Sprawozdanie Projekt AiZO

Prowadzący: dr inż. Dariusz Banasiak

Patryk Pietrzyk 272971

grupa 2, czwartek, 13:15

Wprowadzenie

W tym projekcie analizujemy wydajność różnych algorytmów sortowania, w tym Insertion Sort, Binary Insertion Sort Heap Sort oraz Quick Sort. Każdy z tych algorytmów ma swoje charakterystyczne cechy i złożoność obliczeniową która wpływa na jego efektywność w sortowaniu danych.

Badane Algorytmy:

Insertion Sort

Insertion Sort jest jednym z najprostszych algorytmów sortowania. Polega na przeglądaniu tablicy od lewej do prawej i porównywaniu każdego elementu z jego poprzednikiem, przenosząc go w odpowiednie miejsce. W przypadku średnim jego złożoność obliczeniowa wynosi O(n^2), gdzie n to liczba elementów w tablicy. W przypadku optymistycznym, gdy tablica jest prawie posortowana, złożoność ta zmniejsza się do O(n), ale w przypadku pesymistycznym, gdy tablica jest odwrotnie posortowana, złożoność wzrasta do O(n^2).

Binnary Insertion Sort

Binary Insertion Sort jest modyfikacją algorytmu Insertion Sort, która wykorzystuje wyszukiwanie binarne do znalezienia odpowiedniego miejsca dla każdego elementu w tablicy. Złożoność obliczeniowa tego algorytmu jest podobna do algorytmu Insertion Sort i wynosi O(n^2) w przypadku średnim oraz pesymistycznym.

Heap Sort

Heap Sort wykorzystuje strukturę kopca (heap) do sortowania danych. Polega na budowaniu kopca z danych wejściowych, a następnie usuwaniu elementów z kopca i umieszczaniu ich na odpowiednich pozycjach w posortowanej tablicy. Złożoność obliczeniowa tego algorytmu wynosi O(nlogn) w przypadku średnim oraz pesymistycznym.

Quick Sort

Quick Sort jest jednym z najszybszych algorytmów sortowania. Działa na zasadzie dziel i zwyciężaj, dzieląc tablicę na mniejsze części, sortując je osobno, a następnie łącząc w pełną posortowaną tablicę. Złożoność obliczeniowa Quick Sort wynosi O(nlogn) w przypadku średnim, ale może osiągnąć O(n^2) w przypadku pesymistycznym gdy wybór pivota nie jest optymalny (chociaż w tym projekcie nie poruszam tematu wyboru pivota).

Podsumowując, każdy z tych algorytmów ma swoje zalety i wady, które warto brać pod uwagę w zależności od charakterystyki danych wejściowych i oczekiwanego czasu sortowania. W tym projekcie dokładniej przyjrzymy się ich wydajności w różnych scenariuszach sortowania.

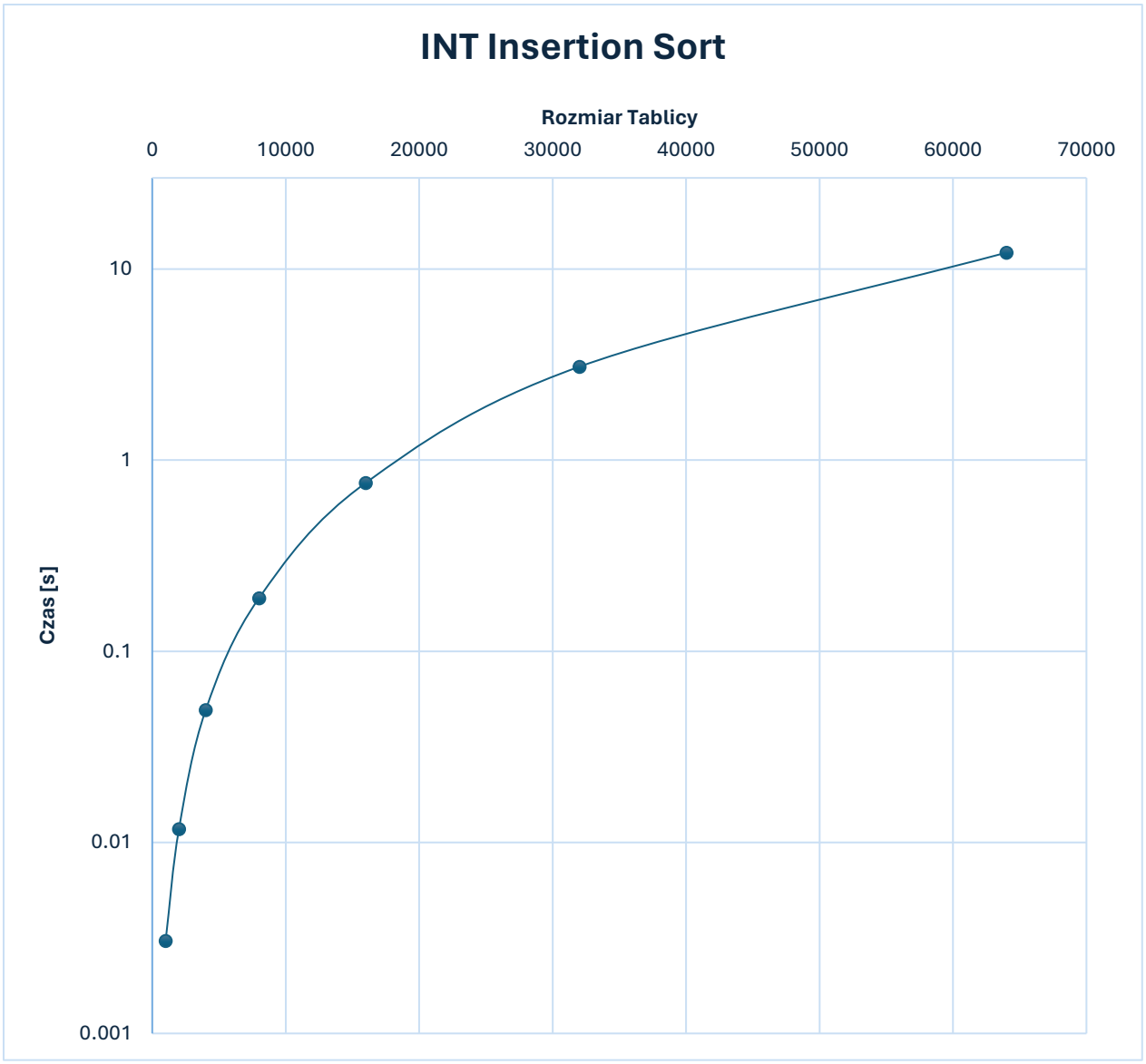
Plan Eksperymentu

1. Założenia co do rozmiaru tablic:
    a. W celu uzyskania miarodajnych wyników, czasy sortowania algorytmów powinny być większe niż 0.001 sekundy.
    b. Dla algorytmów Insertion Sort i Binary Insertion Sort przyjęto rozmiary tablic: 1000, 2000, 4000, 8000, 16000, 32000, 64000. Ich mniejsza wydajność wymaga stosunkowo mniejszych danych do testów.
    c. Natomiast dla algorytmów Heap Sort i Quick Sort, które są bardziej wydajne, użyto większych rozmiarów tablic: 10000, 20000, 40000, 80000, 160000, 320000, 640000.
    d. Każdy test jest powtarzany 30 razy dla algorytmów Insertion Sort i Binary Insertion Sort oraz 80 razy dla algorytmów Heap Sort i Quick Sort, aby zminimalizować błędy pojedynczego przypadku.
2. Sposób generowania tablic:
    a. Wykorzystano podejście obiektowe, korzystając z szablonów (template<typename T>), co pozwoliło na zmniejszenie ilości klas i uprościło strukturę kodu.
    b. Generacja losowych wartości odbywa się przy użyciu funkcji rand() oraz srand(), a następnie wartości są rzutowane do odpowiedniego typu danych za pomocą static_cast<T>.
    c. Sprawdzanie typu danych wykonuje się przy pomocy std::is_integral_v<T>.
    d. Tablice są dynamicznie alokowane za pomocą operatora new, a kopia tworzona jest poprzez głęboką kopię (deep copy), co zapewnia niezależność danych.
    e. Problemowe okazało się porównanie wartości (FLOAT). Poradzono sobie z tym stosując technikę z epsilon.
3. Sposób pomiaru czasu:
    a. Do pomiaru czasu wykorzystano bibliotekę std::chrono::high_resolution_clock, co pozwala na precyzyjny pomiar czasu wykonywania algorytmów.
    b. Wyniki pomiarów zapisywane są do plików w formacie .csv, co umożliwia łatwą analizę oraz generowanie wykresów.
    c. Każdy wynik jest zapisywany zgodnie z typem danych oraz rodzajem algorytmu, co pozwala na efektywną analizę wyników.
4. Specjalne przypadki tablic:
    a. Dla algorytmu Insertion Sort przeprowadzono testy dla tablic o specjalnych cechach, takich jak:
    b. Tablica całkowicie losowa,
    c. Tablica posortowana rosnąco,
    d. Tablica posortowana malejąco,
    e. Tablica posortowana częściowo, z 33% i 66% początkowych elementów już posortowanych.
    f. Wykorzystano funkcję std::sort do wygenerowania poprawnie posortowanych danych, co pozwoliło na sprawdzenie odporności na przypadki specjalne.

