Raport - projekt 2

Mikołaj Strużykowski

10 maja 2024

1 Podsumowanie

1.1 Stopień realizacji zadania

Zadanie zostało wykonane w całości, zaimplementowany algorytmy sortujące to:

- Quicksort,
- Bucketsort,
- Introsort

1.2 Quicksort

Algorytm *quicksort* wykorzystuje Hoare partition scheme, ponieważ w podanym pliku testowym kolumna "rating"zawierała tylko 9 różnych kategorii i naiwna implementacja algorytmu powodowała wielokrotną rekurencję, która nieproporcjonalnie spowalniała program.

1.3 Bucketsort

Algorytm *bucketsort* dzieli dane na liczbę pojemników zależną od odległości pomiędzy minimalną, a maksymalną wartością, poszczególne wiaderka są sortowane za pomocą wcześniej zaimplementowanego *quicksort'a*.

1.4 Introsort

Algorytm *introsort* sprawdza maksymalną głębokość - zależną od ilości elementów, dla wartości maksymalnej głebokości równej 0 wykonywany jest *heapsort*. Próg poniżej, którego wykonywany jest introsort. Dla wartości powyżej progu dane sortowane są za pomocą *quicksort'a*.

1.5 Testowanie

Program był tesotwany na systemie Linux Ubuntu 22.04.3 oraz na procesorze Intel Core i5-8250U o zegarze 1.6 GHz, przy pomocy kompilatora gcc. Do obliczania czasu wykonania zadania została użyta funkcji składowej:

```
std::chrono::high_resolution_clock::now()
```

oraz

```
std::chrono::duration_cast<std::chrono::microseconds>
```

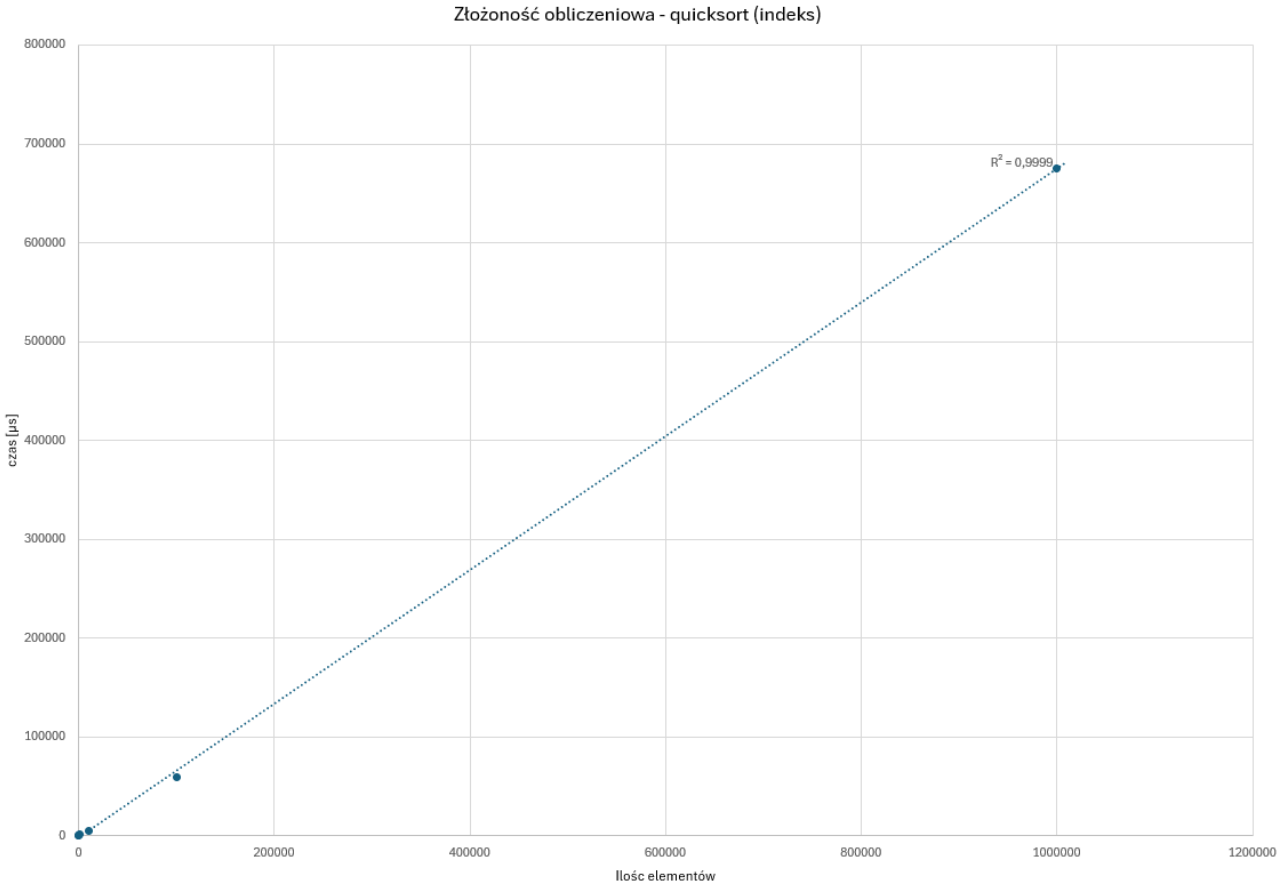
Program odfiltrowuje wszystkie wiersze, dla których nie udało się przekonwertować klucza na typ **int**.

