Sprawozdanie nr 1. Algorytmy sortowania

Algorytmy i Struktury Danych. Prowadzący Dominik Witczak.

Przygotowali:	Jan Czyżewski 160377	Albert Łapa 160381
Grupa LAB 13	Kierunek Informatyka	Zajęcia pt. 8:00- 9:30

Poniższe sprawozdanie przedstawia efektywność następujących algorytmów sortujących:

- Insertion Sort
- Selection Sort
- Shell Sort
- Quick Sort:
  - Pivot skrajnie lewy
  - Pivot na pozycji losowej
- Heap Sort

Efektywność algorytmów sprawdzono przy pomocy dostarczonych plików typu benchmark.

Komputer testujący wyposażony był w jednostkę obliczeniową AMD Ryzen 7 5800h i 24 GB pamięci operacyjnej. Pracował pod kontrolą systemu operacyjnego Windows 11 22H2.
Testy wykonano na algorytmach napisanych w języku Python w wersji 3.12.2.

Sortowanie przez wstawianie – algorytm Insertion Sort

Algorytm ten swoim działaniem przypomina sortowanie talii kart. Elementy posortowane w modyfikowanej tablicy przenoszone są na jej prawą stronę, zaś nieposortowane zostają po lewej. Czas działania algorytmu uzależniony jest od ustawienia elementów w tablicy. Oznacza to, że dla każdego zbioru czas ten będzie inny, ponieważ im więcej operacji (pętli) trzeba wykonać aby wykonać sortowanie, tym czas będzie dłuższy.

```python
def insertionSort(tab):
    for i in range(1, len(tab)):
        tempNumber = tab[i]
        x = i - 1

        while(x >= 0 and tempNumber < tab[x]):
            tab[x+1] = tab[x]
            x -= 1
        tab[x + 1] = tempNumber
    return tab
```

Kod algorytmu 1

Złożoność obliczeniowa algorytmu	Przypadek pesymistyczny	Przypadek średni	Przypadek optymistyczny
	$O(n^2)$	$O(n^2)$	$O(n)$

Sortowanie przez wybór – algorytm Selection Sort

Sortowanie przez wybór odbywa się podobnie jak sortowanie przez wstawianie. Elementy uporządkowane również trafiają na lewą stronę tablicy. Jednak różnią się one czasem działania programu. Złożoność obliczeniowa algorytmu Selection Sort jest stała!

```python
def selectionSort(tab):
    for i in range (len(tab)):
        for j in range(i, len(tab)):
            temp = tab.index(min(tab[i:len(tab)]))
            tab[i], tab[temp] = tab[temp], tab[i]
            break
    return tab
```

Kod algorytmu 2

Złożoność obliczeniowa algorytmu	Dane wejściowe nie mają znaczenia $O(n^2)$

Sortowanie Shella – algorytm Shell Sort

Sortowanie Shella wykorzystuje algorytm Insertion Sort, jednak robi to na mniejszych podzbiorach. Zbiór główny zostaje podzielony na podzbiory według wcześniej ustalonej wartości k. Jeżeli k=5, to tablica zostanie podzielona na podzbiory zawierające co 5-ty jej element. Podzbiory te są następnie sortowane poprzez wstawianie. Po zakończeniu pierwszego sortowania zmniejszamy k według ustalonego wcześniej wzoru, np. k = k/2. Operacje te są powtarzane do momentu, gdy k=1. Wtedy cały, posortowany ciąg należeć będzie do jednego zbioru.

```python
def sadgewickShellSort(arr):
    gaps = [1]
    k = 1
    while True:
        gap = 4**k + 3 * 2**(k-1) + 1
        if gap > len(arr):
            break
        if gap < len(arr):
            gaps.append(gap)
        k += 1
    gaps.sort()
    print(gaps)

    for gap in reversed(gaps):
        for i in range(gap, len(arr)):
            temp = arr[i]
            j = i
            while j >= gap and arr[j - gap] > temp:
                arr[j] = arr[j - gap]
                j -= gap
            arr[j] = temp

    return arr
```

Kod algorytmu 3

Złożoność obliczeniowa algorytmu	Przypadek pesymistyczny $O(n \cdot log(n))$	Przypadek średni $O(n^{1,25})$	Przypadek optymistyczny $O(n^2)$

Sortowanie szybkie – algorytm Quick Sort

Algorytm ten działa na zasadzie „dziel i zwyciężaj”. Tablica wejściowa zostaje podzielona na dwa podzbiory takie, że każdy element pierwszego podzbioru jest nie większy niż każdy element podzbioru drugiego. Następnie obie tablice zostają posortowane. Podtablice zawierające podzbiory sortowane są w miejscu, zatem ich łączenie nie jest konieczne. Należą już one do tablicy docelowej.