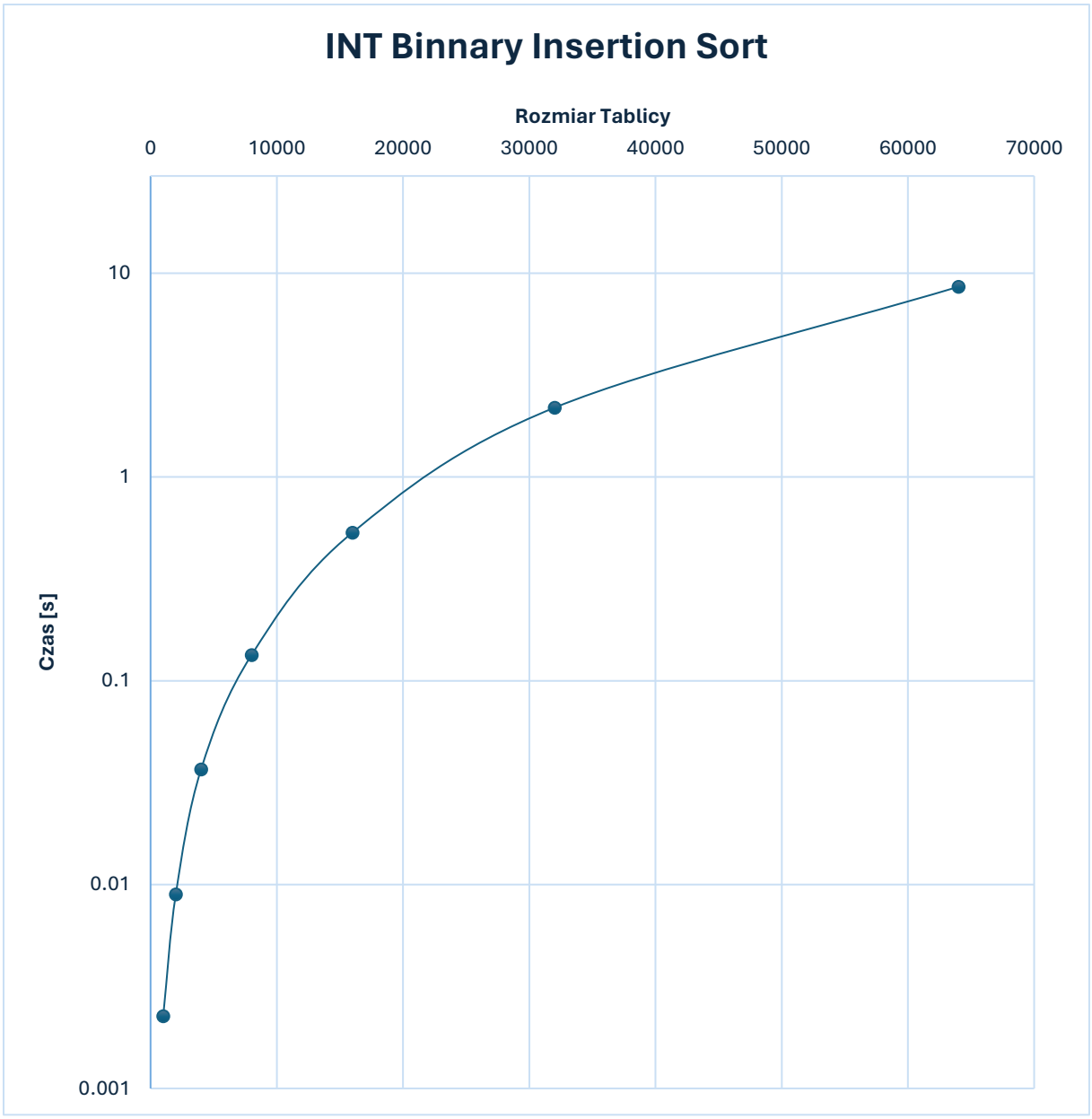
5. Podsumowanie planu eksperymentalnego:
    a. Program podzielono na sekcje Menu oraz testForReport, gdzie generowane są tablice dla różnych typów danych i testowane z użyciem każdego z algorytmów sortowania.
    b. Testy przeprowadzane są dla różnych rozmiarów tablic oraz z odpowiednią ilością powtórzeń, aby uzyskać miarodajne wyniki.
    c. Dzięki dokładnemu planowi eksperymentu możliwe będzie porównanie wydajności poszczególnych algorytmów w różnych scenariuszach sortowania oraz identyfikacja ich mocnych i słabych stron.

Omówienie przebiegu eksperymentów i przedstawienie uzyskanych wyników

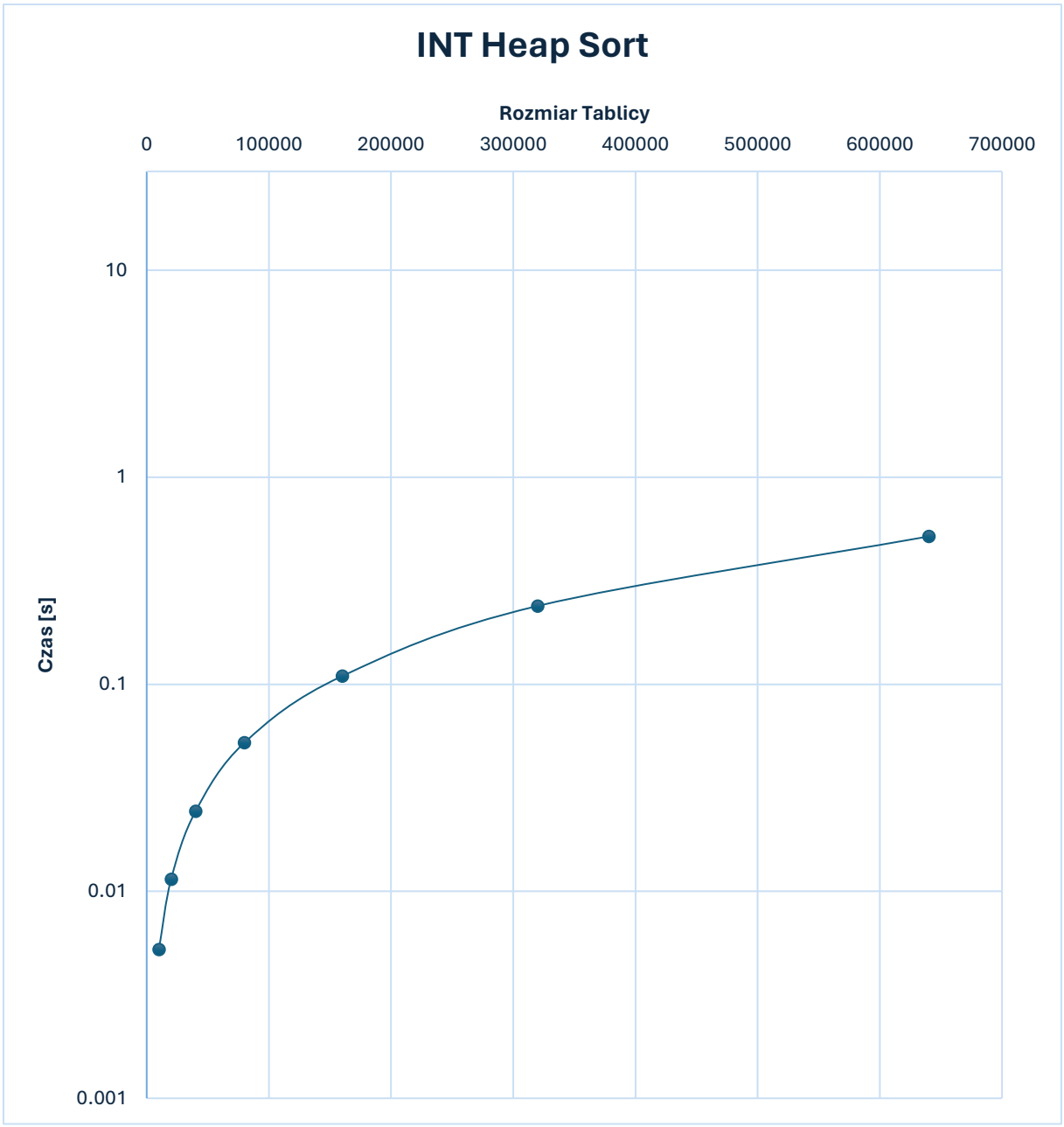
Wyniki dla zależności czasu sortowania od rozmiaru tablicy i typu danych (tabele oraz wykresy poniżej):