2 Złożoność obliczeniowa

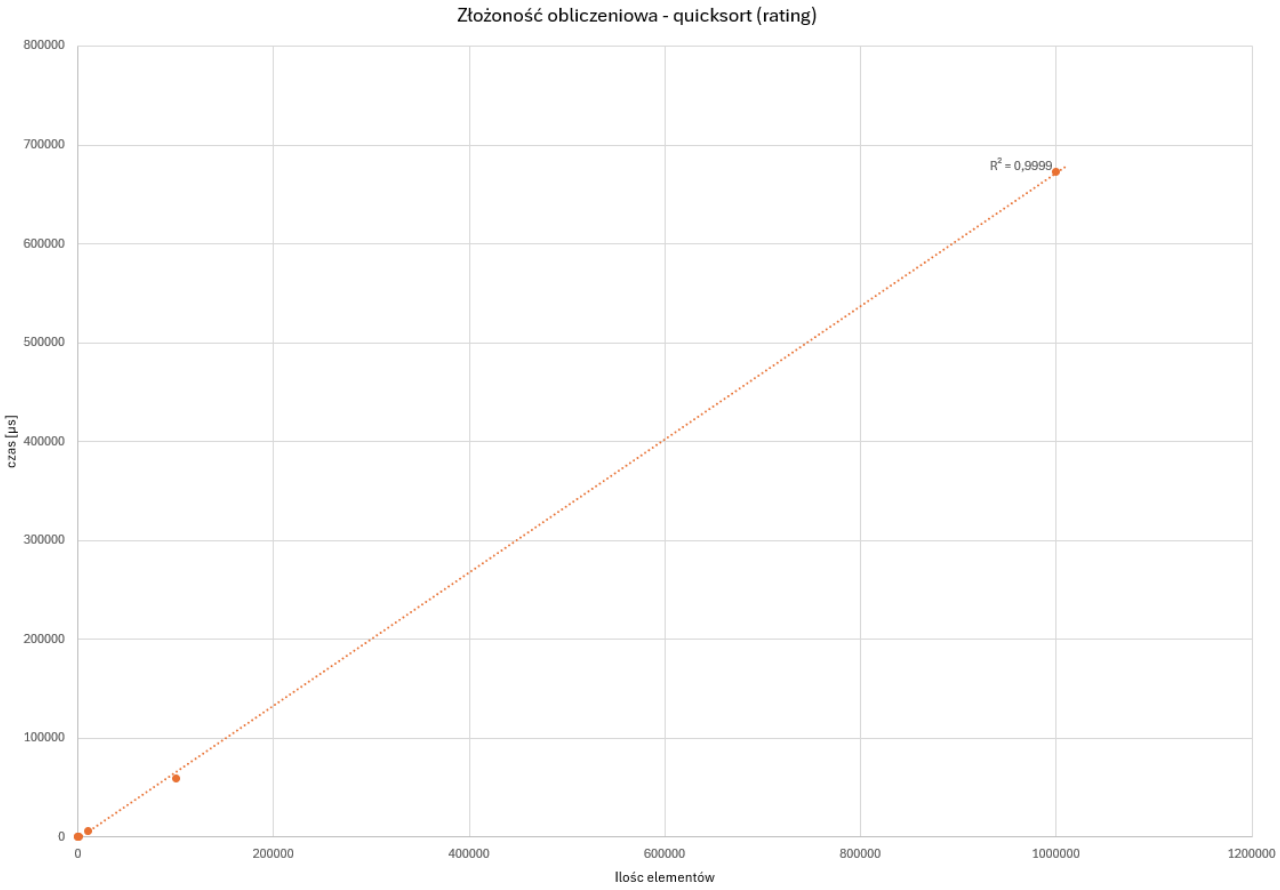
2.1 Quicksort

Wszystkie test zostały przeprowadzone dla **pliku_projekt2_dane_daszek.csv** oraz wczytane dane zostały raz przemieszane funkcją *shuffle()*.

