Projekt 1: Algorytmy Sortowania
Prowadzący: Mgr inż. Marta Emirsajłow
Zajęcia: Piątek 13:15

Jan Bronicki 249011

Marzec 2020

1 Opis

Projekt polega na stworzeniu trzech algorytmów i przetestowaniu ich efektywności czasowej. W projekcie analizowane będą następujące algorytmy:

- MergeSort (Przez scalanie)
- QuickSort
- IntroSort (Introspektywne)

Porcje danych (elementy typu całkowitoliczbowego) będą nie tylko różniły się objętością ale także swoim ułożeniem względem siebie, tzn. w niektórych przypadkach będą wstępnie posortowane do określonego poziomu. W projekcie będą rozważane następujące objętości danych:

- 10 000
- 50 000
- 100 000
- 500 000
- 1 000 000

Oraz następujące stopnie posortowania:

- Wszystkie elementy losowe
- 25%
- 50%
- 75%

- 95%
- 99%
- 99.7%
- Wszystkie elementy posortowane, ale w odwrotnej kolejności

2 Algorytmy

2.1 MergeSort (Przez Scalanie)

Algorytm stosujący metodę "divide and conquer". Działanie MergeSort'a można opisać w dwóch prostych krokach. Na początku dzieli rekurencyjnie całą tablice, aż do osiągnięcia jednoelementowych tablic, gdzie następnie scala je spowrotem odpowiednio je sortując, aż do uzyskania całego posortowanego zestawu.
Złożoność obliczeniowa MergeSort'a wygląda następująco:

- Najlepszy przypadek O(nlogn)
- Typowy przypadek O(nlogn)
- Najgorszy przypadek O(nlogn)

Złożoność pamięciowa:

- O(n)

MergeSort jest algorytmem stabilnym, co oznacza, że jeśli przed sortowaniem mamy elementy o takiej samej wartości, które są względem siebie ułożone w konkretnej kolejnościto po sortowaniu te elementy będą nadal w takiej samej kolejności.

Złożoność pamięciowa MergeSort'a wynika z potrzeby posiadania dodatkowej tymczasowej struktury danych. Jest to fakt który decyduje otym, że HeapSort(algorytm sortujący „w miejscu")jest preferowanym algorytmem nad MergeSort'em.

2.2 QuickSort

Algorytm, który również stosujący metodę divide and conquer. Działanie QuickSort'a można opisać w dwóch prostych krokach. Wybierz na zasadzie określonych zasad element rozdzielający (pivot), który rozdziela zestaw danych na dwie części. Ważne jest, że elementy większe od pivota przenoszone są do części po jego prawej stronie a mniejsze do części po lewej. Wykonuj rekursywnie powyższą czynność dla kolejnych fragmentów zestawu danych aż do osiągnięcia jednoelementowego zbioru który jest uważany za posortowany.

Złożoność obliczeniowaQuickSort'a wygląda następująco:

- Najlepszy przypadek O(nlogn)
- Typowy przypadek O(nlogn)
- Najgorszy przypadek O(n^2)

Złożoność pamięciowa:

- O(logn) — O(n)

QuickSort nie potrzebuje dodatkowej struktury danych żeby sortować, tzn. sortuje on „w miejscu". Złożoność pamięciowa O(n)wynika z występowania pesymistycznego przypadku QuickSorta. QuickSort jest niestabilny. Z uwagi na możliwość manipulacji wyboru pivota, algorytm ten może działać z bardzo rożną wydajnością.

2.3 Introsort (Introspektywne)

Algorytm ten jest hybrydowy, tzn. że składa się on z kilku innych algorytmów sortowania. Na IntroSort'a składają się: QuickSort, HeapSort, InsertionSort. Działanie IntroSort'a można opisać w kilku prostych krokach. Partycjonuj dane w taki sam sposób jak QuickSort, czyli wykorzystując element rozdzielający. Jeżeli głębokość rekurencji (dozwolona głębokość wywołań rekurencyjnych określana jako współczynnik $2 \cdot \log n$) jest równa 0 przełącz się na HeapSort'a. Jeżeli pozostała ilość danych w określonym fragmencie jest mniejsza od 16 to należy przełączyć się na InsertionSort'a i posortować fragment do końca.
Złożoność obliczeniowa IntroSort'a wygląda następująco:

- Najlepszy przypadek O(nlogn)
- Typowy przypadek O(nlogn)
- Najgorszy przypadek O(nlogn)