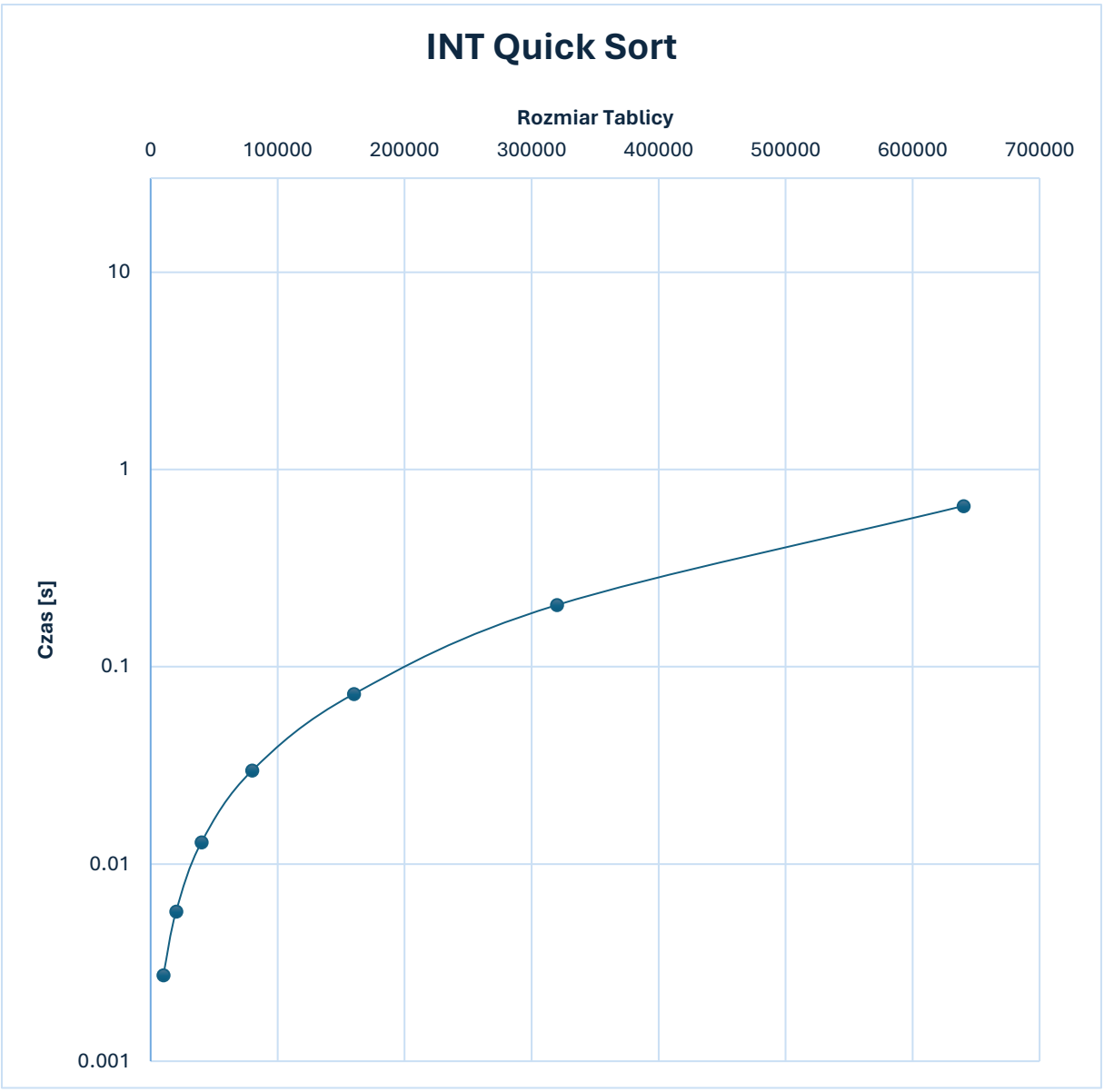
Size	Time [s]
1000	0.0031
2000	0.0117
4000	0.0492
8000	0.1897
16000	0.7606
32000	3.082
64000	12.1851



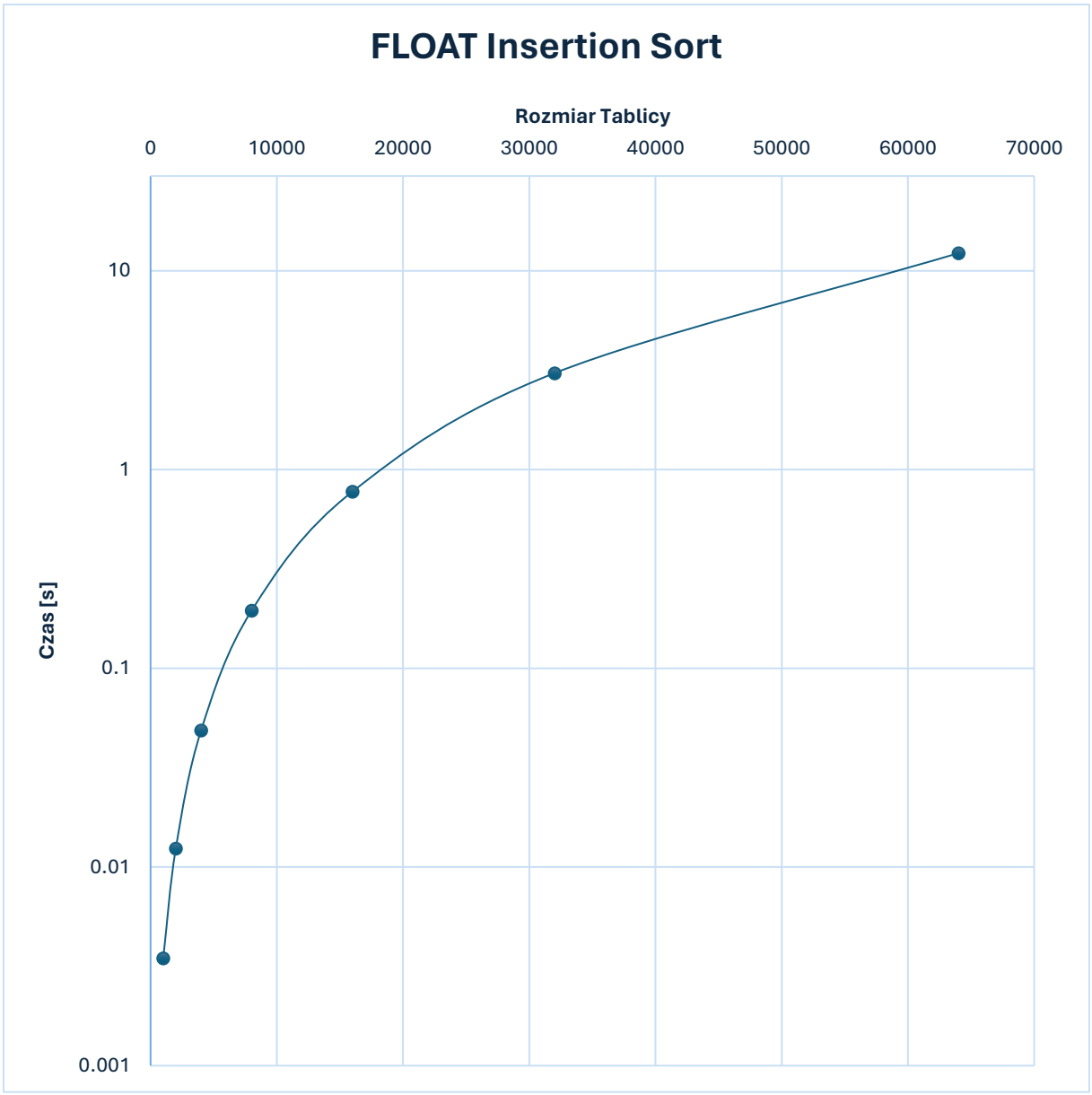
Size	Time [s]
1000	0.0023
2000	0.009
4000	0.0368
8000	0.1339
16000	0.5342
32000	2.1888
64000	8.5812



