Analiza eksperymentalna nowych algorytmów sortowania w miejscu

Damian Baliński

Praca napisana pod kierunkiem
dra Zbigniewa Gołębiewskiego

2021, Wrocław

Motywacja

Cele tworzenia algorytmów hybrydowych

- *poprawa wydajności*
- *optymalizacja przypadku pesymistycznego*
- *działanie w miejscu*
- *minimalizacja liczby porównań*

Metodologia

Łączny koszt operacji

$$C = \alpha n_c + 3n_s + n_a$$

α – wartość współczynnika kosztu
n_c – liczba operacji porównania
n_s – liczba operacji zamiany miejsc
n_a – liczba operacji przypisania

Wady algorytmów podstawowych

Wady algorytmu Quick Sort

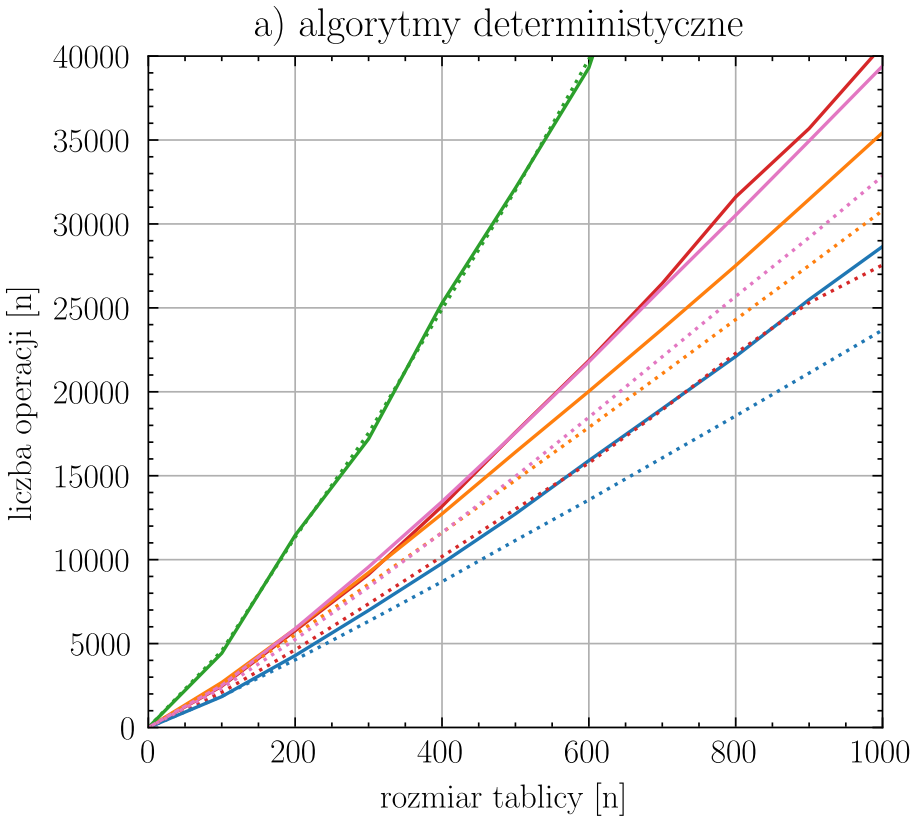
- *$O(n^2)$ dla przypadku pesymistycznego*
- *duża liczba porównań*

