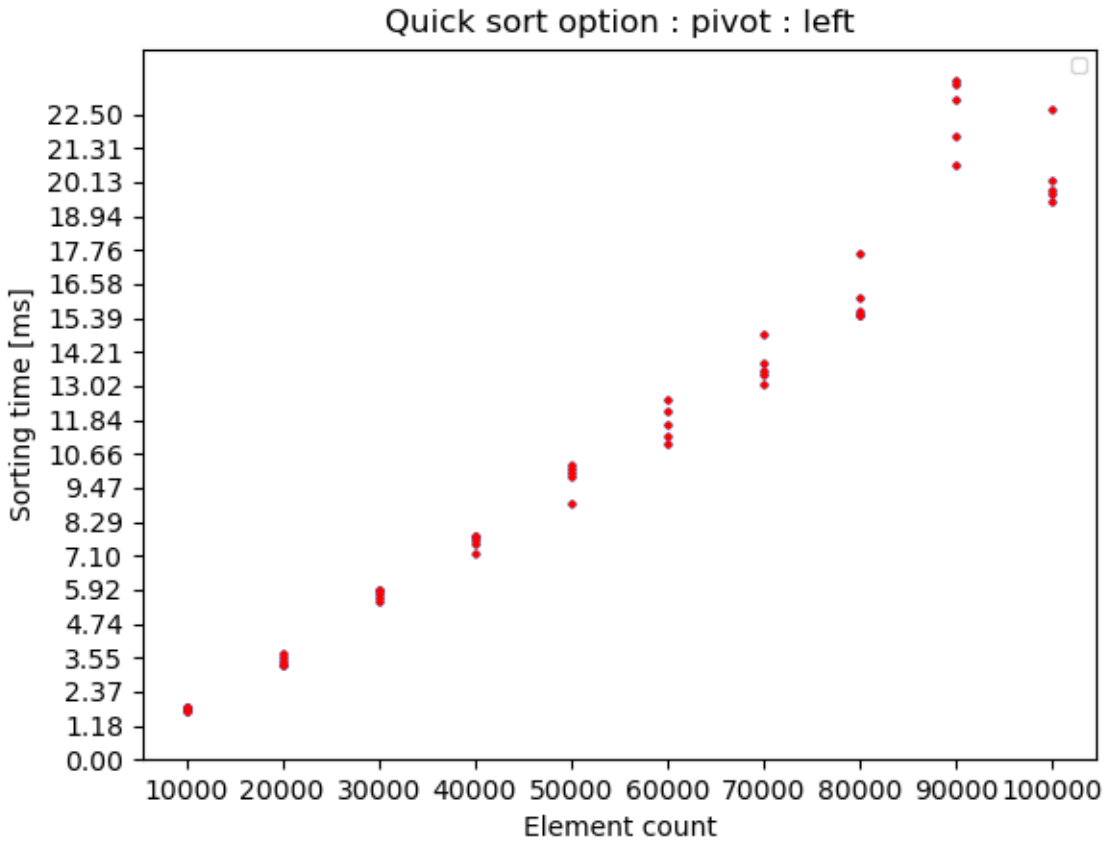


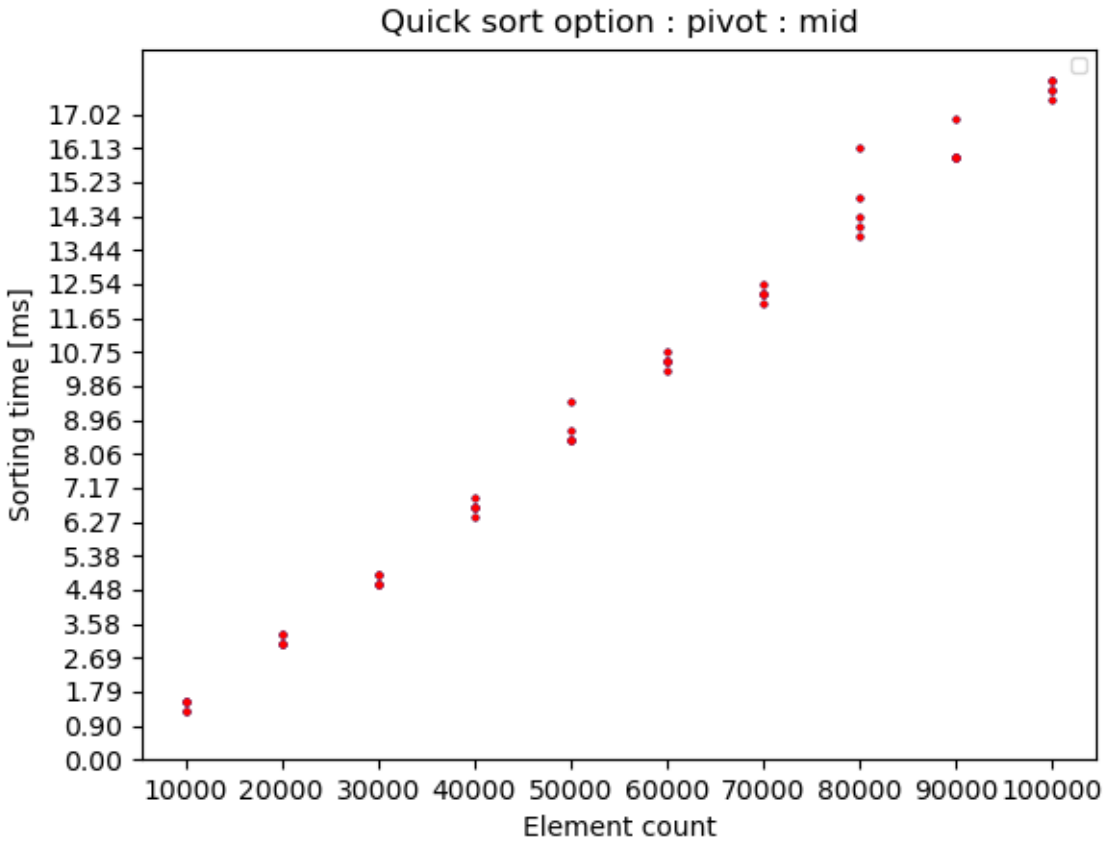
Wykresy dla sortowania Shella:

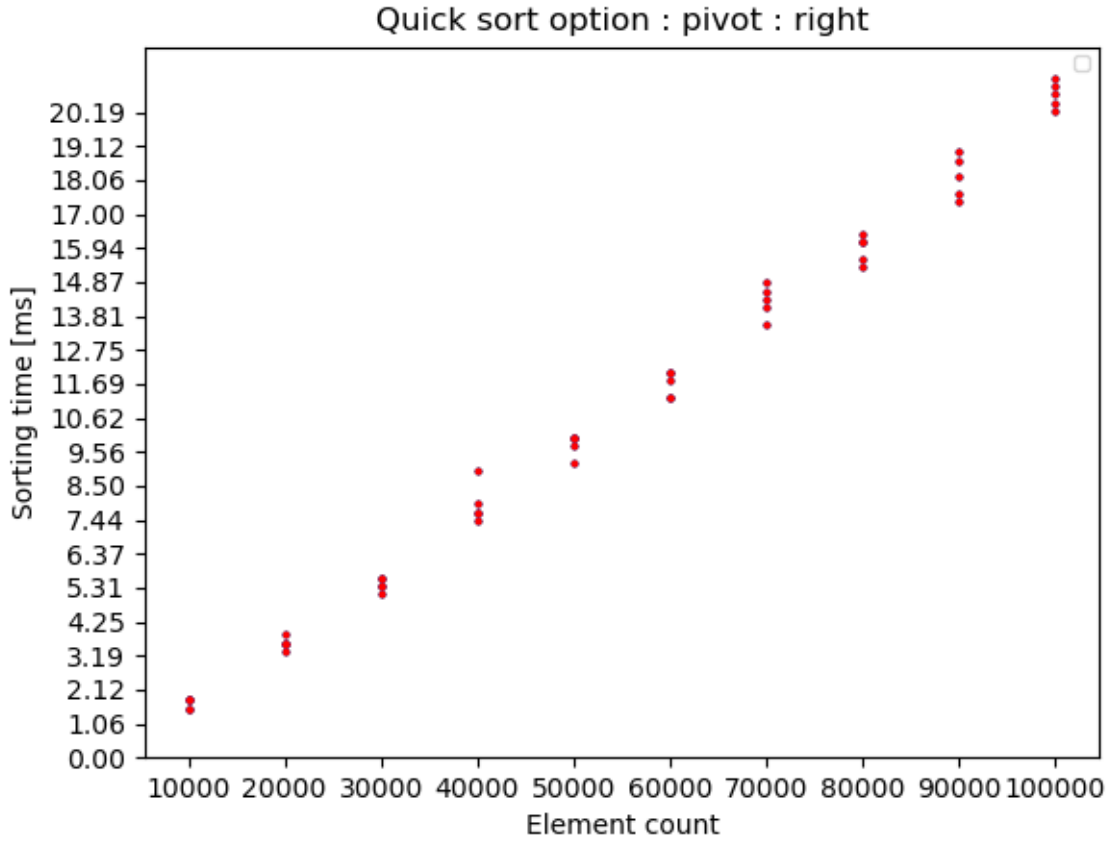


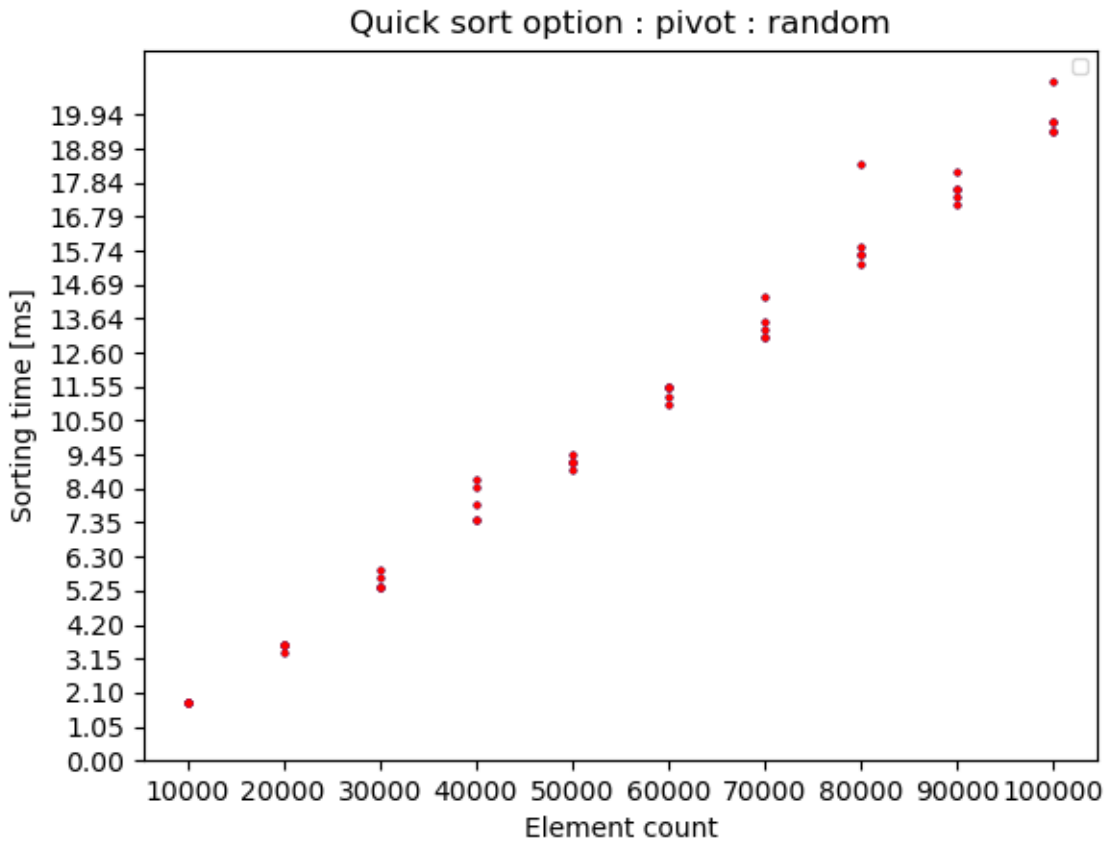


Wykresy dla sortowania szybkiego:


Quick sort option : pivot : left
Sorting time [ms]
22.50
21.31
20.13
18.94
17.76
16.58
15.39
14.21
13.02
11.84
10.66
9.47
8.29
7.10
5.92
4.74
3.55
2.37
1.18
0.00
10000
20000
30000
40000
50000
60000
70000
80000
90000
100000
Element count


Quick sort option : pivot : mid
Sorting time [ms]
17.02
16.13
15.23
14.34
13.44
12.54
11.65
10.75
9.86
8.96
8.06
7.17
6.27
5.38
4.48
3.58
2.69
1.79
0.90
0.00
10000 20000 30000 40000 50000 60000 70000 80000 90000 100000
Element count


Quick sort option : pivot : right
Sorting time [ms]
20.19
19.12
18.06
17.00
15.94
14.87
13.81
12.75
11.69
10.62
9.56
8.50
7.44
6.37
5.31
4.25
3.19
2.12
1.06
0.00
10000 20000 30000 40000 50000 60000 70000 80000 90000 100000
Element count