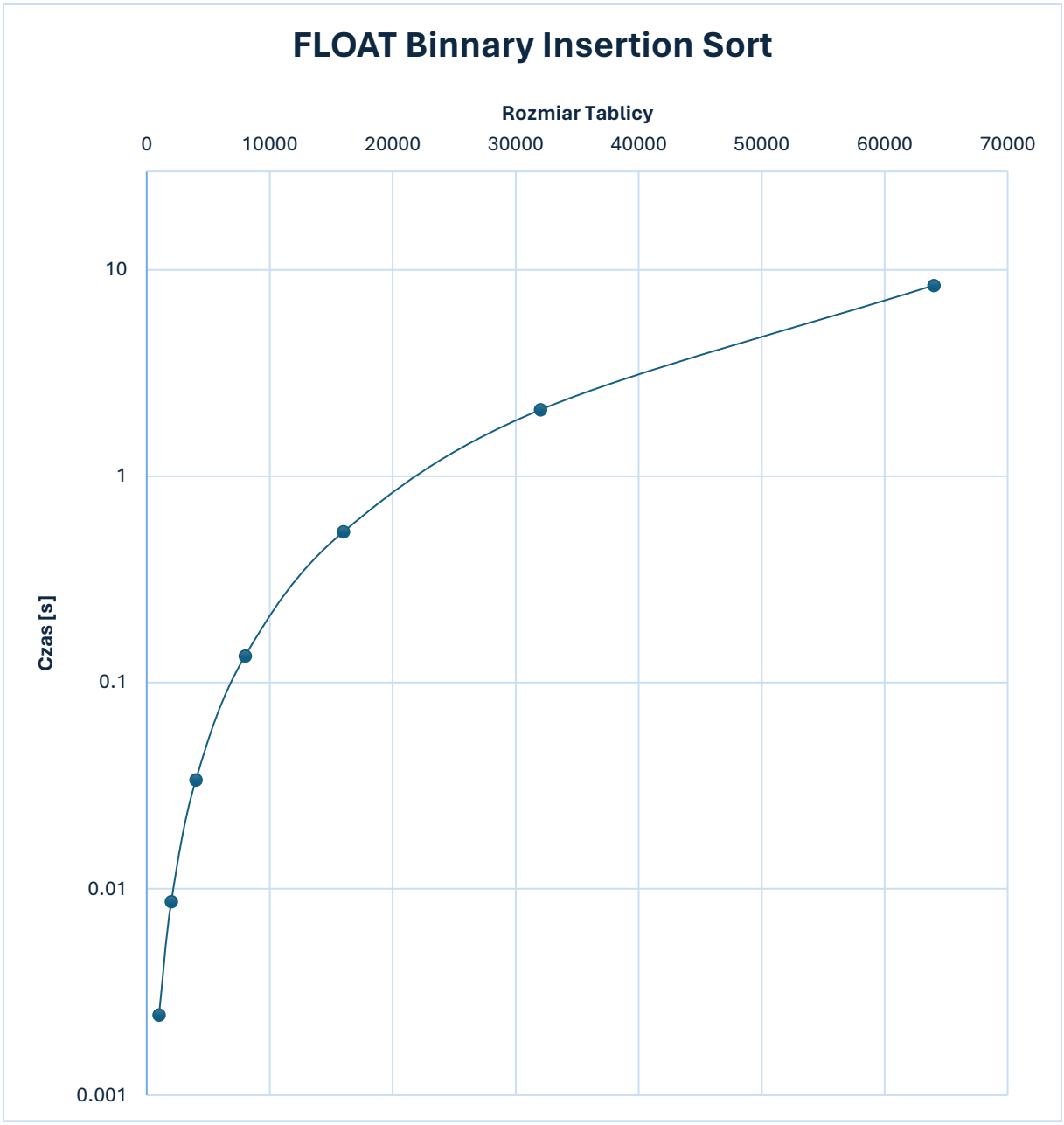
Size	Time [s]
10000	0.0052
20000	0.0114
40000	0.0243
80000	0.0522
160000	0.1095
320000	0.2389
640000	0.5178



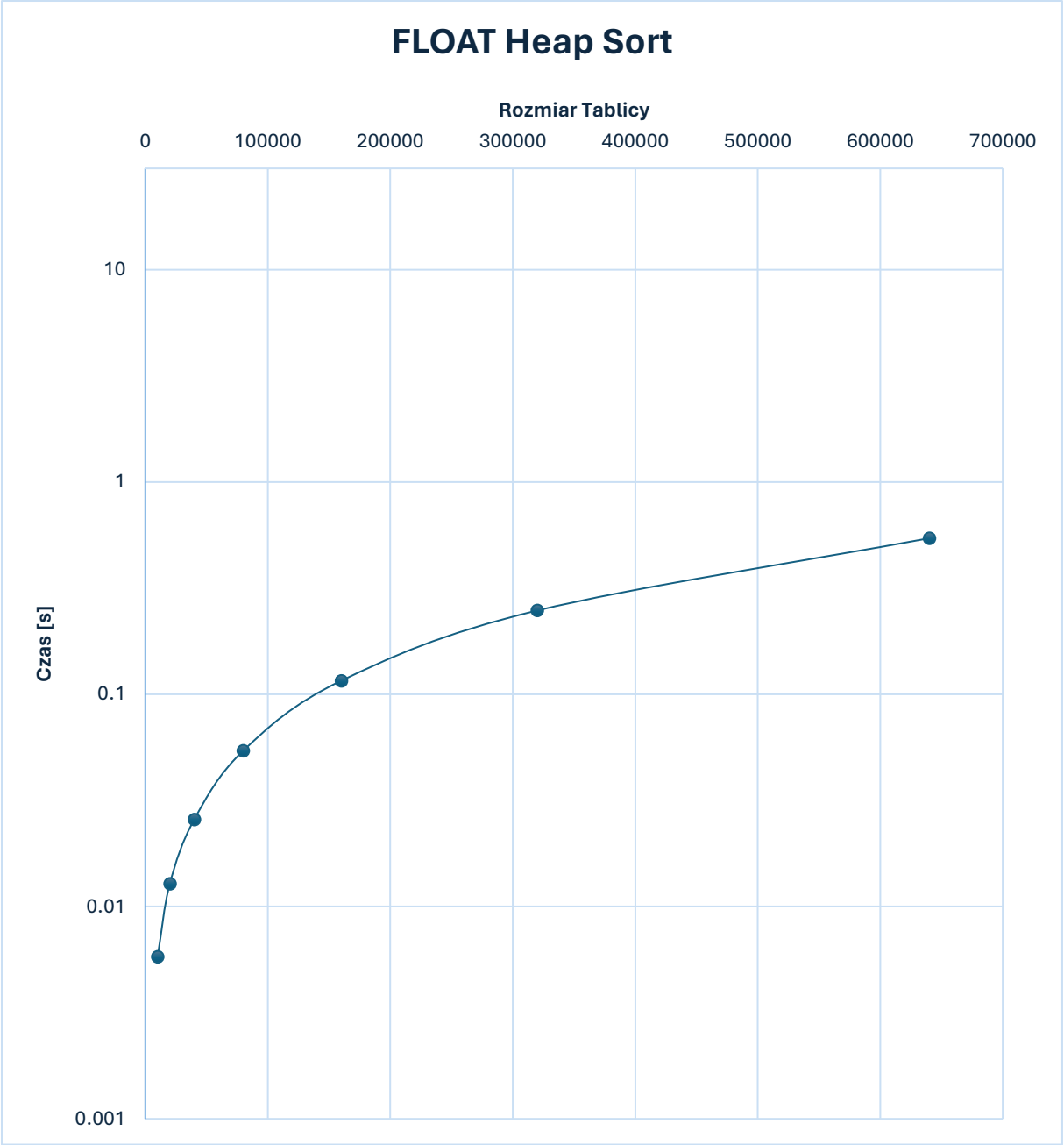
Size	Time [s]
10000	0.0027
20000	0.0057
40000	0.0129
80000	0.0298
160000	0.0728
320000	0.2056
640000	0.6532



