|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Języki Asemblerowe - projekt | | | | | | |
| Rok akademicki | Termin | Rodzaj studiów | Kierunek | Prowadzący | Grupa | Sekcja |
| 2014/2015 | poniedziałek | Dzienne | INF | JP | 3 | 6 |
| 14:45 – 16:15 |

Jakub Bujny

**Wielowątkowe sortowanie bloków danych za pomocą algorytmów quick sort, insert sort oraz bubble sort**

Plan:

1. Opis wstępny (założenia, co ma robić, analiza itp.)
2. Implementacja
3. Interfejs użytkownika (instrukcja obsługi itp.)
4. Analiza wydajnościowa
5. Wnioski

# Opis wstępny

## Założenia projektu

**Dane wejściowe:**

* ***n nieuporządkowanych* multizbiorów (bloków) liczb całkowitych o rozmiarze *m g*dzie m,n N/{0}, zapisanych w pliku tekstowym**
* **liczba wątków**
* **funkcja sortująca**
* **typ funkcji sortującej (wysokopoziomowa lub asm)**

**Dane wyjściowe:**

* **n posortowanych multizbiorów(bloków) liczb całkowitych o rozmiarze m *g*dzie m,n N/{0}, zapisanych w pliku tekstowym**
* **czas wykonania programu**

**Wykonanie programu:**

**Program pobiera od użytkownika nazwę pliku z danymi wejściowymi, nazwę pliku do którego należy zapisać dane wyjściowe, jedną z funkcji sortujących (quick sort, bubble sort, insert sort), liczbę wątków *k* oraz bibliotekę (asm lub wysokopoziomowa). Następnie określona zostaje ilość bloków wejściowych oraz ich rozmiar. Program wczytuje dane wejściowe do pamięci a następnie tworzy *k* wątków, które wykonują sortowanie. Jeden wątek sortuje dane wewnątrz jednego bloku. Gdy wszystkie dane zostaną przetworzone następuje zapis do pliku oraz zwrócenie czasu wykonania programu.**

## Założona praca programu

Zapis danych wynikowych do pliku oraz wyświetlenie czasu wykonania

Wykonywanie obliczeń w wątkach i pomiar czasu wykonania

Dane poprawne

Dane niepoprawne

Walidacja danych

Wybór pliku z danymi wejściowymi

Wybór liczby wątków, funkcji sortującej, typu biblioteki

## Metodyka działania

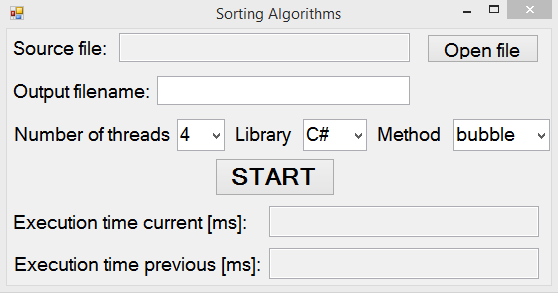
W czasie realizacji projektu przyjąłem następującą strategie działania:

1. Odnalazłem koncepcje algorytmu zapisaną w pseudokodzie
2. Zaimplementowałem algorytm w języku C#
3. Przetestowałem implementację porównując wyniki z funkcją Array.Sort
4. Na podstawie implementacji w C# wykonałem implementację w Asemblerze
5. Przetestowałem implementację porównując wyniki z funkcją Array.Sort
6. Wykonałem testy wydajnościowe

# Interfejs użytkownika

## GUI

**1**



**5**

**8**

**7**

**6**

**4**

**3**

**2**

**9**

1. Nazwa pliku wejściowego. Pole nieaktywne, wartość ustalana automatycznie po wskazaniu pliku przez użytkownika.
2. Przycisk umożliwiający użytkownikowi wskazanie pliku z danymi wejściowymi
3. Nazwa pliku wyjściowego ustalana przez użytkownika. Plik wyjściowy zapisywany jest w tym samym katalogu co plik .exe.
4. Wybór jednej z trzech metod sortujących
5. Wybór liczby wątków biorących udział w sortowaniu
6. Wybór biblioteki z funkcjami sortującymi
7. Czas wykonania dla aktualnie wprowadzonych parametrów
8. Czas wykonania poprzednich obliczeń (dla pierwszego uruchomienia przyjmuje wartość -1)
9. Przycisk rozpoczynający obliczenia

## Format danych wejściowych

Dane używają separatora w postaci znaku ; (średnik). Bloki separowane są za pomocą znaku nowej linii. Przykład:

1;2;3;4;5

1;2;3;4;5

5;4;3;2;1

# Testy wydajnościowe

Testy zostały przeprowadzone na laptopie posiadającym:

1. 6GB pamięci DRAM
2. Procesor Intel Core i3-3120M 2.5GHz, który posiada 2 rdzenie fizyczne logicznie widziane jako 4 rdzenie.