Przy danych zwiększających się lub stałych o rozmiarze większym od 2^8 występuje limit rekurencji. Zwiększenie tego limitu nie działa. Zamiast tego został użyty algorytm wykorzystujący stack.

Wartość według której tablica wejściowa zostaje podzielona (wartość największa podtablicy pierwszej) to pivot. Może nim być:

- środkowy element ciągu
- pierwszy element ciągu (skrajnie lewy)
- ostatni element ciągu (skrajnie prawy)
- losowy element ciągu
- mediana z 3 elementów ciągu (np. pierwszego, środkowego i ostatniego)
- element wybrany według jakiegoś innego schematu dostosowanego do zbioru danych.

Złożoność obliczeniowa algorytmu	Przypadek pesymistyczny	Przypadek średni	Przypadek optymistyczny
	$O(n \cdot log(n))$	$O(n \cdot log(n))$	$O(n^2)$

Sortowanie stogowe (przez kopcowanie) – algorytm Heap Sort

Sortowanie stogowe wykorzystuje właściwości struktury danych jaką jest kopiec. Pierwszym krokiem sortowania jest ułożenie elementów tablicy wejściowej tak, aby spełniała właściwości kopca. Sprawi to, że na jej pierwszym miejscu będzie największy element zbioru. Przenosimy go na ostatnią pozycję tablicy. Czynności te powtarzane są do momentu uporządkowania zbioru.

```python
def heapSort(arr):
    n = len(arr)
    for i in range(n // 2 - 1, -1, -1):
        heapify(arr, n, i)
    for i in range(n - 1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i]
        heapify(arr, i, 0)
    return arr
```

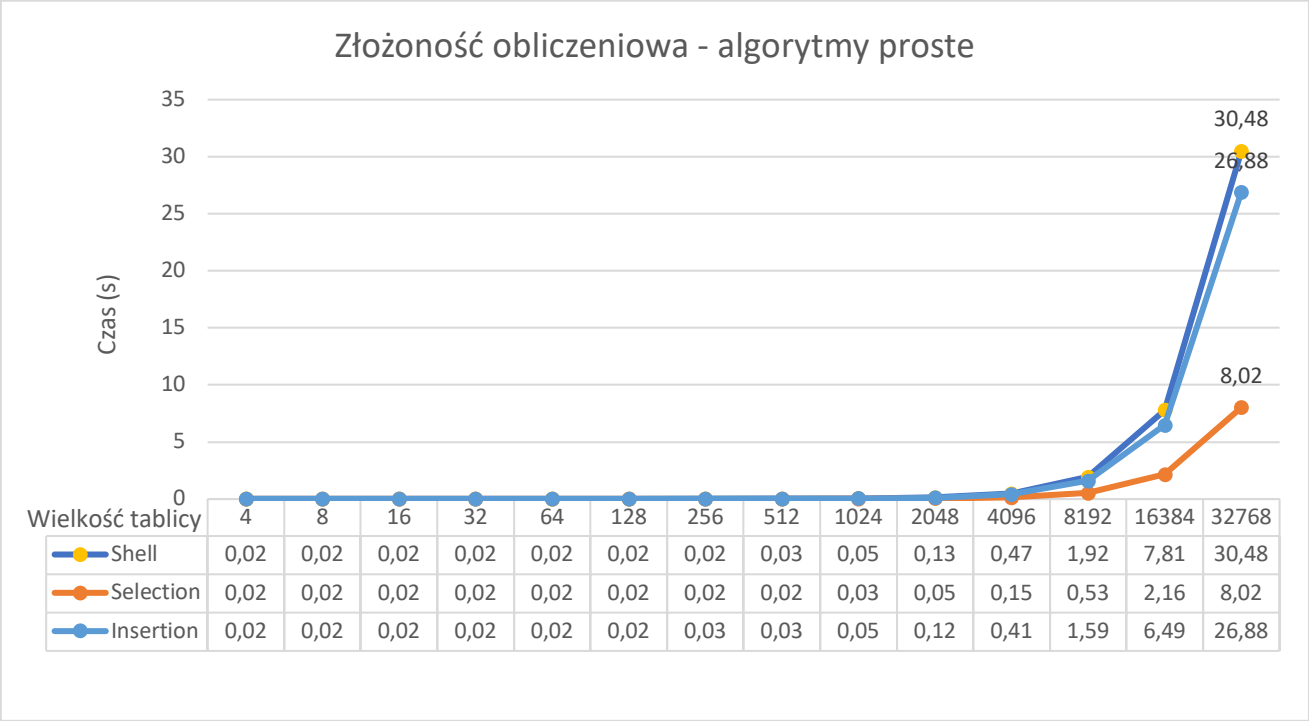
Kod algorytmu 4

Złożoność obliczeniowa algorytmu	Zbuduj kopiec (przywracanie własności kopca na losowym ciągu)	$O(n \cdot log(n))$
	Zamień korzeń z „ostatnim” liściem (ostatni element tablicy)	$O(1)$
	Przywróć własność kopca	$O(log(n))$
	Jeśli w tablicy są elementy nieposortowane, idź do kroku drugiego	$O(n \cdot log(n))$