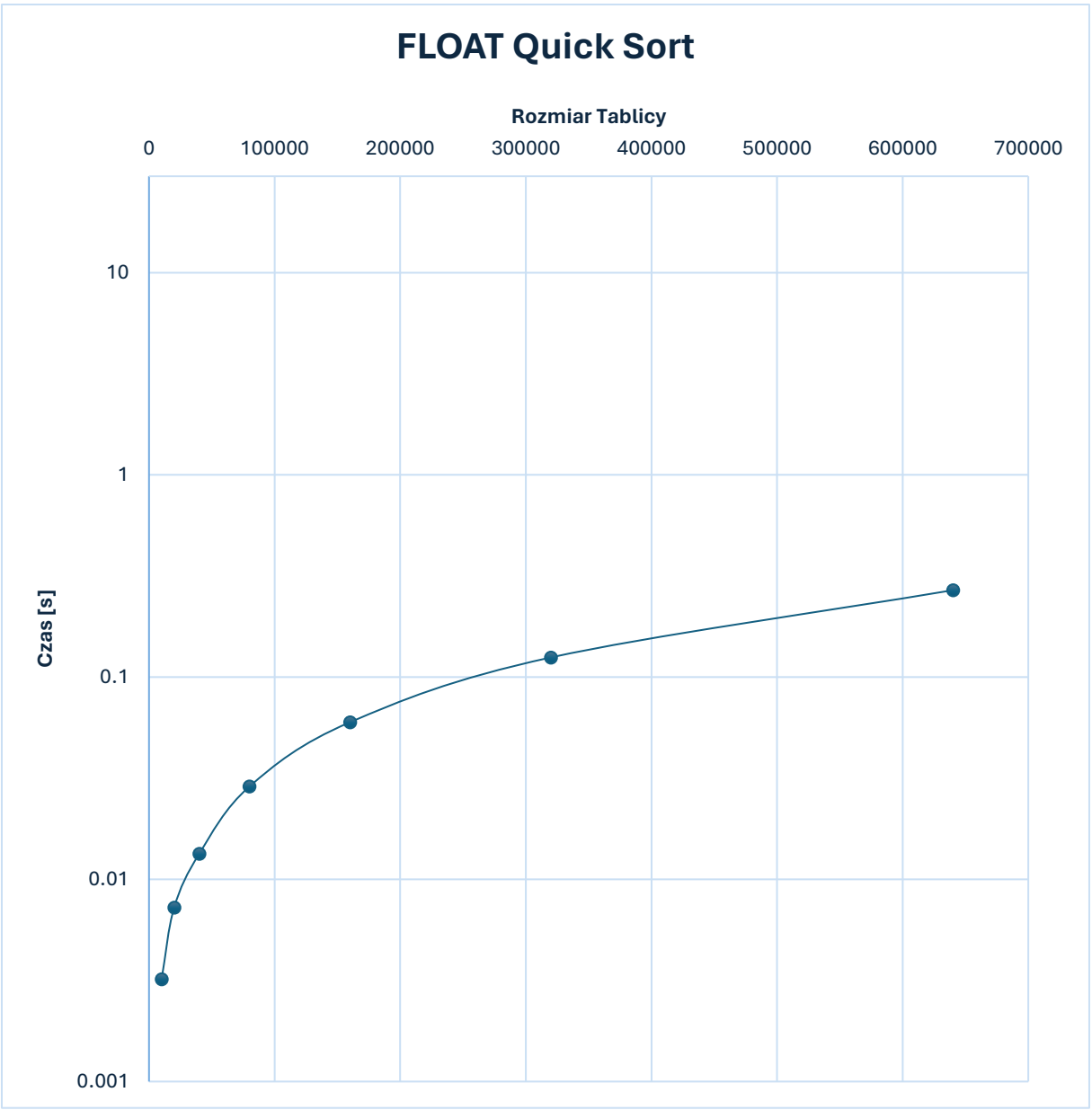
Size	Time [s]
1000	0.0035
2000	0.0124
4000	0.0486
8000	0.1948
16000	0.7748
32000	3.0552
64000	12.271



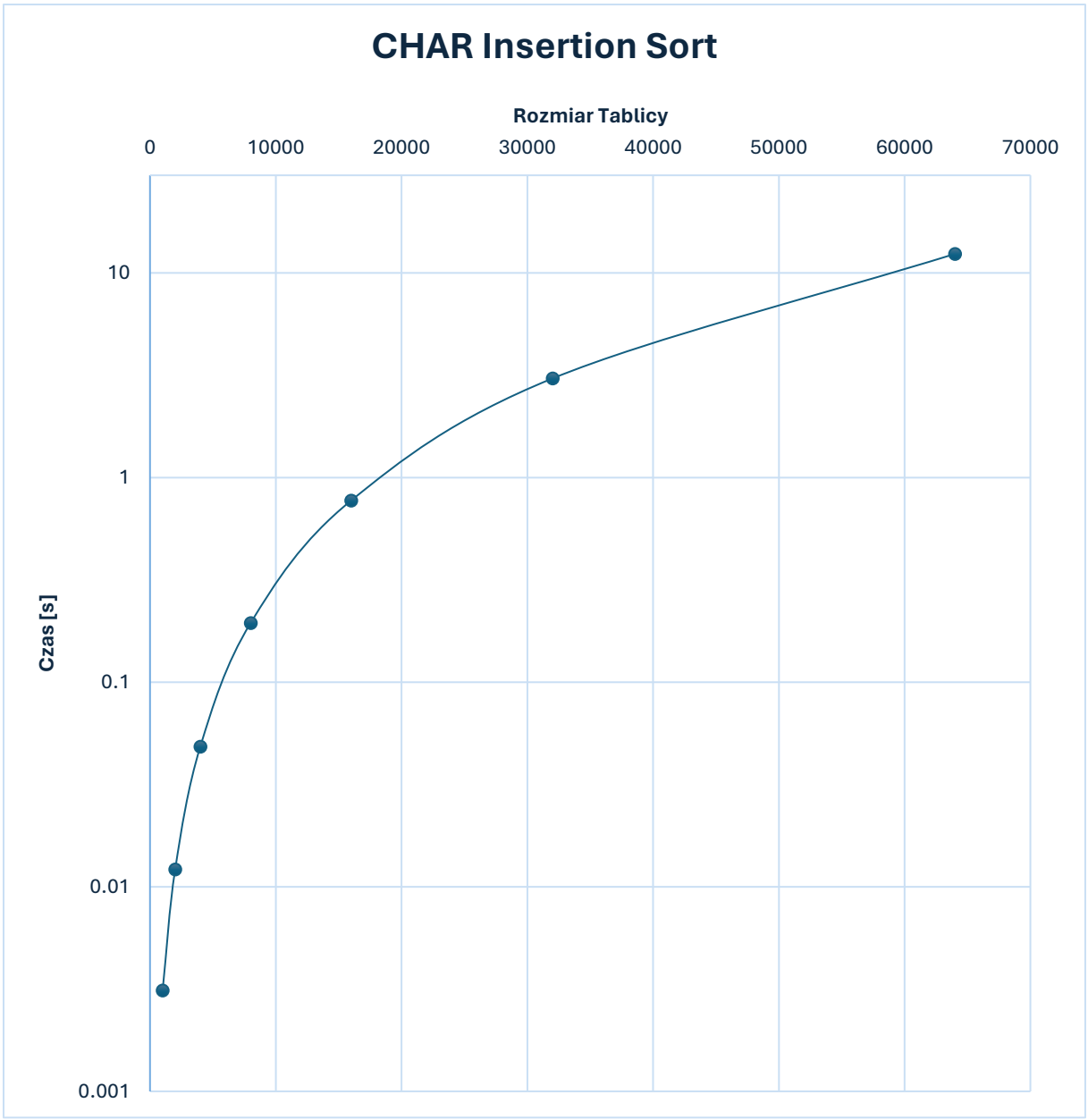
Size	Time [s]
1000	0.0024
2000	0.0087
4000	0.0338
8000	0.1346
16000	0.5381
32000	2.0975
64000	8.4145



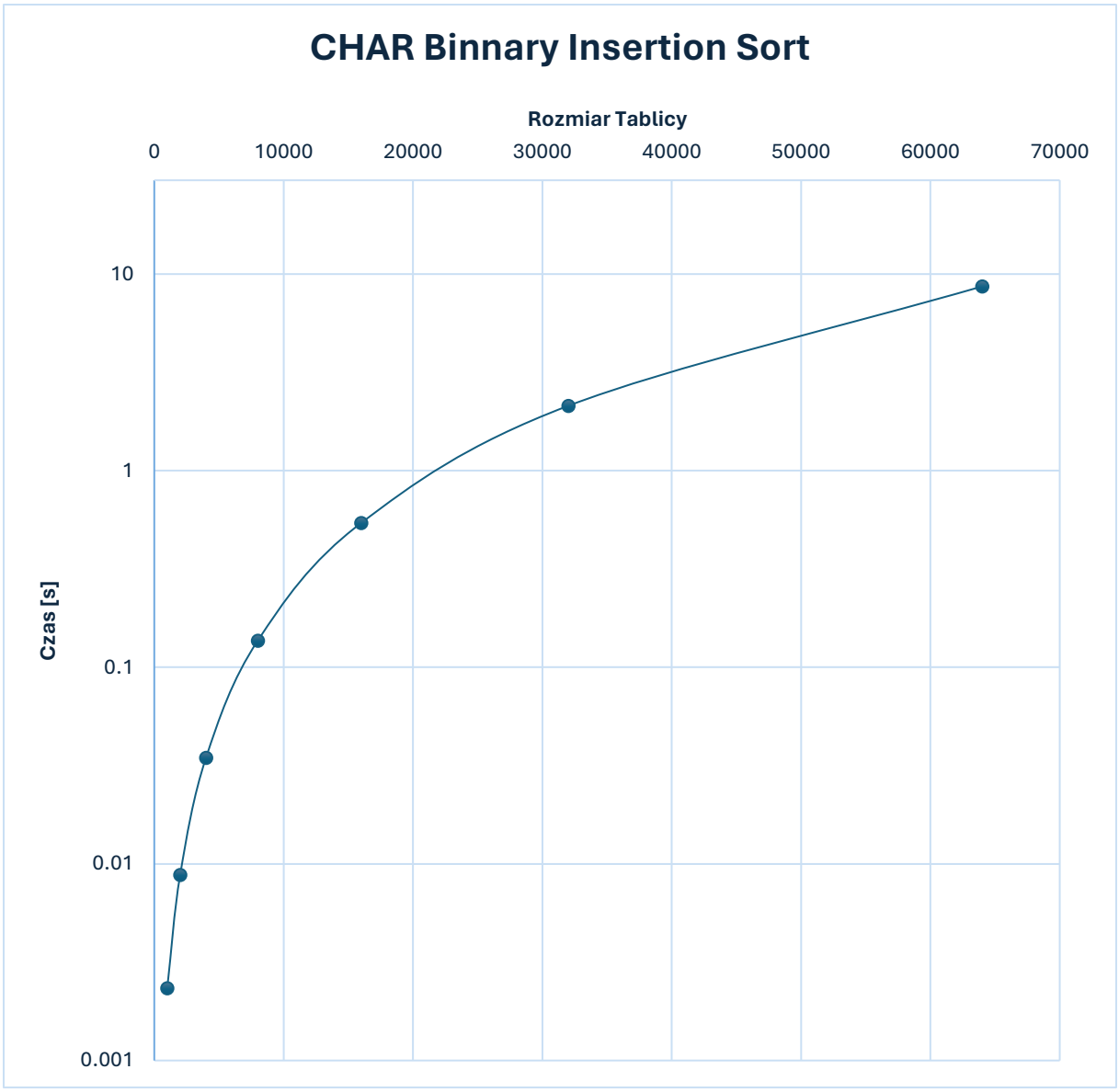
Size	Time [s]
10000	0.0058
20000	0.0128
40000	0.0258
80000	0.0543
160000	0.1158
320000	0.2481
640000	0.5433