ilość wierszy	czas sortowania - według indeksu [μs]	czas sortowania - według ratingu [μs]
10	21,6	10,4
100	205,6	203,2
1000	2175,6	1030,0
10000	5625,8	6157,6
100000	59948,8	59341,6
1000000	675481,4	672511,2



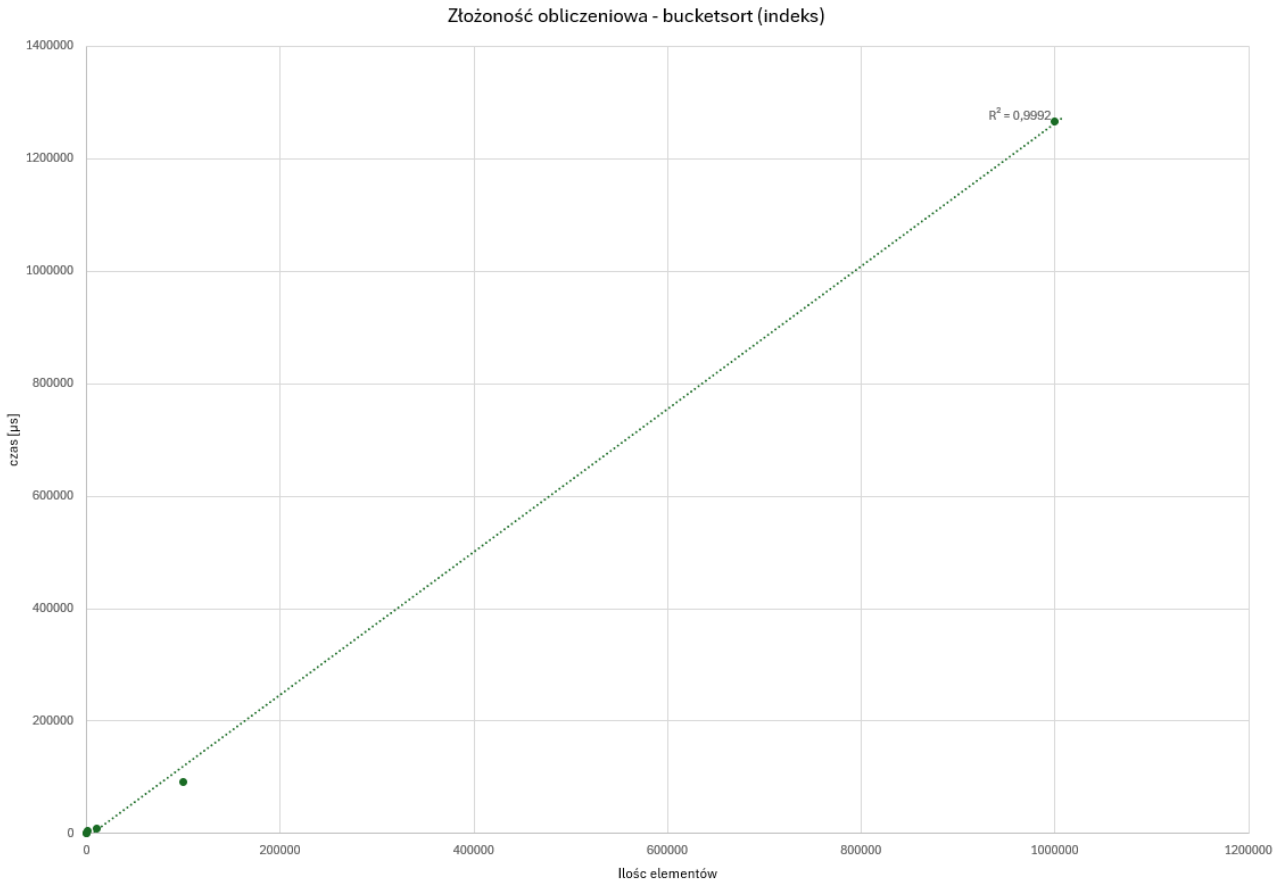
Rysunek 1: Wykres czasu wykonywania programu od ilości sortowanych elementów - kluczem jest indeks.