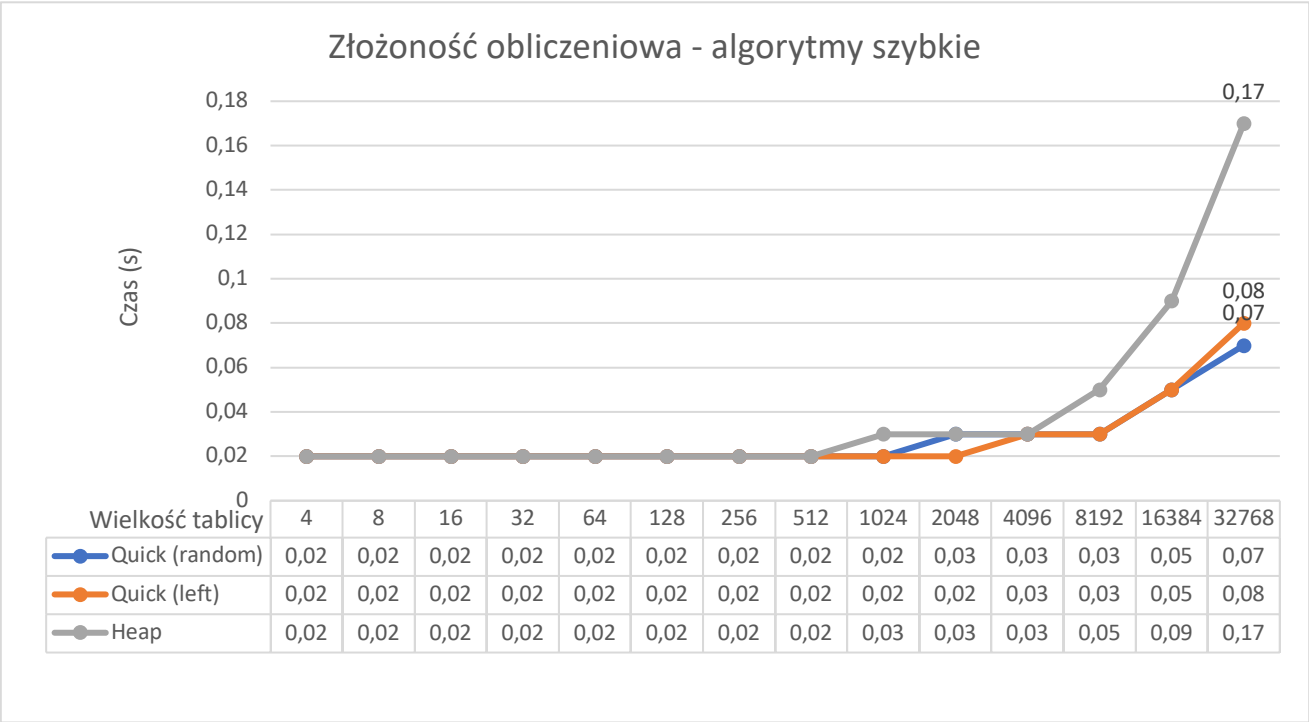
Dane wejściowe nie mają znaczenia

Złożoność czasowa algorytmów

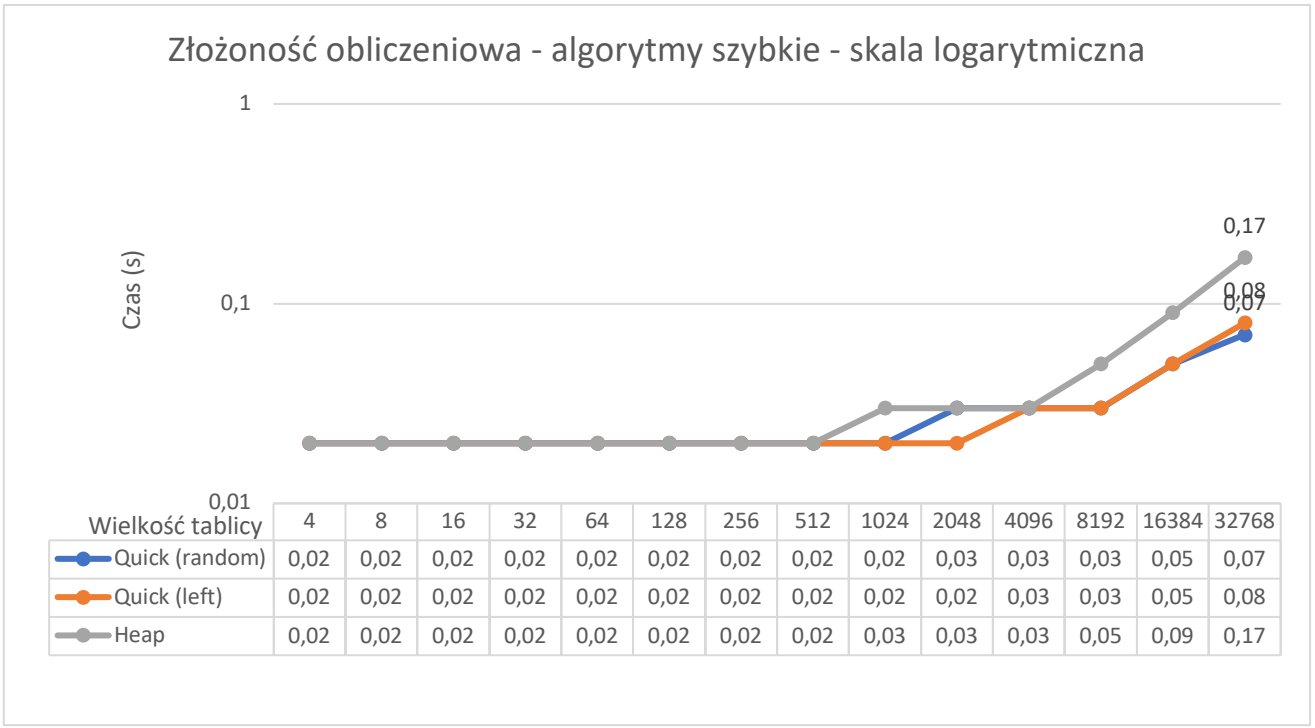
Algorytm	Wielkość tablicy	Czas (s)
Insertion Sort	32768	26,88
Selection Sort		8,02
Shell Sort		30,48
Quick Sort (random pivot)		0,07
Quick Sort (left pivot)		0,08
Heap Sort		0,17



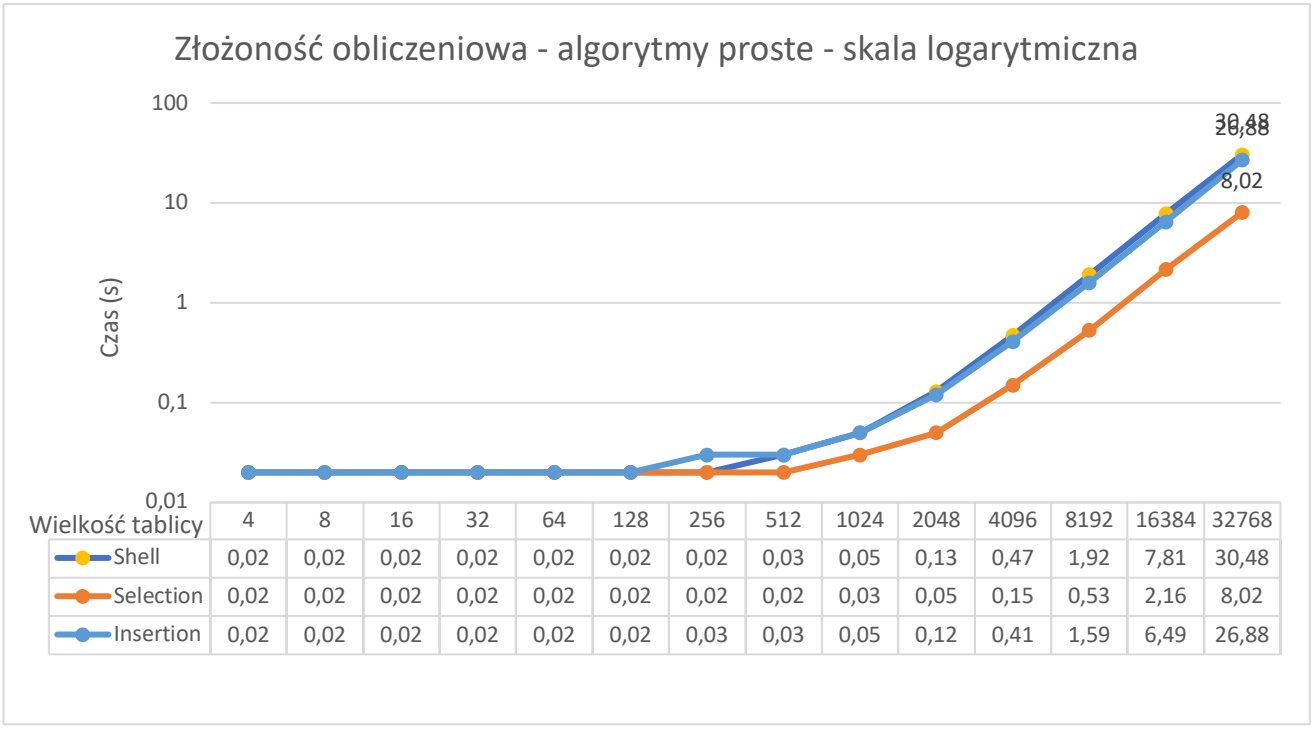
Wielkość tablicy	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768
Shell	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,03	0,05	0,13	0,47	1,92	7,81	30,48
Selection	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,03	0,05	0,15	0,53	2,16	8,02
Insertion	0,02	0,02	0,02	0,02	0,02	0,02	0,03	0,03	0,05	0,12	0,41	1,59	6,49	26,88