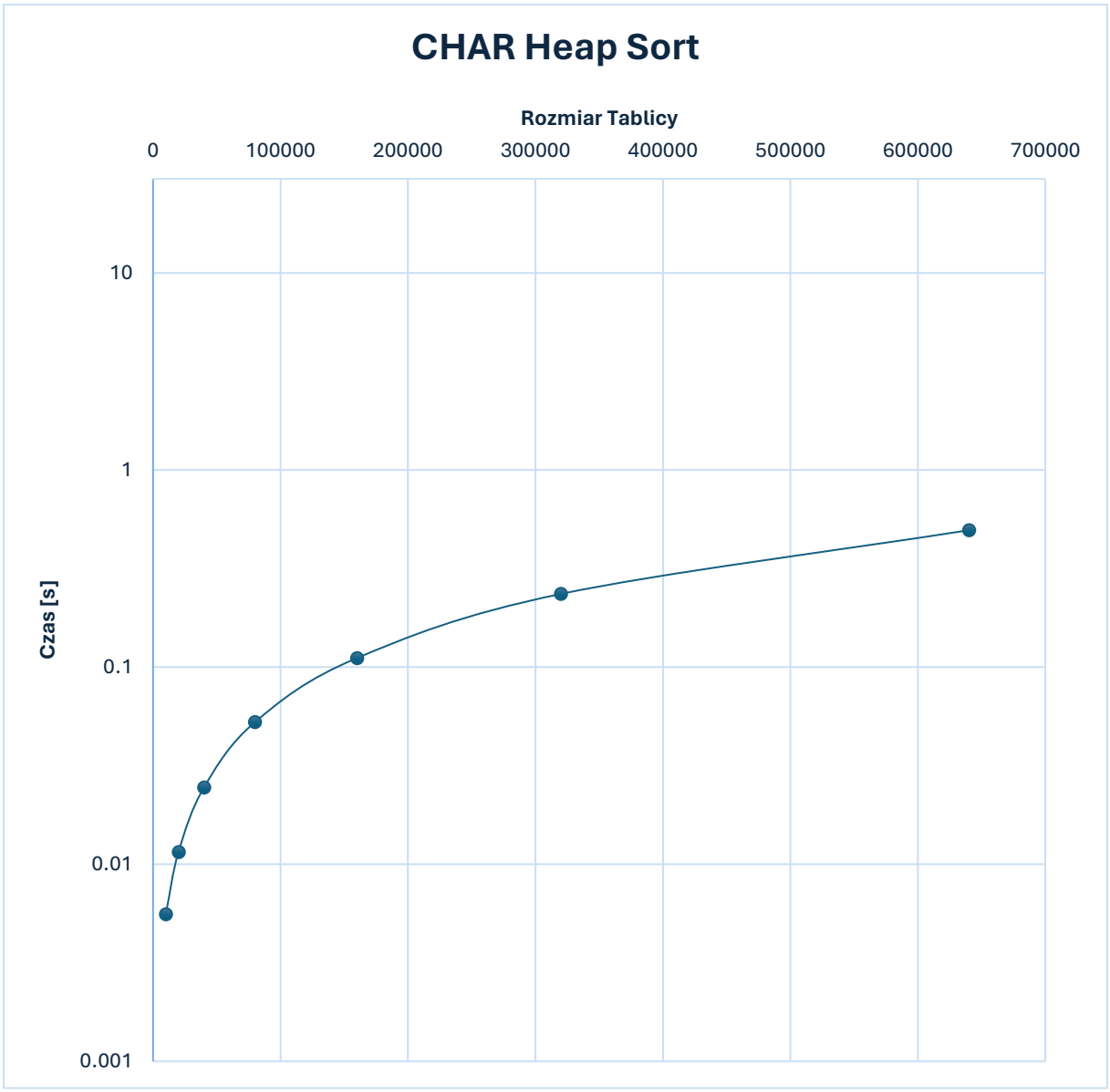
Size	Time [s]
10000	0.0032
20000	0.0072
40000	0.0134
80000	0.0288
160000	0.0598
320000	0.1252
640000	0.2692



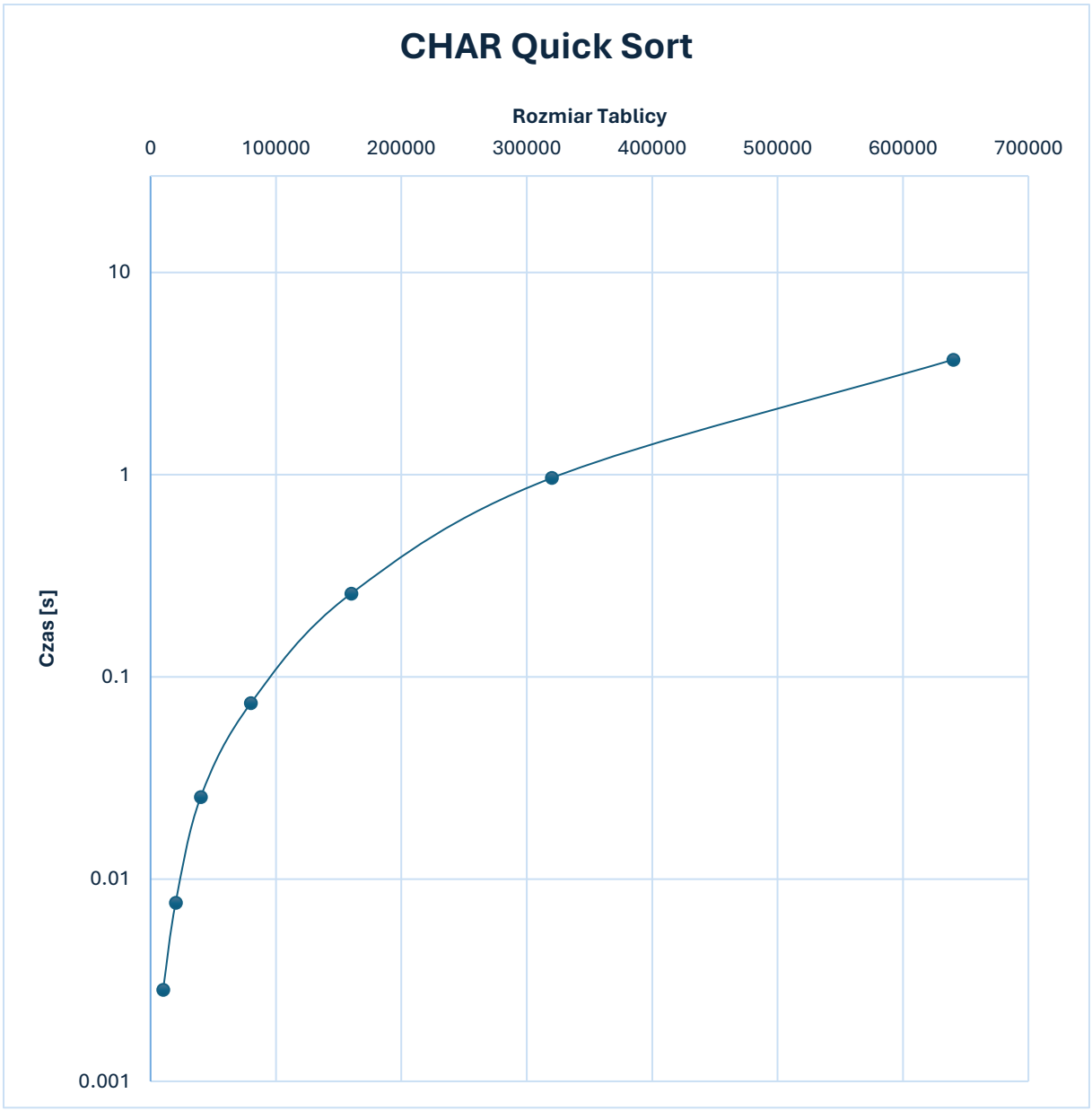
Size	Time [s]
1000	0.0031
2000	0.0122
4000	0.0484
8000	0.1945
16000	0.7726
32000	3.0491
64000	12.3826