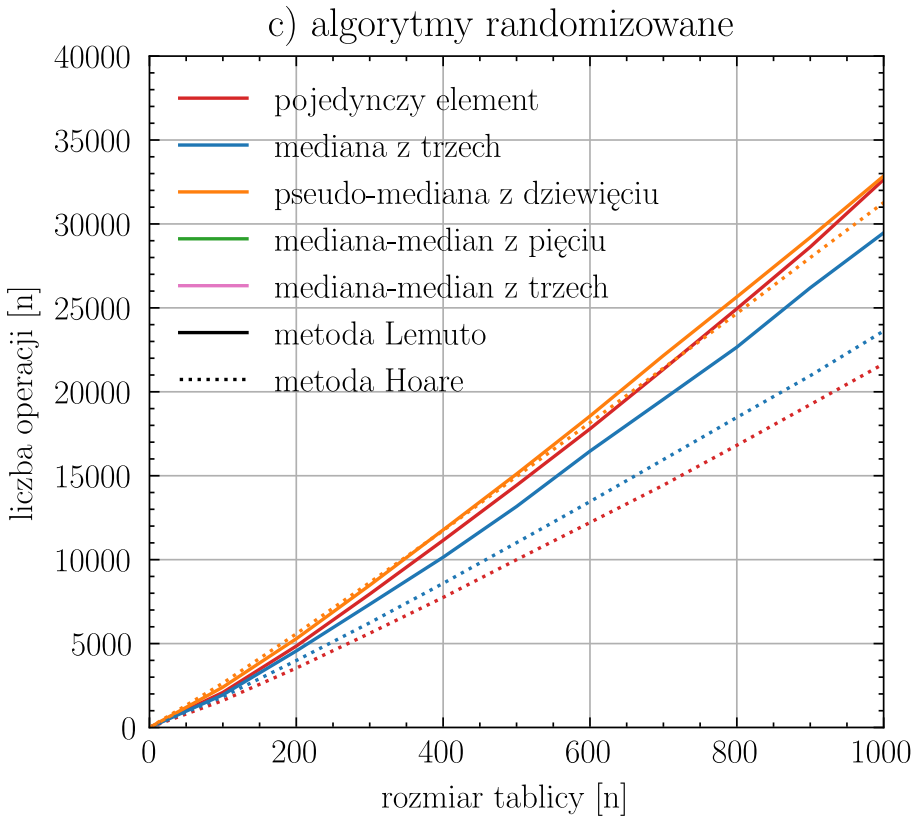
Wady algorytmu Merge Sort

- *Konieczność posiadania dodatkowej pamięci o wielkości $O(n)$*
- *Dodatkowy nakład czasowy związany z alokacją oraz zwalnianiem pamięci*