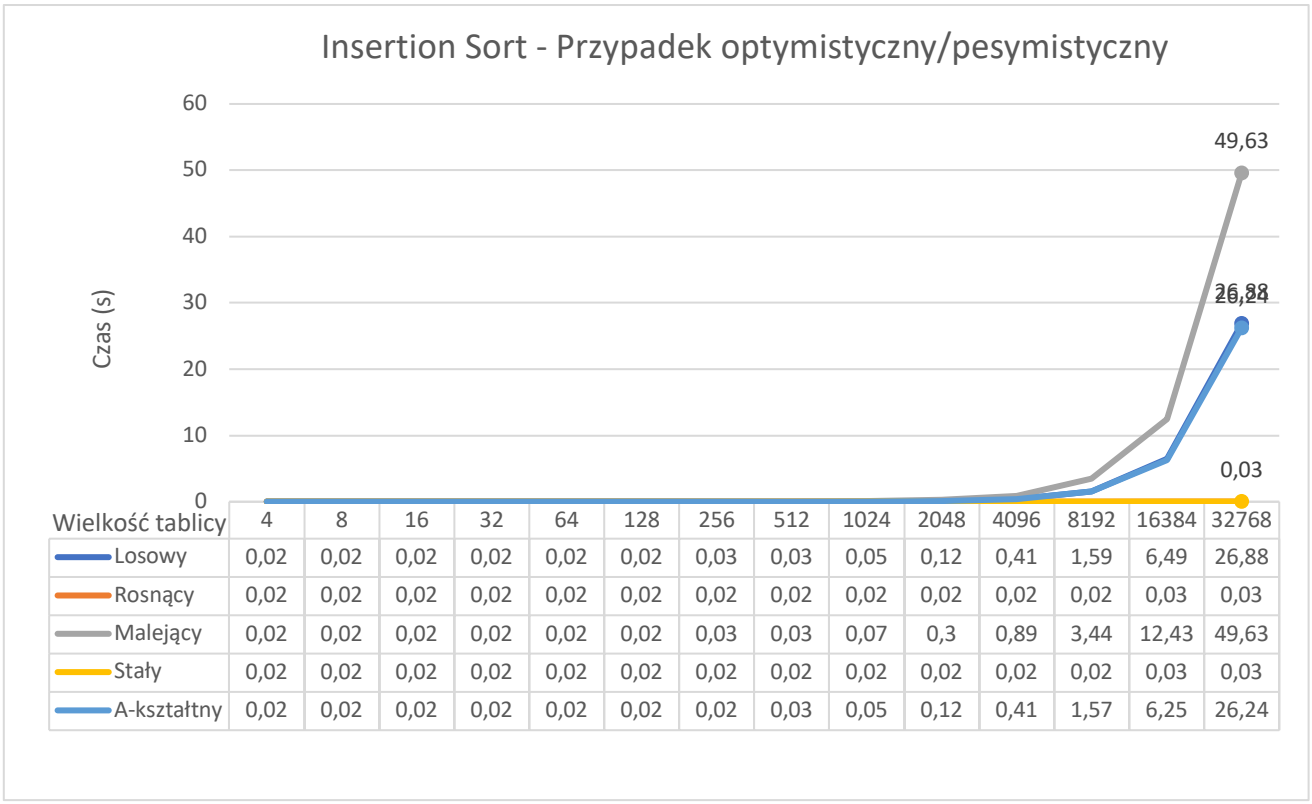
Wielkość tablicy	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768
Quick (random)	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,03	0,03	0,03	0,05	0,07
Quick (left)	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,03	0,03	0,05	0,08
Heap	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,03	0,03	0,03	0,05	0,09	0,17