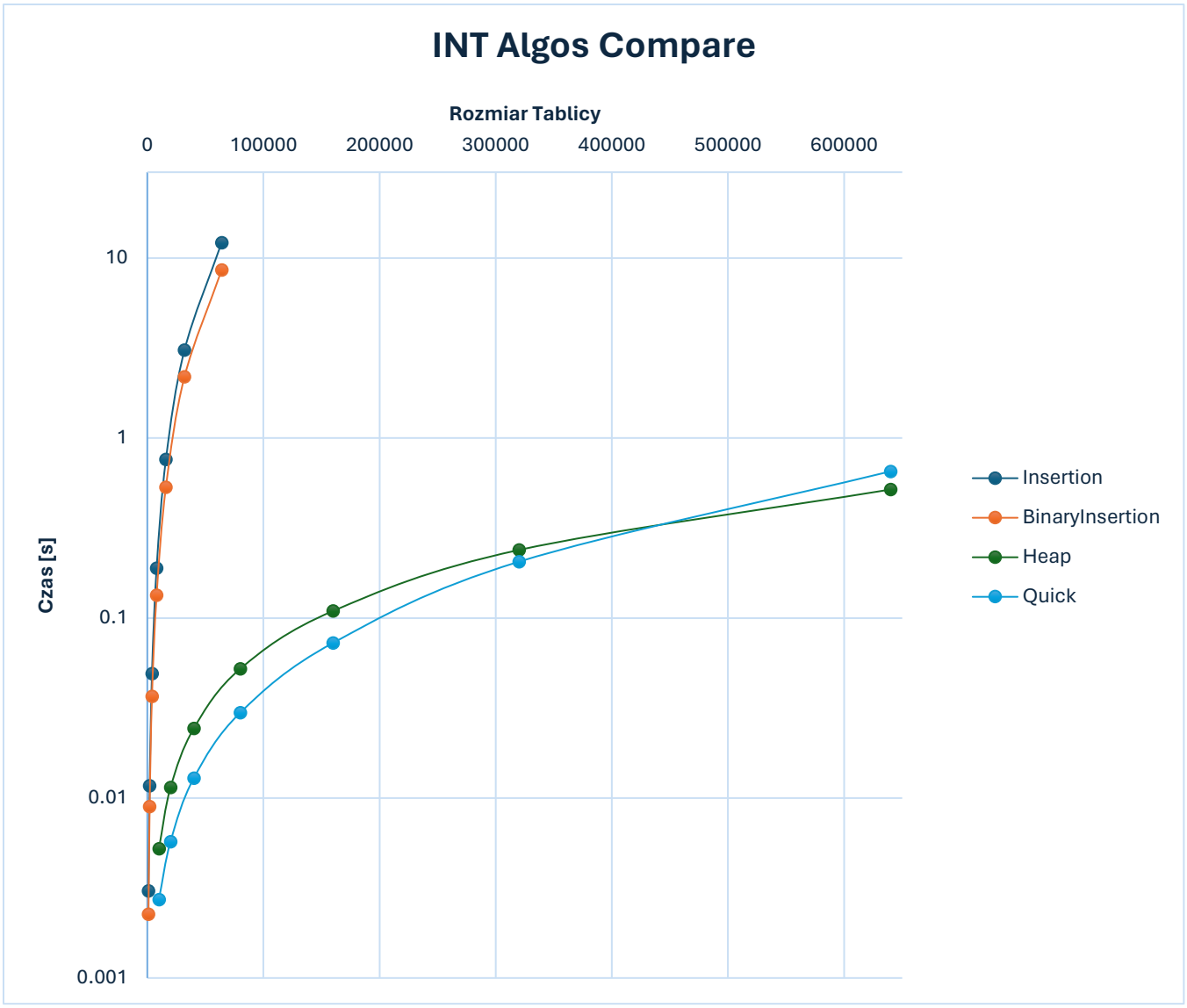
Size	Time [s]
1000	0.0023
2000	0.0088
4000	0.0347
8000	0.1366
16000	0.5422
32000	2.1366
64000	8.6531



Size	Time [s]
10000	0.0056
20000	0.0116
40000	0.0245
80000	0.0526
160000	0.1112
320000	0.2352
640000	0.495



Size	Time [s]
10000	0.0028
20000	0.0076
40000	0.0255
80000	0.0742
160000	0.2583
320000	0.9638
640000	3.7039


INT Algos Compare
Rozmiar Tablicy
0
100000
200000
300000
400000
500000
600000
Czas [s]
10
1
0.1
0.01
0.001
Insertion
BinaryInsertion
Heap
Quick

Wnioski dla części rozmiaru tablic

Wydajność algorytmów: Algorytmy Quick Sort i Heap Sort wykazują najwyższą wydajność dla wszystkich rodzajów danych i rozmiarów tablic. Ich złożoność obliczeniowa O(nlogn) pozwala na efektywne sortowanie nawet dużych zbiorów danych.

Algorytmy Insertion Sort i Binary Insertion Sort mają wyższą złożoność czasową O(n^2) i są znacznie wolniejsze, szczególnie dla większych tablic. Ich wydajność maleje wraz ze wzrostem rozmiaru tablicy.

Różnice między Insertion Sort a Binary Insertion Sort: Binary Insertion Sort wykazuje nieznacznie lepszą wydajność niż klasyczny Insertion Sort, zwłaszcza dla większych tablic. Jest to spowodowane wykorzystaniem wyszukiwania binarnego do znalezienia odpowiedniego miejsca dla każdego elementu, co zmniejsza liczbę porównań.

Różnice w czasie sortowania między Insertion Sort a Binary Insertion Sort są widoczne, ale nie są znaczące dla małych tablic.

Wpływ rodzaju danych na czas sortowania: Dla wszystkich algorytmów, czas sortowania tablicy liczb całkowitych (INT) jest najkrótszy, co może być związane z prostotą operacji porównywania liczb całkowitych.

Dla tablic liczb zmiennoprzecinkowych (FLOAT), czas sortowania jest nieco dłuższy niż dla liczb całkowitych, ale wciąż jest akceptowalny.

Dla tablic znaków (CHAR), czas sortowania jest najdłuższy, szczególnie dla algorytmów Insertion Sort i Binary Insertion Sort. To może być spowodowane dodatkowymi operacjami porównywania dla typów danych znakowych. Oraz rozkładem wartości ponieważ (CHAR) przyjmuje wartości (od -128 do 127)

Wpływ rozmiaru tablicy na czas sortowania: Wszystkie algorytmy wykazują wykładniczy wzrost czasu sortowania wraz ze wzrostem rozmiaru tablicy, szczególnie dla algorytmów o złożoności O(n^2) (Insertion Sort, Binary Insertion Sort).

Algorytmy Quick Sort i Heap Sort wykazują bardziej stabilny wzrost czasu sortowania, co może być związane z ich złożonością O(nlogn), która nie jest tak szybko rosnąca jak O(n^2).

Podsumowując: W przypadku małych tablic i prostych danych, różnice między algorytmami mogą być mniej zauważalne, ale dla dużych zbiorów danych wybór odpowiedniego algorytmu sortowania może znacząco wpłynąć na czas wykonania operacji sortowania.

Wyniki dla przypadków szczególnych:

Przeprowadzone na algorytmów sortowania (Insertion, Binnary Insertion, Heap, Quick(w tej kolejności)) Sort dla danych typu INT.