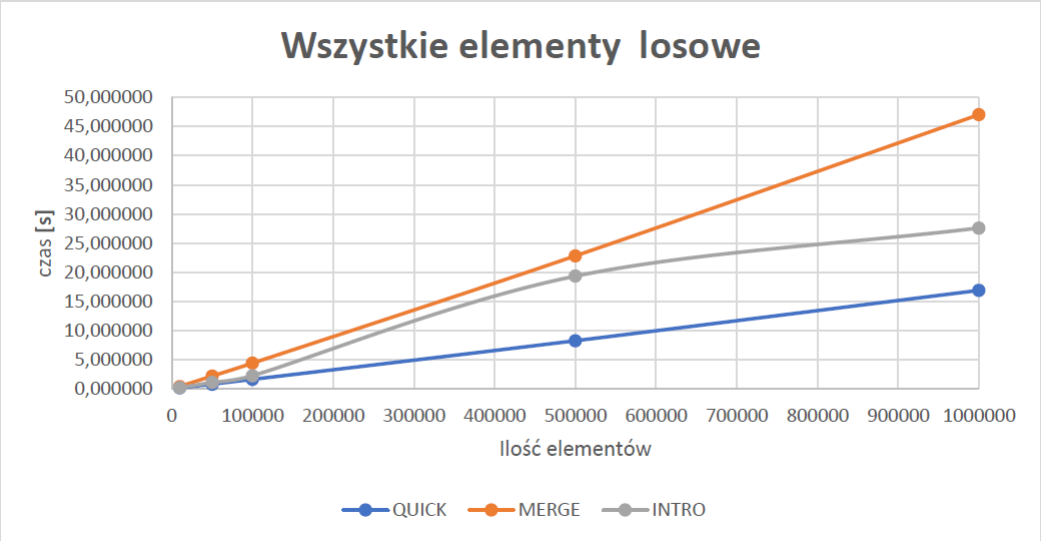
Złożoność pamięciowa:

- O(logn)

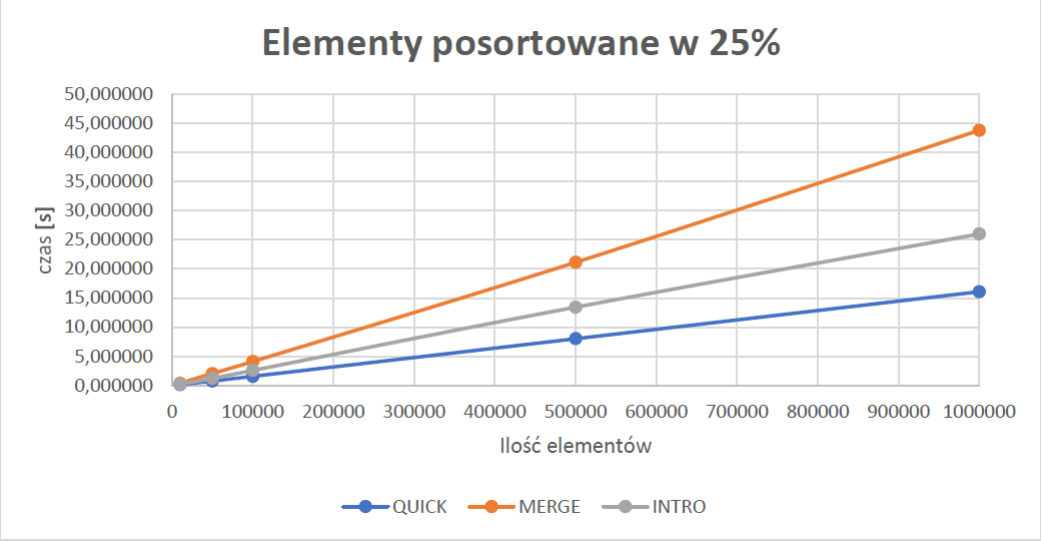
IntroSort również nie potrzebuje dodatkowej struktury danych żeby sortować oraz jest niestabilny. Algorytm ten został stworzony aby wyeliminować najgorszy przypadek złożoności obliczeniowej QuickSort'a. Warto wspomnieć że jest to algorytm którego używa język programowania C++ pod instrukcją std::sort()w nagłówku ¡algorithm¿.

