# Wstęp do wysokowydajnych komputerów - Laboratorium

## Raport – lab 2

Prowadzący: mgr Przemysław Świercz

Aleksander Mliczek 281139

Zaimplementowano funkcje quicksort w języku C w pliku quicksort.c

Kod implementuje algorytm QuickSort, wybierając pivot ze środka tablicy i dokonując partycjonowania przez zamianę elementów. Po podziale na dwie części rekurencyjnie sortuje każdą z nich.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

Kod w asemblerze wykonuje sortowanie tablicy przy użyciu algorytmu QuickSort i mierzy czas jego wykonania w cyklach CPU.Obraz zawierający tekst, zrzut ekranu

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

**Działanie:**

1. **Rezerwacja pamięci** – zapisuje bieżący stan stosu.
2. **Pomiar czasu przed sortowaniem** – odczytuje znacznik czasu (TSC).
3. **Wywołanie funkcji quicksort** – sortuje tablicę array.
4. **Pomiar czasu po sortowaniu** – ponownie odczytuje TSC.
5. **Obliczenie różnicy** – odejmuje wcześniejszy pomiar, uzyskując liczbę cykli CPU.
6. **Wyświetlenie wyniku** – używa printf do pokazania czasu sortowania.
7. **Zakończenie programu** – zwraca 0 jako kod wyjścia.

# Badania

W zależności od zawartości tablicy Quicksort zajmuje różną ilość cykli procesora.

Dla tablicy:

- 8 elementowej [2, 5, 8, 10, 14, 5, 3, 7] jest to średnio 10260 cykli CPU.

- 10 elementowej posortowanej [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] jest to średnio 8420 cykli CPU.

- 10 elementowa posortowana malejąco [10, 9, 8, 7, 6, 5, 4, 3, 2, 1] jest to średnio 10700 cykli CPU

- 10 elementowa w połowie posortowana [1, 2, 3, 4, 5, 12, 8, 14, 7, 10] jest to średnio 8990 cykli CPU

- 10 elementowa z powtarzającymi się wartościami [5, 3, 5, 7, 5, 8, 5, 2, 5, 9] jest to średnio 11000 cykli CPU, raz na parę prób zdarza się wartość rzędu 20000.

- 10 elementowa z małą liczbą unikalnych wartości [1, 2, 2, 1, 3, 3, 1, 2, 3, 1] jest to średnio 10200 cykli CPU

- 100 elementowa tablica losowa

[97, 3, 45, 78, 23, 11, 56, 90, 31, 67, 88, 41, 15, 74, 50, 99, 10, 6, 21, 83, 46, 59, 34, 72, 18, 62, 26, 95, 89, 14, 32, 80, 53, 100, 24, 64, 75, 92, 55, 27, 1, 48, 39, 35, 17, 28, 9, 79, 37, 30, 77, 7, 47, 86, 42, 5, 20, 96, 8, 16, 85, 43, 33, 51, 93, 44, 71, 29, 54, 2, 66, 58, 98, 19, 4, 13, 12, 87, 36, 60, 81, 22, 70, 91, 68, 52, 73, 82, 84, 61, 57, 76, 38, 40, 25, 94, 49, 63, 65, 69]

Jest to średnio 10080 cykli CPU.

- Losowa tablica 10 000 elementów – jest to średnio 8800 cykli CPU.

# Wnioski

1. **Tablica posortowana rosnąco** – Quicksort działa najlepiej (najmniej cykli CPU), ponieważ jego podział jest najbardziej zrównoważony, minimalizując liczbę porównań.
2. **Tablica posortowana malejąco** – Najgorsza wydajność (więcej cykli CPU), ponieważ Quicksort wykonuje maksymalną liczbę porównań i podziałów w każdej iteracji.
3. **Tablica z duplikatami** – Wydajność spada, zwłaszcza przy dużych duplikatach, ponieważ algorytm napotyka powtórzone wartości, co skutkuje mniej efektywnymi podziałami.
4. **Tablica częściowo posortowana** – Lepsza niż odwrotna, ale gorsza od posortowanej rosnąco, ponieważ pewne elementy są już na swoich miejscach, ale algorytm wciąż wykonuje zbędne operacje.
5. **Tablica losowa (100 elementów)** – Wydajność średnia, podobna do większych tablic, ponieważ algorytm działa z przeciętną liczba porównań i podziałów.
6. **Tablica losowa (10 000 elementów)** – Quicksort dobrze skalowalny przy dużych danych, nieznaczny wzrost cykli CPU, co pokazuje, że algorytm wciąż działa efektywnie, mimo wzrostu rozmiaru tablicy.

**Przyczyna**: Quicksort osiąga najlepsze wyniki na tablicach posortowanych rosnąco, ponieważ wykonuje efektywne podziały. Na tablicach odwrotnie posortowanych i z dużymi duplikatami podziały są mniej efektywne, co prowadzi do większej liczby operacji.

**Wpływ wyboru pivota**: W przypadku tablicy posortowanej lub odwrotnie posortowanej, wybór pierwszego elementu jako pivota prowadzi do najgorszego przypadku (O(n²)), podczas gdy losowy wybór pivota oraz mediana z trzech elementów prowadzą do zbliżonego wyniku O(n log n).

**Ogólne wnioski**: Nasze wyniki potwierdzają teoretyczną złożoność algorytmu Quicksort: dla tablic losowych (przeciętnych) czas wykonania jest bliski O(n log n), podczas gdy w najgorszym przypadku (tablica posortowana) czas wykonania może wynieść O(n²).