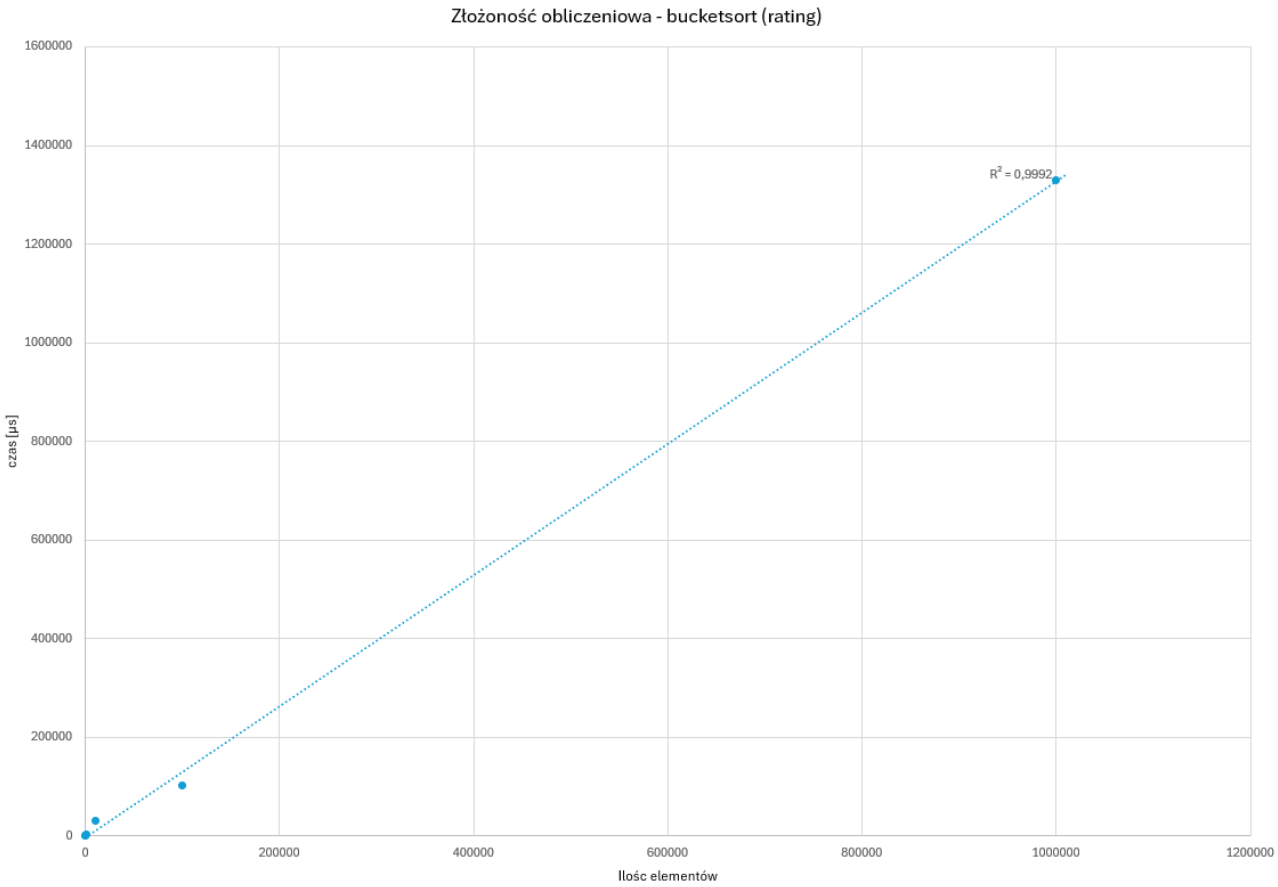
Rysunek 2: Wykres czasu wykonywania programu od ilości sortowanych elementów - kluczem jest rating.