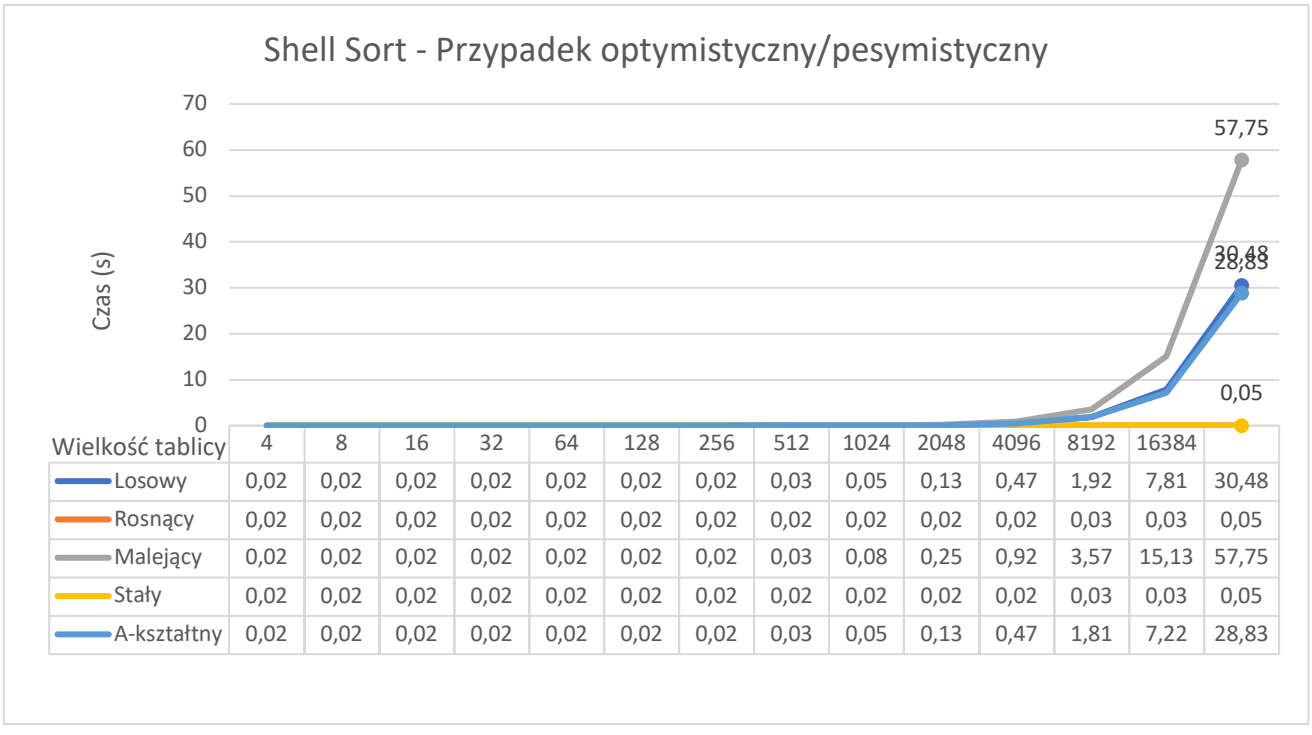
Wielkość tablicy	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768
Quick (random)	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,03	0,03	0,03	0,05	0,07
Quick (left)	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,03	0,03	0,05	0,08
Heap	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,03	0,03	0,03	0,05	0,09	0,17