3. Wpływ wstępnego posortowania danych na szybkość sortowania.

Wielkość zbioru	Typ algorytmu sortującego [ms]		
	Sortowanie przez wstawianie uint32_t wstępne przesortowanie : 0.00	Sortowanie przez wstawianie uint32_t wstępne przesortowanie : 0.33	Sortowanie przez wstawianie uint32_t wstępne przesortowanie : 0.50
10 000	131	92	87
20 000	550	483	420
30 000	1150	970	930
40 000	2160	1640	1520
50 000	3305	2480	2363
60 000	4850	3590	3540
70 000	6667	4650	4520
80 000	8550	6205	5639
90 000	10500	7850	7161
100 000	12750	9552	8809

Wielkość zbioru	Typ algorytmu sortującego [ms]		
	Sortowanie przez kopcowanie wstępne przesortowanie : 0.00	Sortowanie przez kopcowanie wstępne przesortowanie : 0.33	Sortowanie przez kopcowanie wstępne przesortowanie : 0.50
10 000	3.50	3.52	3.48
20 000	7.52	7.53	7.59
30 000	11.43	11.38	11.42
40 000	17.07	17.12	17.05
50 000	21.41	21.34	21.48
60 000	25.02	25.07	25.02
70 000	30.46	30.45	30.49
80 000	34.72	34.68	34.71
90 000	39.56	39.53	39.58
100 000	44.23	44.21	44.23

Wielkość zbioru	Typ algorytmu sortującego [ms]		
	Sortowanie Shella, potega dwojki wstępne przesortowanie : 0.00	Sortowanie Shella, potega dwojki wstępne przesortowanie : 0.33	Sortowanie Shella, potega dwojki wstępne przesortowanie : 0.50
10 000	3.3	3.20	2.98
20 000	7.2	7.25	7.02
30 000	10.7	10.23	10.02
40 000	16.2	15.92	15.45
50 000	22.3	21.54	20.98
60 000	25.3	24.76	24.56
70 000	30.2	30.02	29.56
80 000	38.2	37.45	36.22
90 000	43.7	41.23	39.43
100 000	50.1	49.67	49.54

Wielkość zbioru	Typ algorytmu sortującego [ms]		
	Sortowanie szybkie pivot losowy, wstępne przesortowanie : 0.00	Sortowanie szybkie pivot losowy, wstępne przesortowanie : 0.33	Sortowanie szybkie pivot losowy, wstępne przesortowanie : 0.50
10 000	1.78	1.65	1.53
20 000	3.32	3.26	3.04
30 000	7.58	5.63	5.31
40 000	8.70	7.84	7.42
50 000	9.21	9.12	9.03
60 000	11.52	11.26	11.00
70 000	13.57	13.45	13.31
80 000	15.67	15.64	15.61
90 000	18.72	17.89	17.40
100 000	19.45	19.31	19.24

4. Omówienie wyników oraz wnioski

Na podstawie pomiarów możemy zauważyć, że algorytmy zachowywały się zgodnie z swoimi własnościami. Analizując wyniki liczbowe oraz rozłożenie na wykresie dla algorytm sortowania przez kopcowanie możemy zauważyć, że wyniki sortowania dla tego algorytmu w przypadku zwiększania ilości elementów wynosi O(n log n). Algorytm przyśpieszał wraz ze wzrostem ilości posortowanych elementów.

Dla algorytmu sortowania przez wstawianie wyniki są również jednoznaczne. Algorytm ten w przypadku dla danych losowych odpowiada jego złożoności obliczeniowej O(n2). Zgodnie z założeniami i teorią ten algorytm wypadł najgorzej na tle wszystkich pozostałych algorytmów, co łatwo możemy dostrzec na wykresie porównującym wszystkie algorytm ze sobą. Zwiększanie się ilości elementów w zbiorze znacząco wpływa na wydajność tego algorytmu. Algorytm ten również przyśpieszał wraz z ilością posortowanych elementów w tablicy.

Wyniki sortowania Shella odpowiadają przewidywanym wzorcem, zachowując się zgodnie z oczekiwaniami. Ten algorytm, ogólnie rzecz biorąc, charakteryzuje się złożonością O(n^3/2), co potwierdzają uzyskane wyniki. Dodatkowo, obserwuje się różnice w efektywności algorytmu w zależności od wybranego sposobu obliczania odstępów. Zgodnie z teorią, algorytm Shella z odstępem według metody Ciura wykazuje gorszą wydajność niż ten z odstępem bazowym. W przypadku tego algorytmu nie obserwujemy dużego przyśpieszenia w porównaniu do sortowania przez wstawianie.

Ostatni z algorytmów sortowania, czyli algorytm sortowania szybkiego zachowuje się również zgodnie z założeniami. Pomiary zostały przeprowadzone na każdym możliwym ustawieniu pivota, jednak nie wpływają one na wynik sortowania w przypadku danych wygenerowanych w sposób losowy. Na ostatni algorytm wstępne posortowanie tablicy nie ma znaczenia.