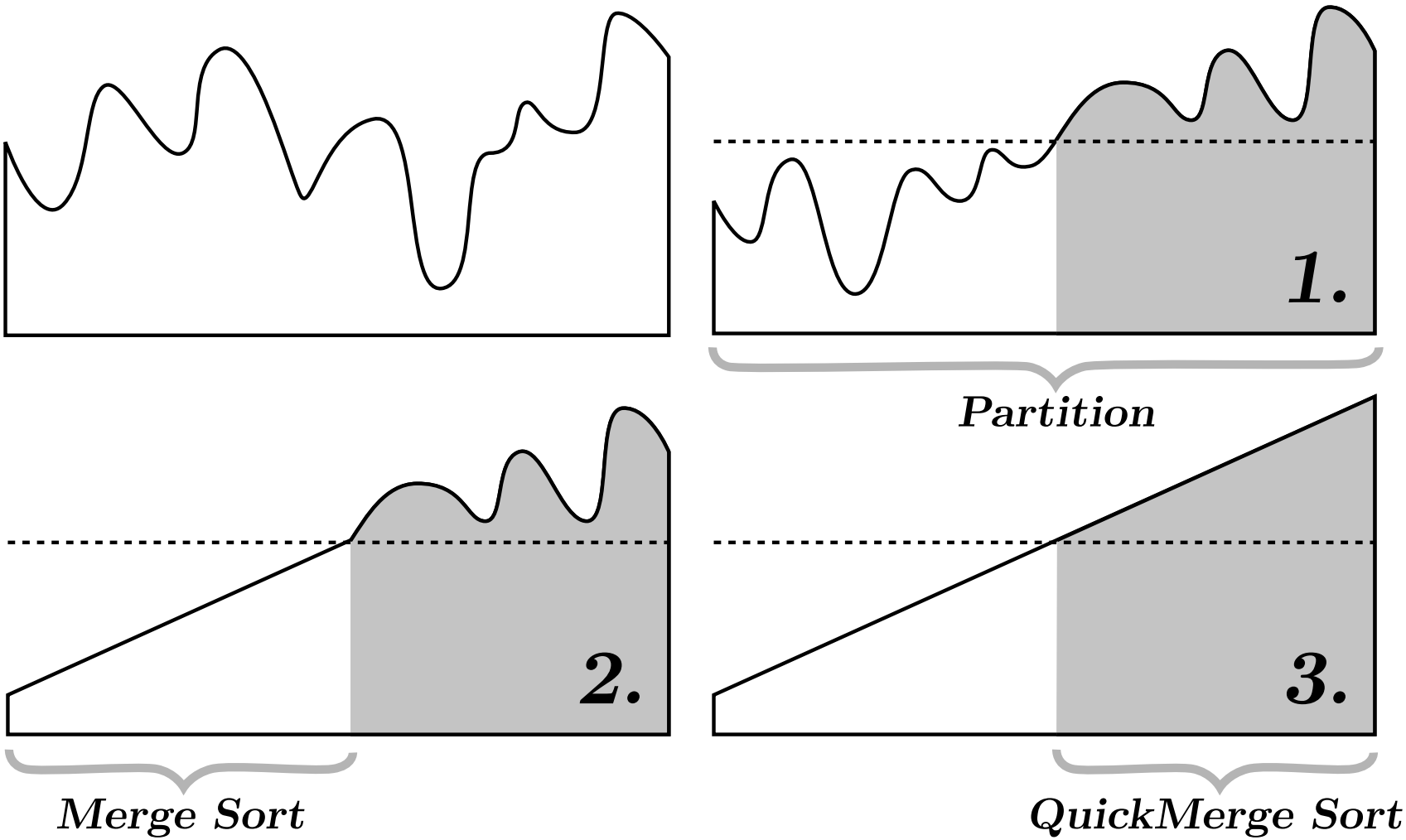
Rodzina algorytmów Quick Sort

Łączna liczba operacji wykonanych przez algorytmy z rodziny Quick Sort z podziałem na metody partycjonowania oraz polityki wyboru pivota dla tablicy uporządkowanych danych