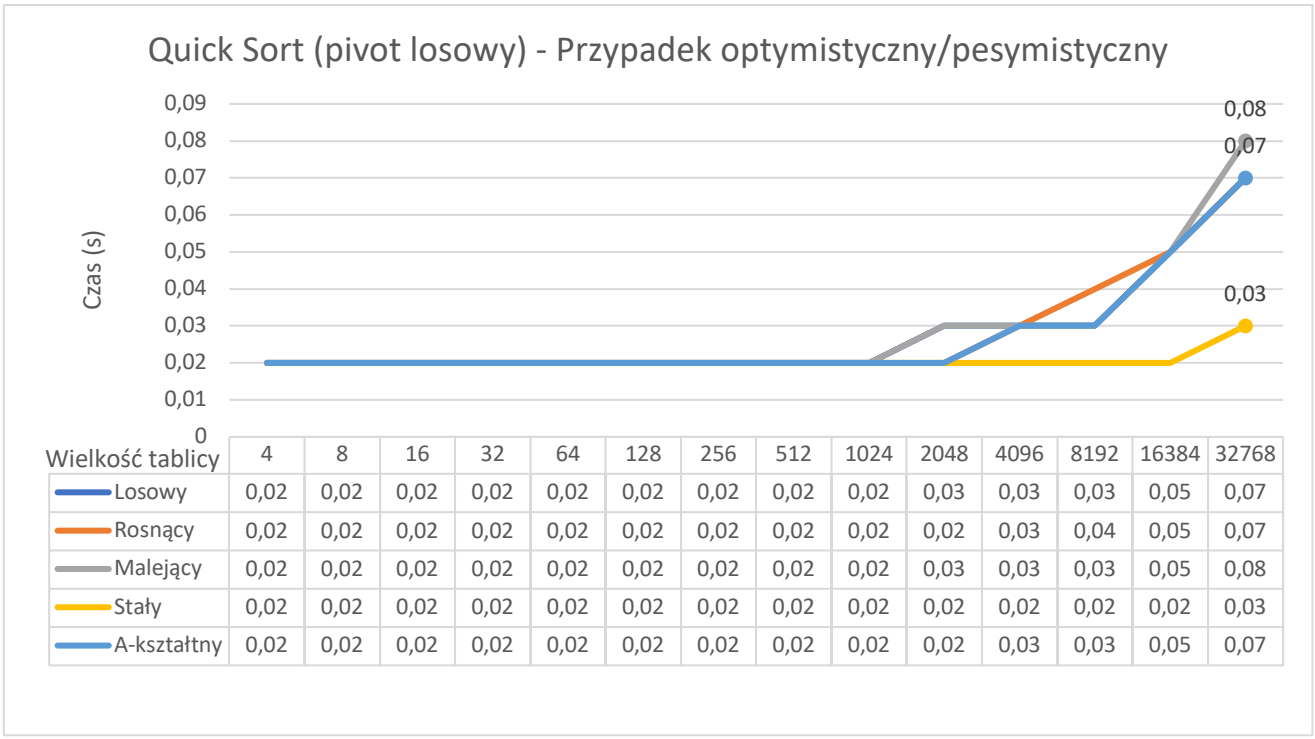
Wielkość tablicy	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768
Shell	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,03	0,05	0,13	0,47	1,92	7,81	30,48
Selection	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,03	0,05	0,15	0,53	2,16	8,02
Insertion	0,02	0,02	0,02	0,02	0,02	0,02	0,03	0,03	0,05	0,12	0,41	1,59	6,49	26,88



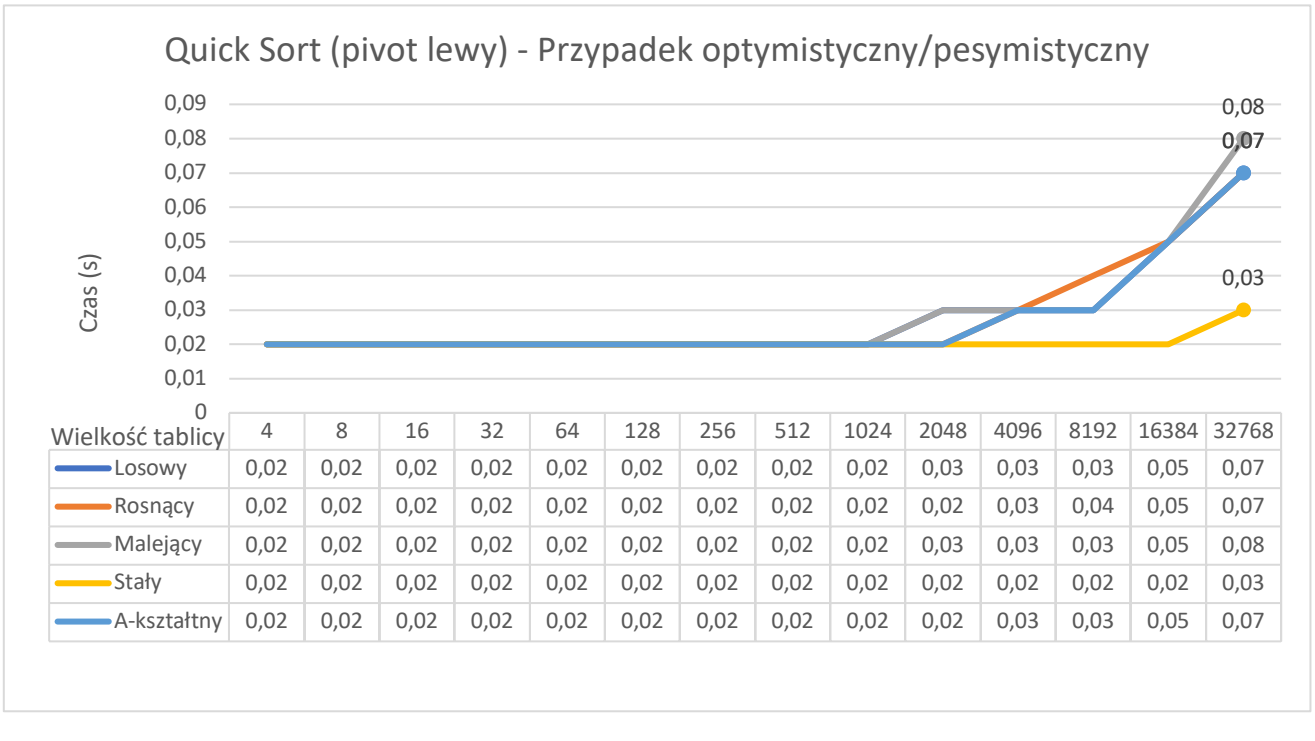
Wielkość tablicy	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768
Losowy	0,02	0,02	0,02	0,02	0,02	0,02	0,03	0,03	0,05	0,12	0,41	1,59	6,49	26,88
Rosnący	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,03	0,03
Malejący	0,02	0,02	0,02	0,02	0,02	0,02	0,03	0,03	0,07	0,3	0,89	3,44	12,43	49,63
Stały	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,03	0,03
A-kształtny	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,03	0,05	0,12	0,41	1,57	6,25	26,24