2.2 Bucketsort

ilość wierszy	czas sortowania - według indeksu [μs]	czas sortowania - według ratingu [μs]
10	38,6	62,8
100	225,6	594,0
1000	4474,8	3826,4
10000	8902,8	30236,4
100000	91168,0	103090,6
1000000	1266714,6	1330388,2



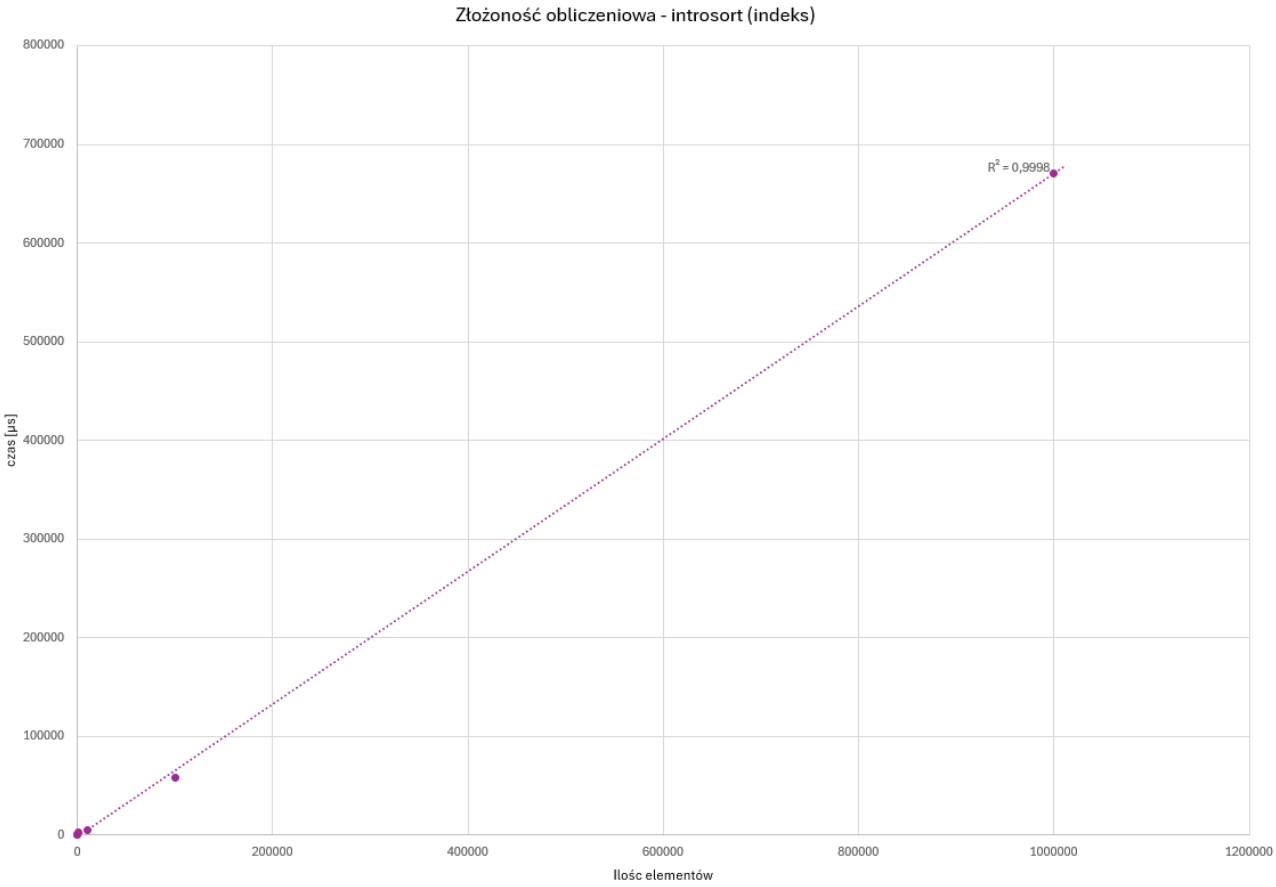
Rysunek 3: Wykres czasu wykonywania programu od ilości sortowanych elementów - kluczem jest rating.