3 Przebieg eksperymentów

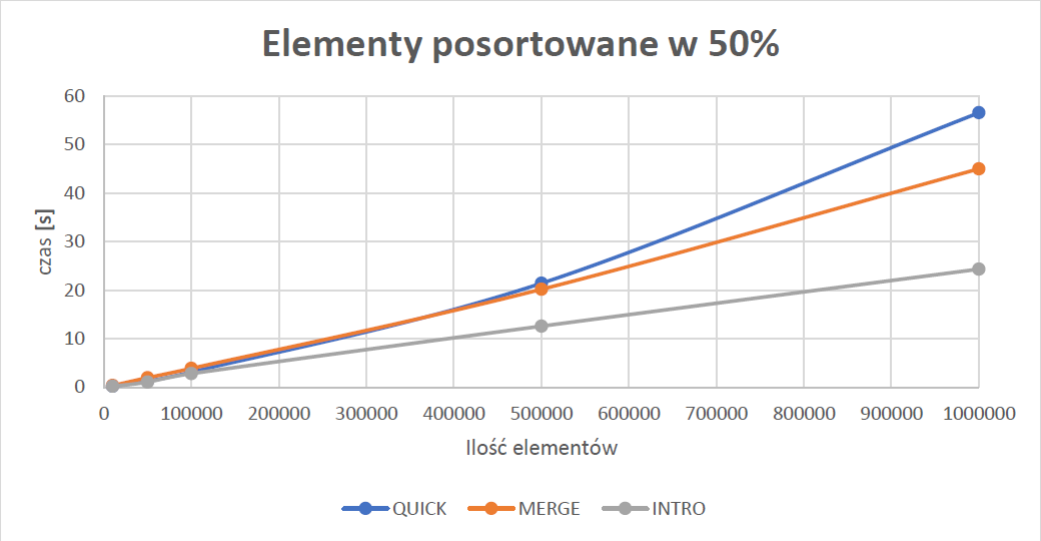
3.1 Wszystkie elementy tablicy losowe



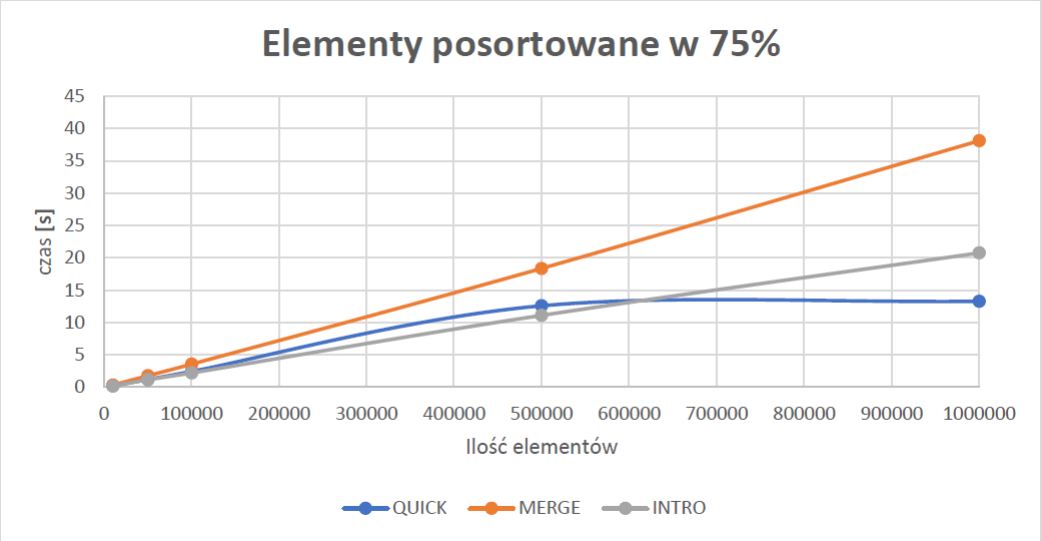
3.2 25% elementów tablicy losowe



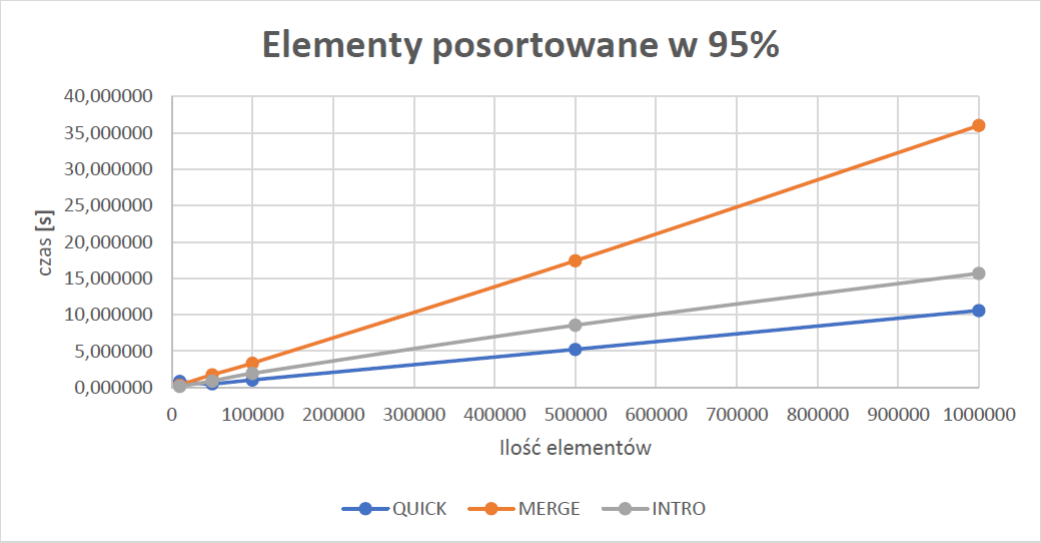
3.3 50% elementów tablicy losowe



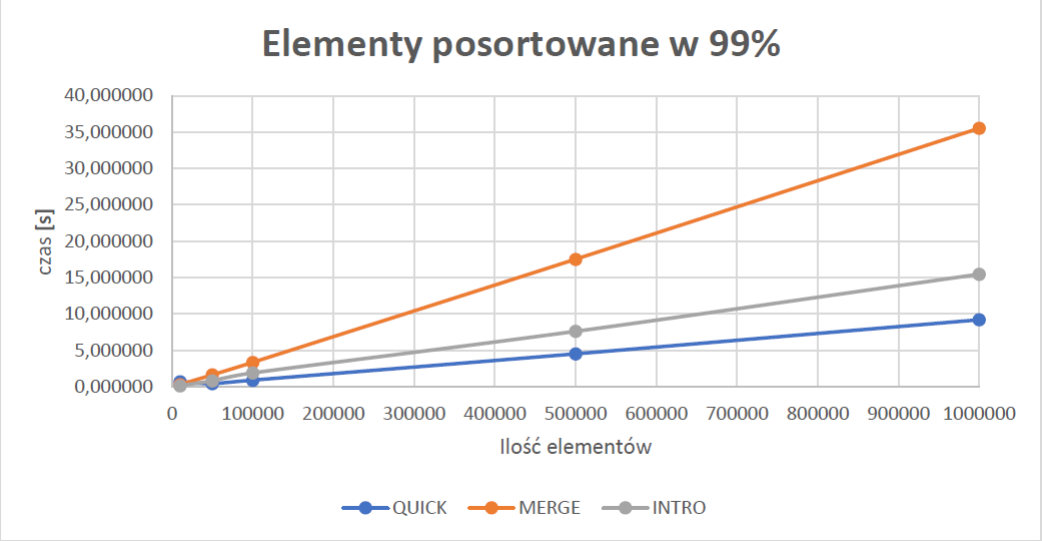
3.4 75% elementów tablicy losowe



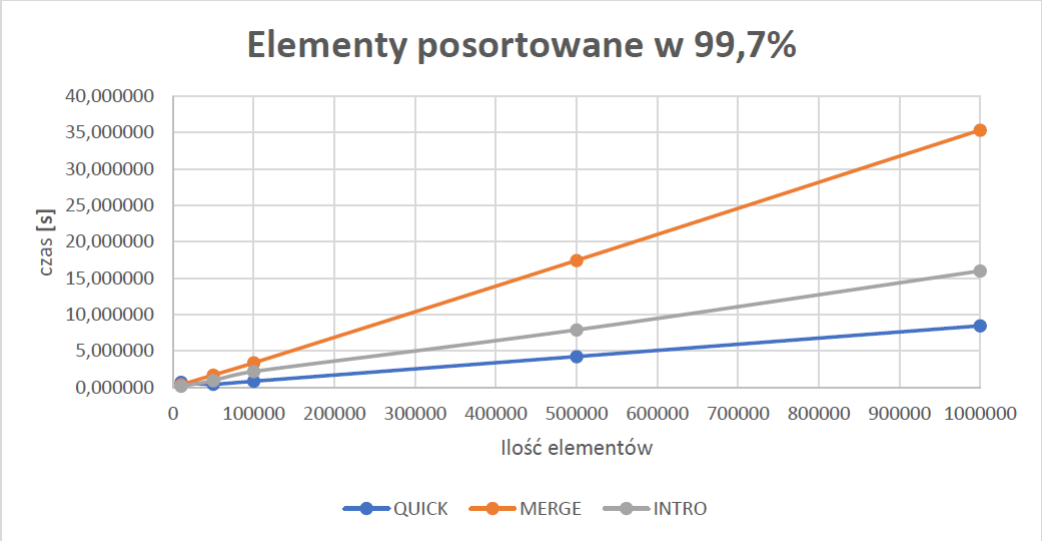
3.5 95% elementów tablicy losowe



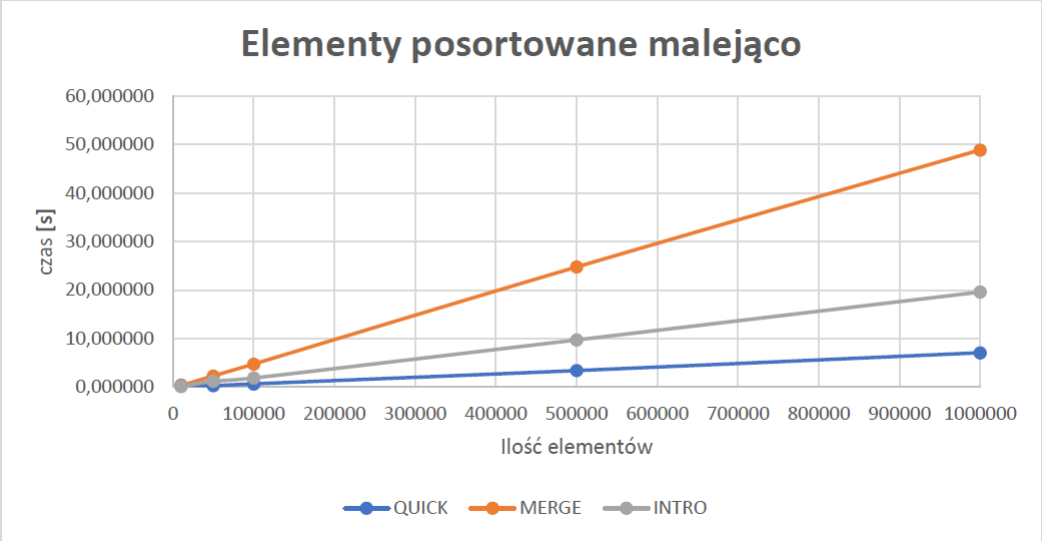
3.6 99% elementów tablicy losowe



3.7 99.7% elementów tablicy losowe



3.8 Wszystkie elementy tablicy posortowane, ale w odwrotnej kolejności



4 Podsumowanie