## Test 1

Dane testowe:

Odwrotnie posortowana tablica (problem z doborem danych)

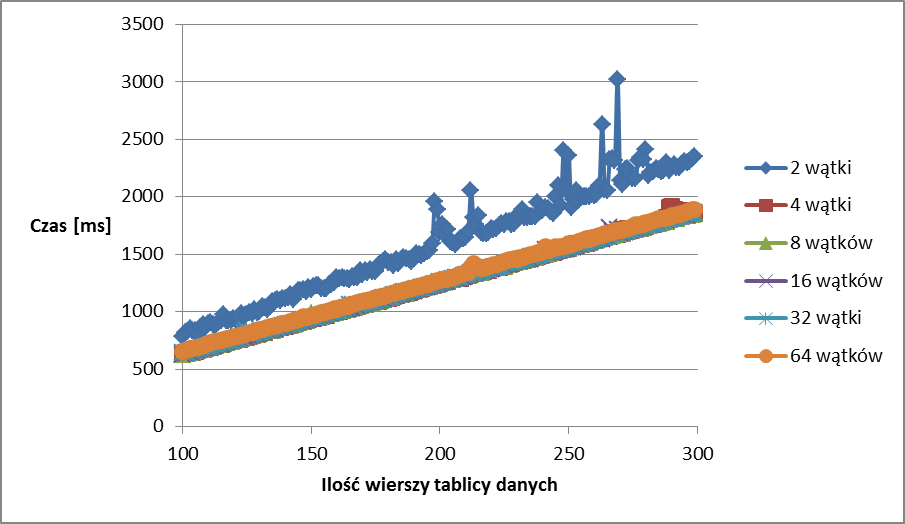
Warunki testowe:

Zmienna liczba wierszy, zmienna liczba wątków, stały rozmiar bloku (2 tyś. elementów)