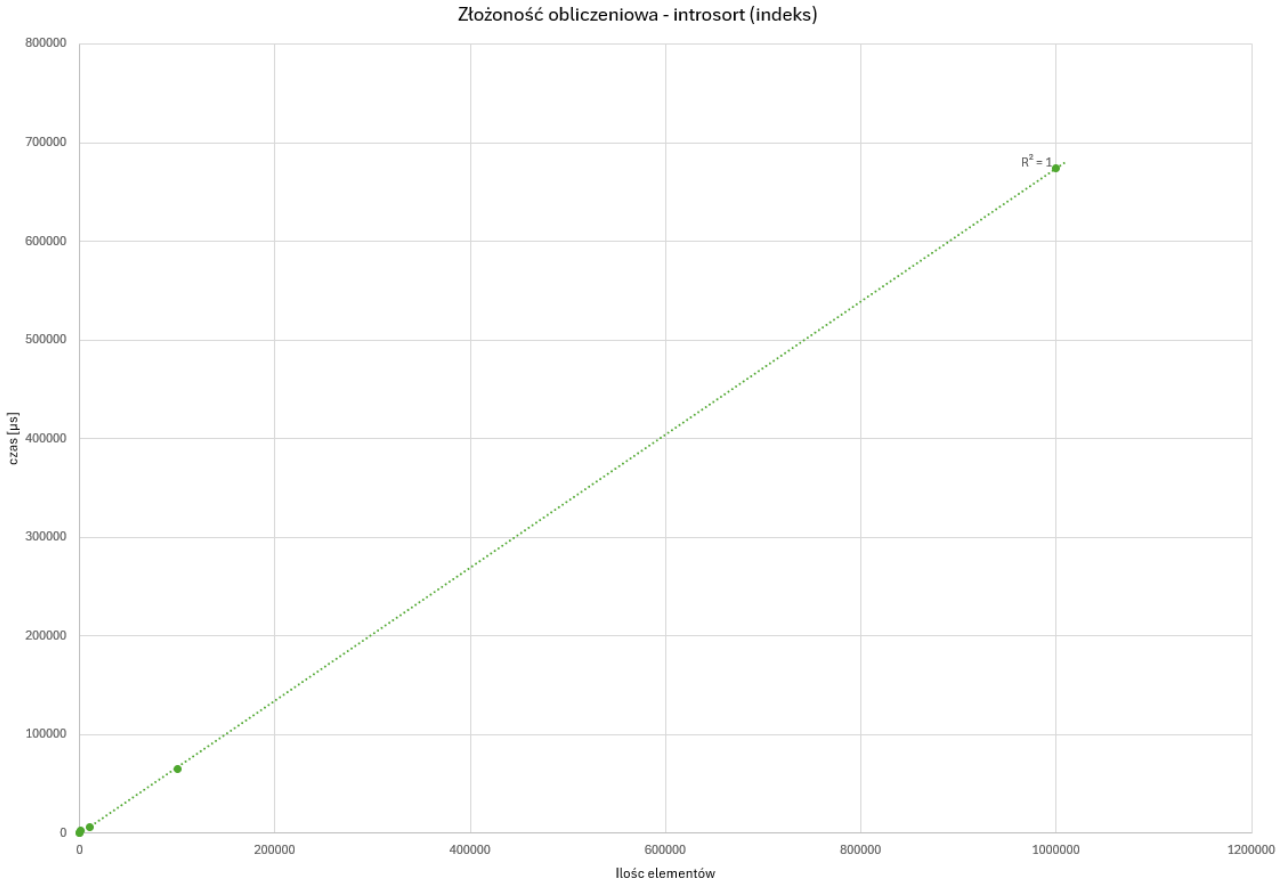
Rysunek 4: Wykres czasu wykonywania programu od ilości sortowanych elementów - kluczem jest rating.

2.3 Introsort

ilość wierszy	czas sortowania - według indeksu [μs]	czas sortowania - według ratingu [μs]
10	74,2	96,8
100	297,0	229,4
1000	2205,8	3323,0
10000	5473,6	5915,2
100000	58639,2	65407,8
1000000	670865,4	673889,8



Rysunek 5: Wykres czasu wykonywania programu od ilości sortowanych elementów - kluczem jest rating.



Rysunek 6: Wykres czasu wykonywania programu od ilości sortowanych elementów - kluczem jest rating.

3 Wnioski i komentarze

- Wszystkie czasy wykonania funkcji sortującej są średnią z pięciu wywołań programu z różnymi losowymi ziarnami do funkcji mieszającej.
- Dla wszystkich algorytmów złożoność obliczeniowa to $O(n)$ co jest przewidywaną wartością dla *quicksort'a*, lepszą dla *bucketsort'a* (co może być spowodowane tym, że poszczególne pudełka są sortowane za pomocą *quicksort*) i gorszą dla *introsort'a*.

4 Użyte materiały

- Quicksort - https://en.wikipedia.org/wiki/Quicksort,
- Bucketsort - https://en.wikipedia.org/wiki/Bucket_sort,
- Introsort - https://en.wikipedia.org/wiki/Introsort,
- Heapsort - https://en.wikipedia.org/wiki/Heapsort,
- Insertionsort - https://en.wikipedia.org/wiki/Insertion_sort,
- Pomiar czasu - https://stackoverflow.com/questions/2808398/easily-measure-elapsed-time