Podczas wstępnego, połowicznego posortowania tablicy (50%) można zauważyć odstępstwa od schematyczności pomiarów prędkości sortowania algorytmów. Jak możemy zauważyć QuickSort wydłużył czas swojej pracy niemal o 400%. Wynika to z jego implementacji a konkretnie dobierania pivot'a jako środkowego elementu danych. To właśnie spowodowało tak rozciągnięcie czasu pracy. Zgodnie z założeniami najwolniejszym algorytmem sortowania okazał się MergeSort. Niestety w przeciwieństwie do założeń najszybszym algorytmem okazał się QuickSort (a nie IntroSort) oprócz jednego przypadku. IntroSort powinien być najszybszym algorytmem ze względu na swoją hybrydowość. Wyniki pomiarów jednak tego nie pokazują. Najprawdopodobniej wynika to z możliwej słabej implementacji tego algorytmu. Wstępne posortowanie części danych znacznie skraca czas sortowania dla każdego algorytmu. Dokładnie tak samo algorytmy zachowują się dla tablicy posortowanej malejąco. Wyniki eksperymentów w ogólnym obrazie pokazują poprawność założeń, aczkolwiek implementacja algorytmów, aby móc użyć ich pełnego potencjału pozostawia sobie wiele do życzenia.

Źródła:

Wikipedia „Sortowanie szybkie"-
https://pl.wikipedia.org/wiki/Sortowanie_szybkie

$Wikipedia „Sortowanie przez scalanie" -$
$https://pl.wikipedia.org/wiki/Sortowanie_przez_scalanie$

$Wikipedia „Sortowanie introspektywne" -$
$https://pl.wikipedia.org/wiki/Sortowanie_introspektywne$