QuickMerge Sort - schemat

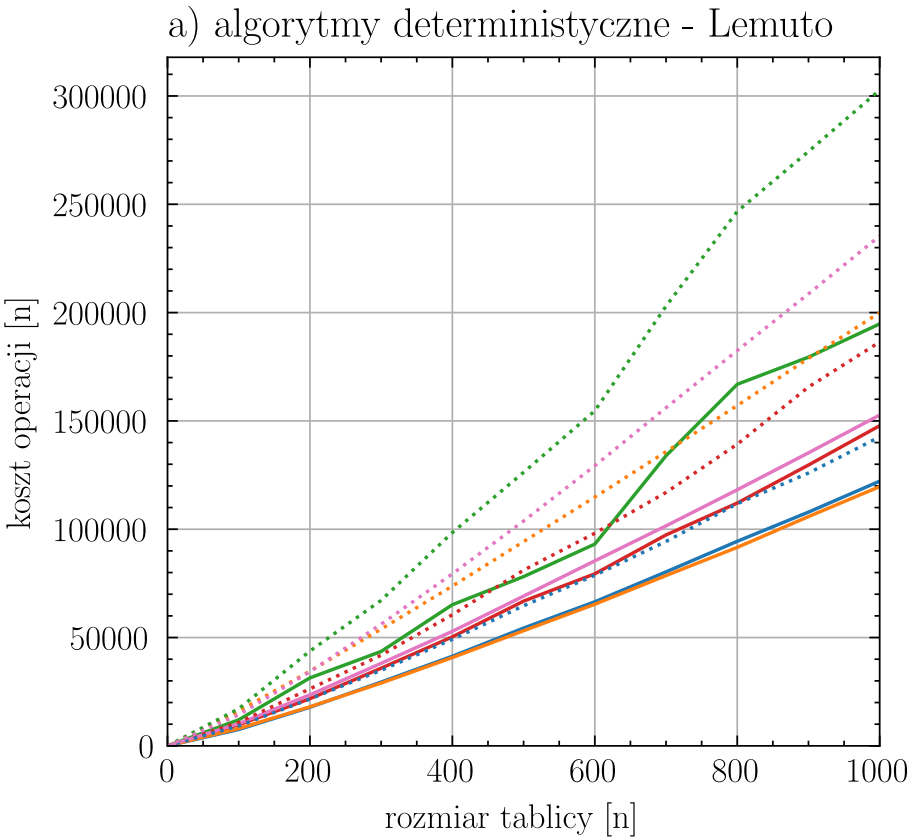


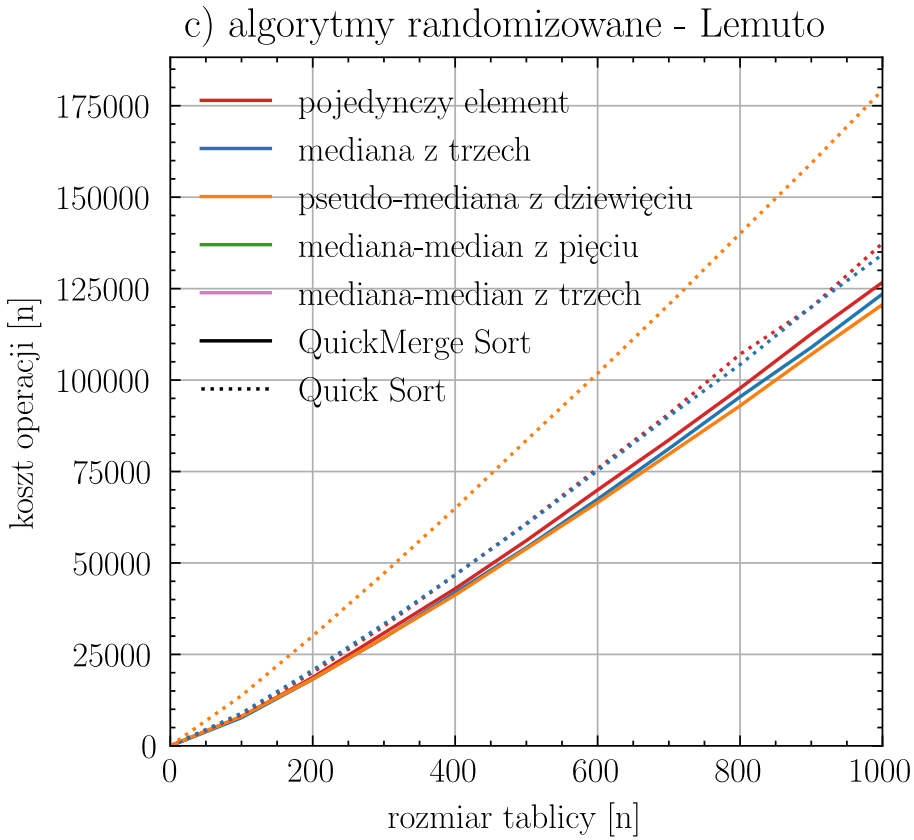
QuickMerge Sort - pseudokod

```
1:  procedure QuickMergeSort(arr)
2:
3:      if len(arr) = 1 then end                          ▷ warunek wyjścia
4:
5:      arr_1, arr_2 ← Partition(arr)                     ▷ partycjonowanie
6:
7:      if len(arr_1) < len(arr_2) then                   ▷ sortowanie
8:          buffer ← arr_2
9:          MergeSortBySwaps(arr_1, buffer)
10:         QuickMergeSort(arr_2)
11:     else
12:         buffer ← arr_1
13:         MergeSortBySwaps(arr_2, buffer)
14:         QuickMergeSort(arr_1)
```

Rodzina algorytmów QuickMerge Sort

Łączny koszt operacji wykonanych przez algorytmy z rodziny QuickMerge Sort z podziałem na metody partycjonowania oraz polityki wyboru pivota dla tablicy uporządkowanych danych przy współczynniku kosztu równym $\alpha = 10.0$



