

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*

Quick Sort

Wady algorytmu Quick Sort

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

Merge Sort

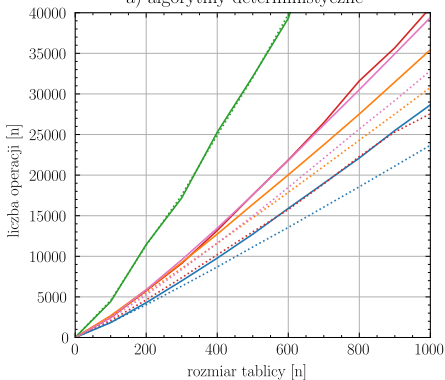
Wady algorytmu Merge Sort

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

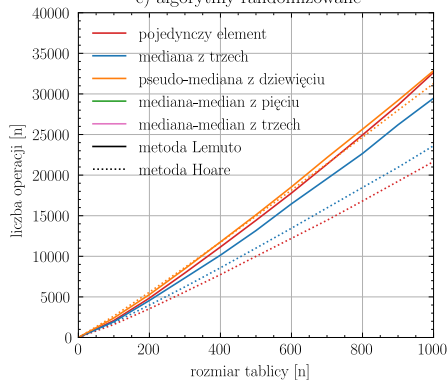
Rodzina algorytmów Quick Sort

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

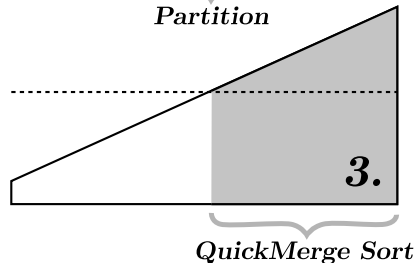
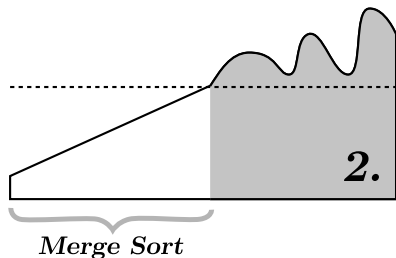
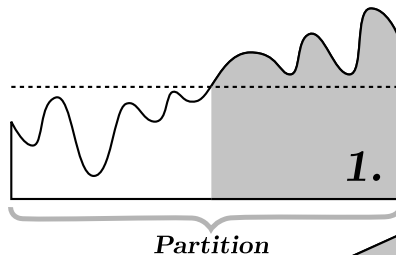
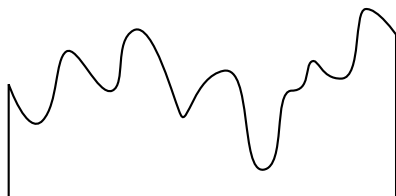
a) algorytmy deterministyczne



c) algorytmy randomizowane



QuickMerge Sort - schemat



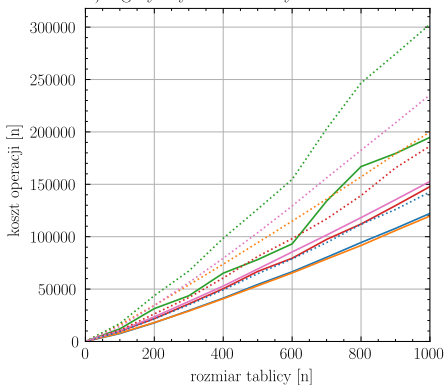
QuickMerge Sort - pseudokod

```
1: procedure QUICKMERGESORT(ARR)
2:
3:   if  $\text{len}(\text{arr}) = 1$  then end                                ▷ warunek wyjścia
4:
5:    $\text{arr}_1, \text{arr}_2 \leftarrow \text{Partition}(\text{arr})$                       ▷ partycjonowanie
6:
7:   if  $\text{len}(\text{arr}_1) < \text{len}(\text{arr}_2)$  then                            ▷ sortowanie
8:      $\text{buffer} \leftarrow \text{arr}_2$ 
9:     MergeSortBySwaps( $\text{arr}_1$ ,  $\text{buffer}$ )
10:    QuickMergeSort( $\text{arr}_2$ )
11:  else
12:     $\text{buffer} \leftarrow \text{arr}_1$ 
13:    MergeSortBySwaps( $\text{arr}_2$ ,  $\text{buffer}$ )
14:    QuickMergeSort( $\text{arr}_1$ )
```

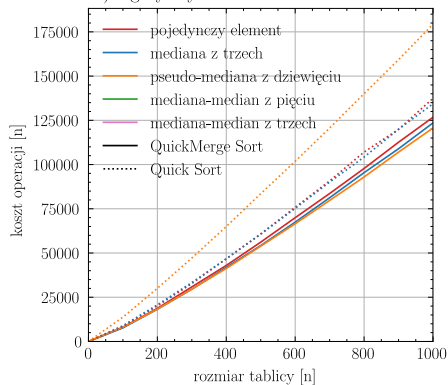
Rodzina algorytmów QuickMerge Sort

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

a) algorytmy deterministyczne - Lemuto



c) algorytmy randomizowane - Lemuto



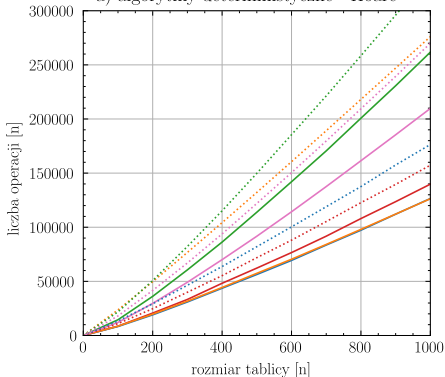
Intro Sort - pseudokod

```
1: procedure INTROSORT(ARR, DEPTH)
2:
3:   if  $\text{len}(\text{arr}) < \text{maxLength}$  then           ▷ sort. przez wstawianie
4:     InsertionSort(arr)
5:
6:   else if  $\text{depth} = 0$  then                       ▷ sort. przez kopcowanie
7:     HeapSort(arr)
8:
9:   else                                           ▷ szybkie sortowanie
10:     $\text{arr}_1, \text{arr}_2 \leftarrow \text{Partition}(\text{arr})$ 
11:    IntroSort(arr1, depth-1)
12:    IntroSort(arr2, depth-1)
```

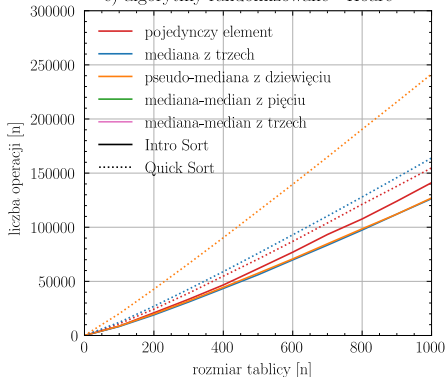
Rodzina algorytmów Intro Sort

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

a) algorytmy deterministyczne - Hoare



c) algorytmy randomizowane - Hoare



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.