**Bubble sort - ASM**



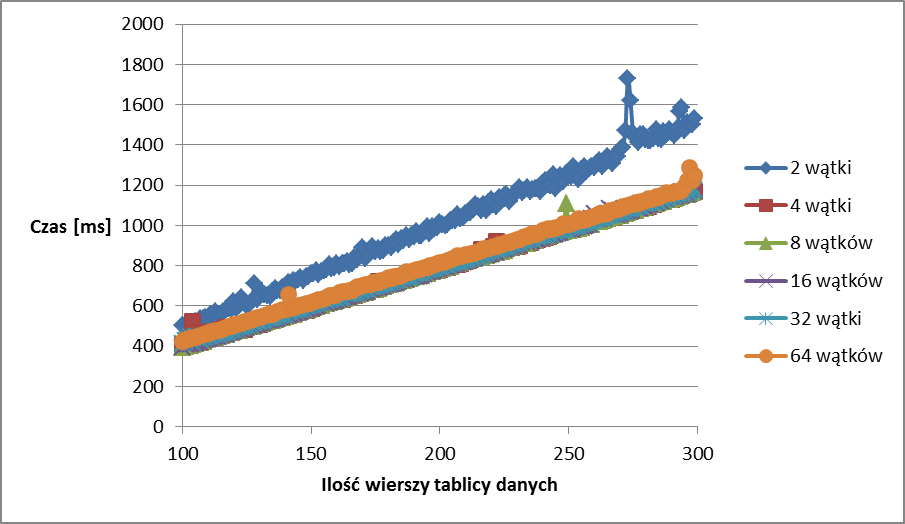
**Bubble sort – C#**



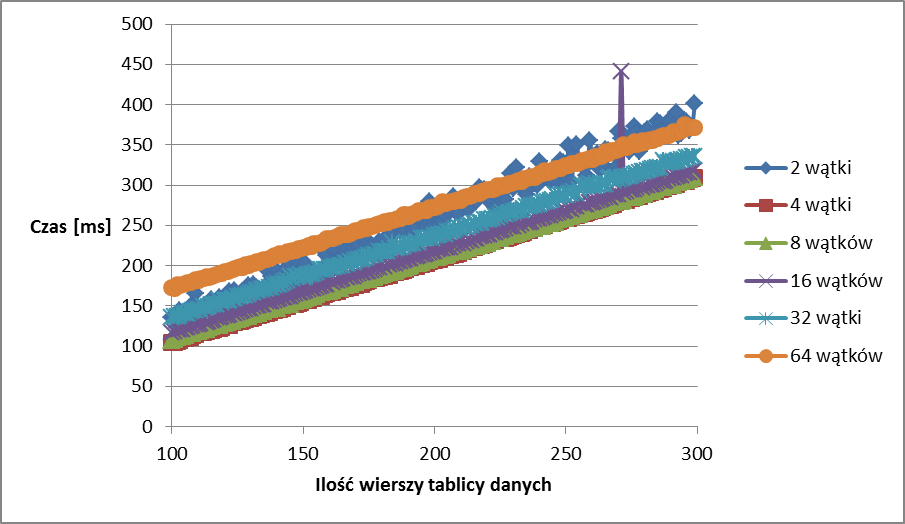
**Insert sort – ASM**

****

**Insert sort – C#**

****

**Quick sort – ASM**



**Quick sort – C#**

****

## Test 2

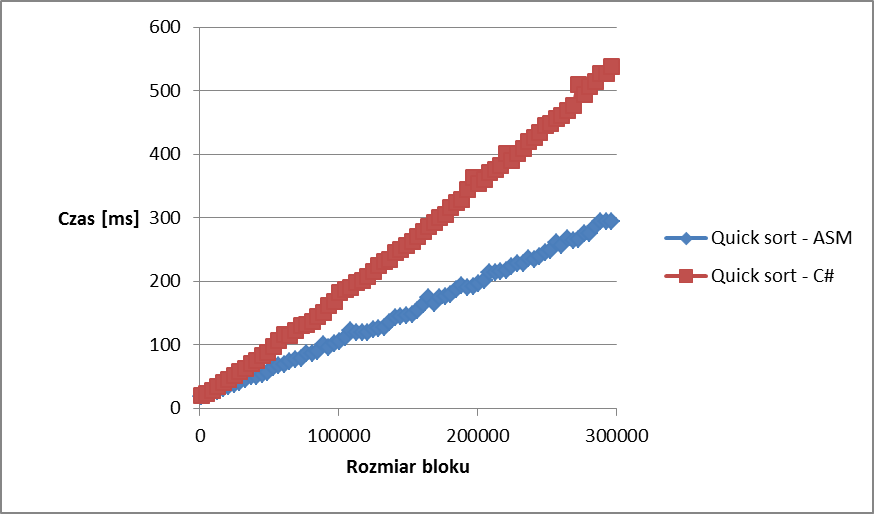
Dane testowe:

Tablica z liczbami pseudolosowymi

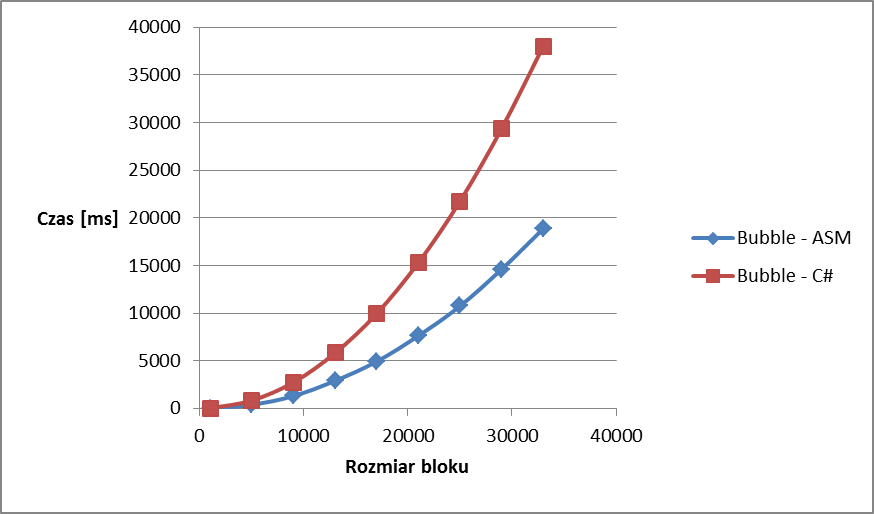
Warunki testowe:

Stała liczba wierszy (20), stała liczba wątków (4), zmienny rozmiar bloku

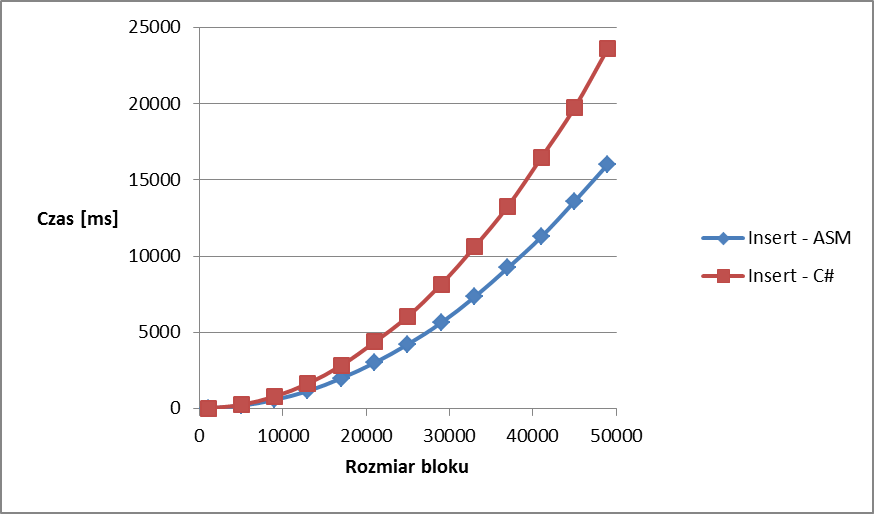
**Quick sort**



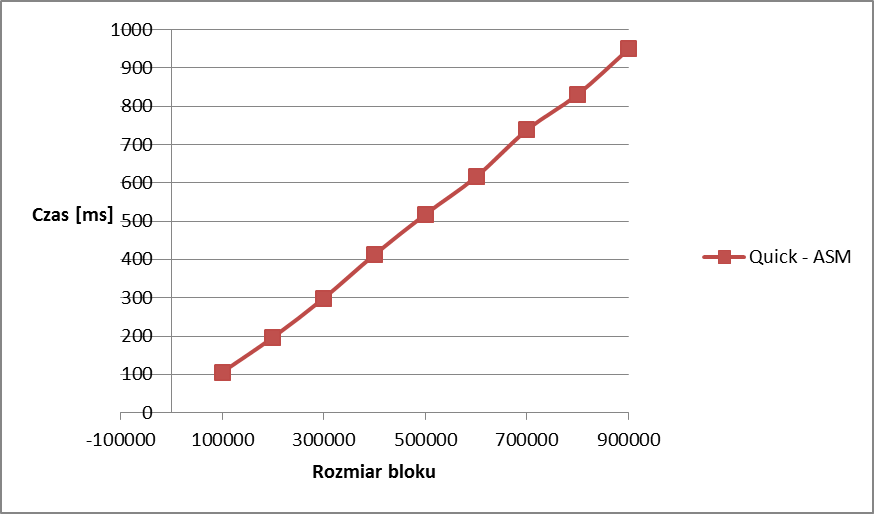
**Bubble sort**



**Insert sort**



**Quick sort – duży zbiór danych**



## Zestawienie wybranych wyników

|  |  |
| --- | --- |
| **Test 1 – średni czas wykonania dla 4 wątków** | |
| Implementacja | Czas [ms] |
| Bubble C# | 1246 |
| Bubble ASM | 533 |
| Insert C# | 786 |
| Insert ASM | 531 |
| Quick C# | 569 |
| Quick ASM | 206 |

|  |  |
| --- | --- |
| **Test 2 – średnia ilość sortowanych danych na ms przy 4 wątkach** | |
| Implementacja | Ilość danych na [ms] |
| Bubble C# | 24 |
| Bubble ASM | 49 |
| Insert C# | 60 |
| Insert ASM | 87 |
| Quick C# | **11243** |
| Quick ASM | **19579** |

# Wnioski