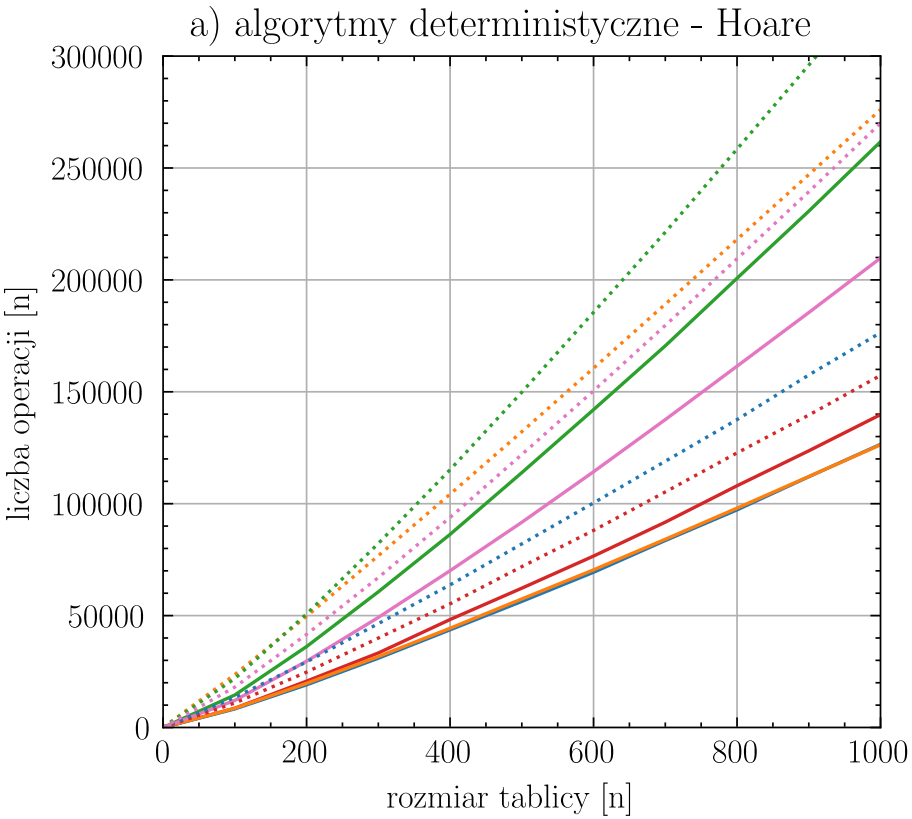


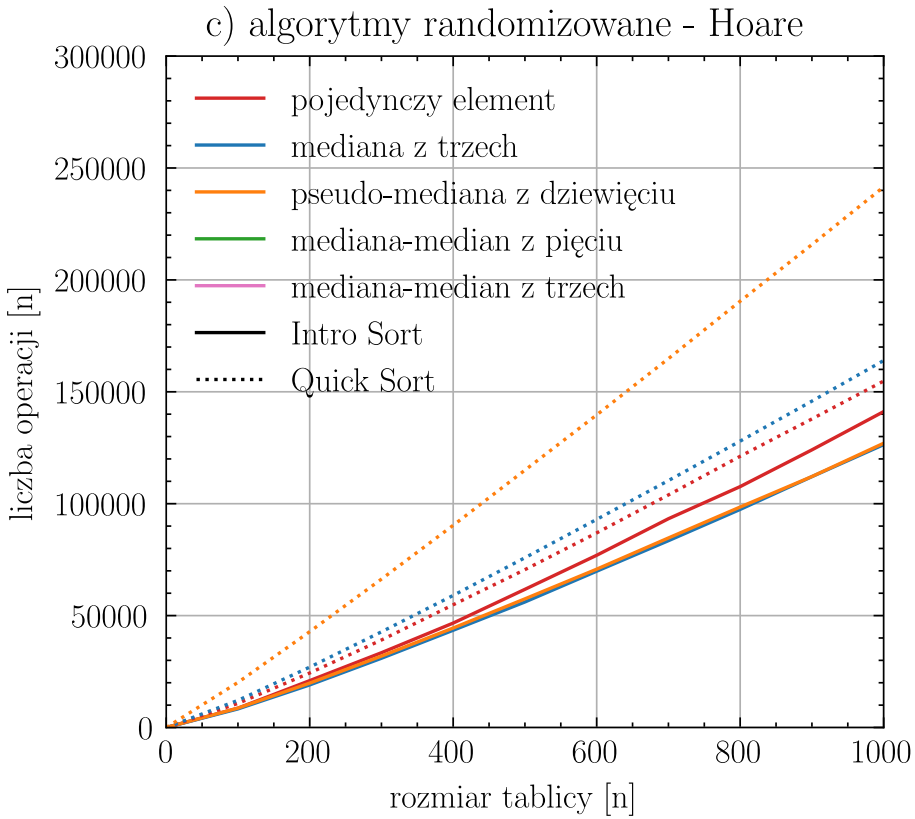
Intro Sort - pseudokod

```
1:  procedure IntroSort(arr, depth)
2:
3:      if len(arr) < maxLength then          ▷ sort. przez wstawianie
4:          InsertionSort(arr)
5:
6:      else if depth = 0 then                ▷ sort. przez kopcowanie
7:          HeapSort(arr)
8:
9:      else                                  ▷ szybkie sortowanie
10:         arr_1, arr_2 ← Partition(arr)
11:         IntroSort(arr_1, depth-1)
12:         IntroSort(arr_2, depth-1)
```

Rodzina algorytmów Intro Sort

Łączna liczba operacji wykonanych przez algorytmy z rodziny Intro Sort z podziałem na polityki wyboru pivota dla tablicy losowych danych





Bibliografia

[EW18a] Stefan Edelkamp and Armin Weiß.
Quickmergesort: Practically efficient constant-factor optimal sorting.
2018.

[EW18b] Stefan Edelkamp and Armin Weiß.
Worst-case efficient sorting with quickmergesort.
2018.

[WE19] Sebastian Wild, Armin Weiß and Stefan Edelkamp.
Quickxsort – a fast sorting scheme in theory and practice.
2019.

Dziękuję za uwagę.