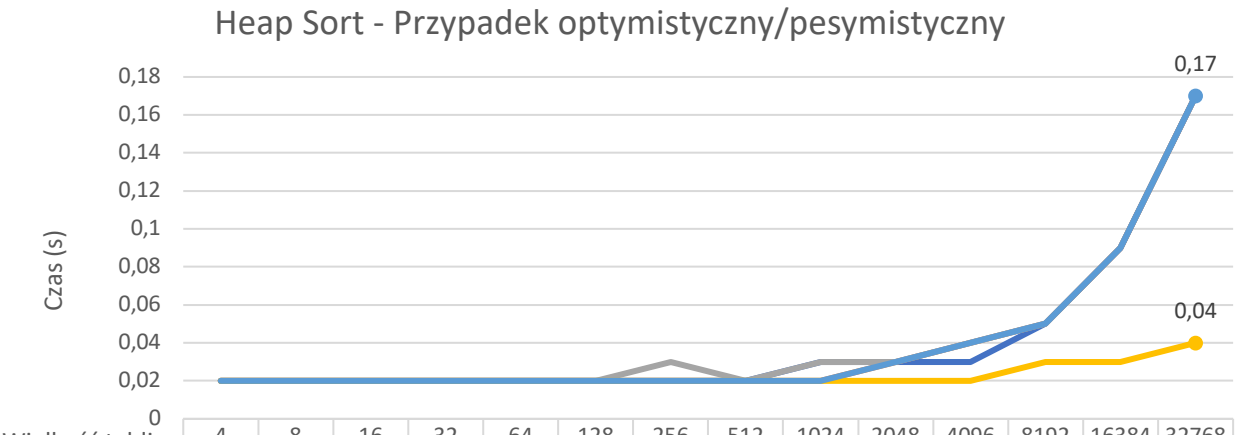
Wielkość tablicy	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	
Losowy	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,03	0,05	0,13	0,47	1,92	7,81	30,48
Rosnący	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,03	0,03	0,05
Malejący	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,03	0,08	0,25	0,92	3,57	15,13	57,75
Stały	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,03	0,03	0,05
A-kształtny	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,03	0,05	0,13	0,47	1,81	7,22	28,83



Wielkość tablicy	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768
Losowy	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,03	0,03	0,03	0,05	0,07
Rosnący	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,03	0,04	0,05	0,07
Malejący	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,03	0,03	0,03	0,05	0,08
Stały	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,03
A-kształtny	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,03	0,03	0,05	0,07



Wielkość tablicy	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768
Losowy	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,03	0,03	0,03	0,05	0,07
Rosnący	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,03	0,04	0,05	0,07
Malejący	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,03	0,03	0,03	0,05	0,08
Stały	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,03
A-kształtny	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,03	0,03	0,05	0,07



Wielkość tablicy	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768
Losowy	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,03	0,03	0,03	0,05	0,09	0,17
Rosnący	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,03	0,04	0,05	0,09	0,17
Malejący	0,02	0,02	0,02	0,02	0,02	0,02	0,03	0,02	0,03	0,03	0,04	0,05	0,09	0,17
Stały	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,03	0,03	0,04
A-kształtny	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,03	0,04	0,05	0,09	0,17

Podsumowanie

Sprawozdanie wykazało, jak duże znaczenie ma prawidłowa optymalizacja algorytmów oraz znajomość zagadnienia złożoności obliczeniowej. Zauważyliśmy również rozbieżność czasu, jaki potrzebny jest na wykonanie algorytmu w przypadku optymistyczny względem pesymistycznego.