Ascending

Insertion			Heap		
Time [s]	**Size**	**Time avg**	**Time [s]**	**Size**	**Time avg**
0.0001354	10000	0.00014	0.005475	10000	0.005117
0.0001675	10000		0.00523	10000	
0.0001352	10000		0.005006	10000	
0.0001359	10000		0.005311	10000	
0.0001359	10000		0.005172	10000	
0.0001524	10000		0.004962	10000	
0.000135	10000		0.004953	10000	
0.0001357	10000		0.004961	10000	
0.0001359	10000		0.004868	10000	
0.0001348	10000		0.005234	10000	
Binary			Quick		
Time [s]	**Size**	**Time avg**	**Time [s]**	**Size**	**Time avg**
0.001603	10000	0.001632	0.114422	10000	0.11447
0.0015837	10000		0.113562	10000	
0.0015829	10000		0.115388	10000	
0.0016728	10000		0.113979	10000	
0.0016632	10000		0.112885	10000	
0.0017048	10000		0.113043	10000	
0.0015997	10000		0.115133	10000	
0.001634	10000		0.114609	10000	
0.001639	10000		0.115049	10000	
0.0016354	10000		0.116629	10000	

Descending

Insertion			Heap		
Time [s]	Size	Time avg	Time [s]	Size	Time avg
0.363882	10000	0.365911	0.005146	10000	0.0050799
0.524178	10000		0.005039	10000	
0.336807	10000		0.005049	10000	
0.317178	10000		0.00522	10000	
0.315948	10000		0.00506	10000	
0.31827	10000		0.005031	10000	
0.431953	10000		0.005199	10000	
0.398552	10000		0.004821	10000	
0.33784	10000		0.0051	10000	
0.314498	10000		0.005133	10000	
Binary			Quick		
Time [s]	Size	Time avg	Time [s]	Size	Time avg
0.205154	10000	0.208787	0.002329	10000	0.0023545
0.207574	10000		0.002377	10000	
0.206741	10000		0.002309	10000	
0.207048	10000		0.002339	10000	
0.210196	10000		0.002349	10000	
0.209205	10000		0.002434	10000	
0.209707	10000		0.002379	10000	
0.209508	10000		0.002357	10000	
0.210874	10000		0.002326	10000	
0.211858	10000		0.002345	10000	

OneThird

Insertion			Heap		
Time [s]	Size	Time avg	Time [s]	Size	Time avg
0.2715	10000	0.3040187	0.004939	10000	0.005092
0.313385	10000		0.004986	10000	
0.282007	10000		0.005248	10000	
0.266819	10000		0.005198	10000	
0.268659	10000		0.005075	10000	
0.263491	10000		0.005216	10000	
0.339051	10000		0.00508	10000	
0.267725	10000		0.005046	10000	
0.377963	10000		0.005103	10000	
0.389587	10000		0.005031	10000	
Binary			Quick		
Time [s]	Size	Time avg	Time [s]	Size	Time avg
0.183547	10000	0.1854722	0.002573	10000	0.002566
0.185664	10000		0.002646	10000	
0.184813	10000		0.002502	10000	
0.184445	10000		0.002626	10000	
0.185521	10000		0.002661	10000	
0.189062	10000		0.002498	10000	
0.183451	10000		0.00253	10000	
0.186595	10000		0.002655	10000	
0.185782	10000		0.00245	10000	
0.185842	10000		0.002518	10000	

TwoThirds

Insertion			Heap		
Time [s]	Size	Time avg	Time [s]	Size	Time avg
0.192305	10000	0.199206	0.005116	10000	0.004996
0.207008	10000		0.005065	10000	
0.195326	10000		0.004724	10000	
0.194477	10000		0.005144	10000	
0.19349	10000		0.00473	10000	
0.196469	10000		0.005022	10000	
0.194195	10000		0.005116	10000	
0.195051	10000		0.005	10000	
0.197659	10000		0.005104	10000	
0.226084	10000		0.004942	10000	
Binary			Quick		
Time [s]	Size	Time avg	Time [s]	Size	Time avg
0.116676	10000	0.116128	0.002545	10000	0.002602
0.117381	10000		0.002601	10000	
0.114859	10000		0.002608	10000	
0.116221	10000		0.00257	10000	
0.116293	10000		0.002926	10000	
0.11504	10000		0.002594	10000	
0.115493	10000		0.002555	10000	
0.117184	10000		0.002581	10000	
0.116111	10000		0.002554	10000	
0.116023	10000		0.002488	10000	

Unsorted

Insertion			Heap		
Time [s]	Size	Time avg	Time [s]	Size	Time avg
0.307639	10000	0.31878	0.005129	10000	0.005144
0.30494	10000		0.00523	10000	
0.299133	10000		0.005045	10000	
0.300341	10000		0.005079	10000	
0.311116	10000		0.005224	10000	
0.299953	10000		0.005098	10000	
0.323005	10000		0.005141	10000	
0.383257	10000		0.005042	10000	
0.355667	10000		0.005381	10000	
0.302752	10000		0.005069	10000	
Binary			Quick		
Time [s]	Size	Time avg	Time [s]	Size	Time avg
0.213978	10000	0.208614	0.002298	10000	0.002355
0.212363	10000		0.002272	10000	
0.207482	10000		0.002229	10000	
0.213518	10000		0.002444	10000	
0.209702	10000		0.002428	10000	
0.206147	10000		0.002311	10000	
0.205995	10000		0.002302	10000	
0.206682	10000		0.002292	10000	
0.204614	10000		0.002619	10000	
0.205656	10000		0.002359	10000	

Wnioski dla części przypadków szczególnych

Ascending (Posortowane rosnąco):

- Średni czas sortowania wynosi około 0.00014 sekundy.
- Algorytm radzi sobie bardzo dobrze z już posortowanymi danymi, co potwierdza jego efektywność w przypadku, gdy dane są już w częściowo posortowanym stanie.

Descending (Posortowane malejąco):

- Średni czas sortowania jest znacznie dłuższy niż w przypadku danych posortowanych rosnąco, wynosząc około 0.366 sekundy.
- Algorytm Insertion Sort wykazuje słabą wydajność dla danych posortowanych w odwrotnej kolejności, mamy tutaj odczynienia z koniecznością przesunięcia każdej wartości aby posortować odwrotnie dlatego czas jest najwyższy.

OneThird (Jedna trzecia danych posortowana):

- Średni czas sortowania wynosi około 0.304 sekundy.
- Wynik jak najbardziej oczekiwany prędkość sortowania mniejsza niż przy tablicy nieposortowanej ale większa niż przy tablicy posortowanej w 2/3. Ponieważ część elementów już jest posortowana nie trzeba układać ich na miejsce.

TwoThirds (Dwie trzecie danych posortowana):

- Średni czas sortowania wynosi około 0.199 sekundy.
- Algorytm osiąga lepsze wyniki niż dla danych nie posortowanych i posortowanych w 1/3, ale gorsze niż dla danych posortowanych rosnąco, co wskazuje na korzystny wpływ częściowego posortowania danych. Natomiast reszta wciąż wymaga posortowania.

Unsorted (Nieposortowane):

- Średni czas sortowania wynosi około 0.319 sekundy.
- Prawie najwyższy czas osiągamy dla danych losowych, wymagają one sortowania ale w mniejszej ilości niż dane posortowane malejącą ze względu na losowość część danych już jest we właściwych miejscach albo prawie we właściwych.

Podsumowując: Wyniki jak najbardziej zgodne z oczekiwanymi

Wnioski i podsumowanie można wysunąć następujące:

- Quick Sort i Heap Sort są szybkie
- Insertion Sort i Binary Insertion Sort wolniejsze
- Dane całkowitoliczbowe sortują się najszybciej
- Dane znakowe najwolniej
- Im większa tablica, tym dłuższy czas sortowania
- Sortowanie danych posortowanych rosnąco jest najszybsze
- Sortowanie danych posortowanych malejąco najwolniejsze
- Częściowe posortowanie danych poprawia czas sortowania

Podsumowując, wybór odpowiedniego algorytmu sortowania powinien uwzględniać rodzaj danych, rozmiar tablicy oraz oczekiwaną wydajność, aby zoptymalizować czas wykonania operacji sortowania dla konkretnego przypadku.

Literatura

[1] Cormen T., Leiserson C.E., Rivest R.L., Stein C., Wprowadzenie do algorytmów, WNT

[2] Drozdek A., C++. Algorytmy i struktury danych, Helion

[3] https://learn.microsoft.com/pl-pl/cpp/cpp/templates-cpp?view=msvc-170

[4] https://learn.microsoft.com/pl-pl/cpp/c-runtime-library/reference/srand?view=msvc-170

https://learn.microsoft.com/pl-pl/cpp/c-runtime-library/reference/rand?view=msvc-170

[5] https://www.ibm.com/docs/pl/i/7.5?topic=functions-srand-set-seed-rand-function

[6] https://learn.microsoft.com/pl-pl/cpp/cpp/static-cast-operator?view=msvc-170

[7] https://en.cppreference.com/w/cpp/types/is_integral

[8] https://stackoverflow.com/questions/184710/what-is-the-difference-between-a-deep-copy-and-a-shallow-copy

[9] http://jaroslaw.mierzwa.staff.iiar.pwr.wroc.pl/

[10] Wykłady prof. dr hab. inż. Jan Magott

[11] https://en.wikipedia.org/wiki/Insertion_sort

[12] https://www.geeksforgeeks.org/binary-insertion-sort/

[13] https://pl.wikipedia.org/wiki/Sortowanie_przez_kopcowanie

[14] https://pl.wikipedia.org/wiki/Sortowanie_szybkie

[15] https://cpp0x.pl/forum/temat/?id=21331

[16] https://en.cppreference.com/w/cpp/chrono/high_resolution_clock

[17] https://en.cppreference.com/w/cpp/algorithm/sort

[18] https://stackoverflow.com/questions/17333/how-do-you-compare-float-and-double-while-accounting-for-precision-loss

[19] https://embeddeduse.com/2019/08/26/qt-compare-two-floats/

[20] https://en.cppreference.com/w/cpp/types/numeric_limits/epsilon

Pliki zawarte w pliki.zip