**Analiza eksperymentalna nowych algorytmów sortowania w miejscu**

1. **Wstęp**

W mojej pracy badałem hybrydowe algorytmy sortujące, będące połączeniem różnych technik sortujących

* 1. **Analiza hybrydowych algorytmów** – TODO opis
  2. **Deterministyczne i randomizowane działanie algorytmów** – TODO

1. **Motywacja**

Zasadniczo głównym powodem tworzenia hybrydowych algorytmów sortujących jest poprawa wydajności algorytmów podstawowych. Czasami wprowadzając nieznaczną modyfikację, jak na przykład wybór piwota jako mediana z trzech elementów zyskujemy poważną optymalizację

* 1. **Wydajność** – minimalna liczba wykonywanych operacji
  2. **Działanie w miejscu** – brak potrzeby alokacji dodatkowej pamięci
  3. **Optymalizacja przypadku pesymistycznego** – logarytmiczno-liniowe działanie algorytmu, niezależnie od porządku danych wejściowych
  4. **Mała liczba porównań** – w praktycznych zastosowaniach sortujemy złożone struktury, dla których pojedyncza operacja porównania jest znacznie kosztowniejsza niż np. operacja przypisania

1. **Quick Sort**

Zanim przejdę do analizy algorytmów hybrydowych, chciałbym po krótce przedstawić problemy z jakimi musimy się zmierzyć stosując standardowe algorytmy sortujące.

* 1. **O(n2) czas działania dla przypadku pesymistycznego** – dla uporządkowanych danych wejściowych nie sprawdza się technika dziel i zwyciężaj wykorzystywana w algorytmie Quick Sort, głębokość drzewa wywołań rekurencyjnych jest równa długości tablicy, sam algorytm Quick Sort działa w czasie kwadratowym
  2. **Duża liczba porównań** – algorytm wykonuje prawie 2 razy więcej operacji porównania w stosunku do operacji przypisania, problem w przypadku sortowania złożonych struktur danych

1. **Merge Sort**
   1. **Konieczność alokacji O(n) dodatkowej pamięci** – Merge Sort potrzebuje dodatkowego bufora, dodatkowy czas na przydzielenie pamięci, nie wszystkie systemy posiadają zasoby aby móc to wykonać
2. **Algorytmy z rodziny Quick Sort**

Tak jak wcześniej wspomniałem, czasami stosunkowo niewielka zmiana może znacząco poprawić wydajność algorytmów sortujących. W mojej pracy badałem wydajność różnych wersji algorytmu Quick Sort pod kątem optymalizacji dla przypadku pesymistycznego, z podziałem na metodę partycjonowania oraz strategię wyboru piwota.

* 1. **Partycjonowanie Hoare, losowy piwot** - najwydajniejszą z badanych metod jest partycjonowanie metodą Hoare przy wyborze piwota jako losowy element, dla uporządkowanych danych, czyli przypadku pesymistycznego w przypadku klasycznego Quick Sort, zmodyfikowany algorytm działa w czasie liniowo-logarytmicznym
  2. **Partycjonowanie Hoare, mediana z trzech** – w przypadku algorytmów deterministycznych, najwydajniejsze jest partycjonowanie metodą Hoare oraz wybór piwota jako mediana z trzech elementów, np. pierwszy, środkowy oraz ostatni element tablicy.

1. **QuickMerge Sort – rysunek**
   1. **Sebastian Wild –** w swojej pracy zatytułowanej **QuickXsort – A Fast Sorting Scheme in Theory and Practice** autor przedstawił koncepcję połączenia

QuickMerge Sort to algorytm Hybrydowy, będący połączeniem algorytmów Quick Sort oraz Merge Sort.

Algorytm rozwiązuje problem z alokacją pamięci dla typowego algorytmu Merge Sort

Celem tego algorytmu jest minimalizacja wykonywanych operacji porównania, przy równoczesnym zachowaniu działania w miejscu. Zasadniczą koncepcją tego algorytmu jest podział tablicy wejściowej na dwie części oraz posortowanie jednej z nich algorytmem Merge Sort w taki sposób, aby bufor potrzebnej pamięci stanowiła druga część tablicy. Najlepiej jest to widoczne w pseudokodzie.

1. **QuickMerge Sort – pseudokod**

QuickMerge Sort to algorytm rekurencyjny, składający się z trzech kroków.

* 1. **Partycjonowanie** – na początku partycjonowana jest tablica wejściowa, przy użyciu metod deterministycznych lub niedeterministycznych
  2. **Sortowanie mniejszej części algorytmem Merge Sort** – przy czym bufor stanowi druga część tablicy
  3. **Sortowanie drugiej części** – rekurencyjne wywołanie algorytmu dla pozostałej podtablicy

1. **QuickMerge Sort – wykres**

Ponieważ składową częścią tego algorytmu jest partycjonowanie ciągu wejściowego, algorytm ten również był analizowany pod kątem technik deterministycznych oraz randomizowanych. Algorytmy badałem pod kątem całkowitego kosztu wykonanych operacji na podstawie współczynnika kosztu, współczynnik kosztu określa ile razy operacja porównania jest droższa od operacji przypisania, ma to sens, jeśli np. sortujemy złożone struktury, oraz operujemy na wskaźnikach do tych struktur. Wtedy przypisanie to podmiana wskaźników, zaś porównanie to dereferencja wszystkich składowych struktury.

Z przeprowadzonej analizy wynika, że w przypadku sortowania złożonych struktur danych, tzn. takich dla których operacja porównania jest znacznie kosztowniejsza niż operacja przypisania, algorytm QuickMerge Sort jest znacznie wydajniejszy niż klasyczny algorytm Quick Sort. Najwydajniejszą kombinację otrzymano łącząc algorytm QuickMerge Sort z partycjonowaniem Lemuto przy wyborze piwota jako pseudo-mediana z trzech elementów lub mediana z trzech.

1. **Intro Sort – pseudokod**
2. **Intro Sort – wykres**
3. **Podziękowanie**