2) **HybridSort**

HybridSort łączy szybkość QuickSort dla dużych zbiorów danych z efektywnością InsertionSort dla małych podzbiorów, co pozwala na znaczącą optymalizację wydajności poprzez eliminację słabostek obu algorytmów. Przełączanie na InsertionSort dla $n \leq 20$ jest kluczowym czynnikiem, jest bardzo szybki dla małych zbiorów danych ze względu na mniejszą liczbę operacji wymaganych do porównywania i zamiany elementów.

InsertionSort

InsertionSort jest najbardziej efektywny dla małych zbiorów Jego prostota i brak potrzeby dodatkowej pamięci sprawiają, że jest idealny dla małych zbiorów. Jednakże, jego wydajność szybko maleje w miarę wzrostu n, co jest związane z kwadratową złożonością czasową. To sprawia, że nie jest on zalecany dla dużych zbiorów danych, ale jako element **HybridSort**, znakomicie sprawdza się w optymalizacji końcowych etapów sortowania.

CustomSort

CustomSort jest interesującym wariantem **MergeSort**, który wykorzystuje naturalne rosnące podciągi w danych wejściowych. To podejście może znacząco zmniejszyć liczbę porównań i operacji wymaganych do sortowania, ponieważ podciągi są już częściowo posortowane. Dla danych, które naturalnie zawierają takie podciągi, **CustomSort** może oferować lepszą wydajność niż klasyczny **MergeSort**, szczególnie w kontekście redukcji operacji porównania.

MergeSort

MergeSort to kolejny algorytm typu dziel i zwyciężaj, który jest znany z gwarantowanej złożoności $O(n\ln(n))$ w najgorszym, średnim i najlepszym przypadku.

- **Stabilność:** MergeSort jest stabilny w różnych scenariuszach, pokazując zbliżoną liczbę porównań niezależnie od rozmiaru danych. To sprawia, że jest niezawodnym wyborem dla szerokiego zakresu aplikacji.
- **Porównanie z CustomSort:** MergeSort i CustomSort wykazują podobne zachowania, co sugeruje, że CustomSort z jego podejściem opartym na rosnących podciągach może być efektywnym ulepszeniem w pewnych przypadkach.

QuickSort

QuickSort jest klasycznym algorytmem sortowania, który wykorzystuje metodę dziel i zwyciężaj. Na dostarczonych grafikach **QuickSort** generalnie wykazuje dobrą wydajność, ale nie jest tak efektywny jak **DualPivotQuickSort**, szczególnie dla większych zbiorów danych.

- **Małe zbiory danych:** QuickSort radzi sobie dobrze, ale jest prześcignięty przez DualPivotQuickSort, co może wynikać z efektywniejszego podziału w DualPivotQuickSort.
- **Duże zbiory danych:** Wzrost liczby porównań i zamian dla QuickSort w stosunku do DualPivotQuickSort sugeruje, że klasyczne podejście do wyboru pivota może nie być optymalne dla wszystkich rodzajów danych.

DualPivotQuickSort

DualPivotQuickSort, ulepszona wersja QuickSort, używa dwóch punktów pivotowych do podziału tablicy na trzy segmenty, co teoretycznie pozwala na szybsze osiągnięcie celu sortowania.

- **Efektywność:** Grafiki pokazują, że DualPivotQuickSort wykazuje mniejszą liczbę porównań i zamian niż QuickSort, co potwierdza jego wyższą efektywność, zwłaszcza w dużych zbiorach danych.
- **Stała C:** Niska wartość stałej C dla DualPivotQuickSort wskazuje na wydajność bliższą optymalnemu, co świadczy o jego efektywności w praktycznych zastosowaniach.

3) Porównania i zamiany dla CustomSort i MergeSort

- **CustomSort** i **MergeSort** wykazują podobne tendencje w liczbie porównań i zamian w miarę wzrostu rozmiaru danych. Podobieństwo to jest oczekiwane, ponieważ MergeSort służy jako punkt odniesienia dla efektywnych algorytmów sortujących, a CustomSort stosuje strategię dziel i zwyciężaj podobną do MergeSort.
- Stosunek liczby porównań do n (**c/n**) pozostaje stosunkowo stały lub nieznacznie rośnie wraz z n, co wskazuje na wzrost wydajności lub stabilną wydajność w miarę wzrostu rozmiaru danych.
- **Poszukiwanie rosnących podciągów: CustomSort** zaczyna od znajdowania rosnących podciągów w oryginalnej tablicy. Każdy taki podciąg jest traktowany jako już częściowo posortowany, co stanowi bazę dla dalszego sortowania. Choć to podejście może wydawać się optymalne, ponieważ teoretycznie redukuje liczbę potrzebnych porównań przez wykorzystanie naturalnej kolejności elementów, w praktyce może prowadzić do dodatkowych porównań. Każde porównanie, potrzebne do zidentyfikowania granic podciągów, dodaje się do ogólnej liczby porównań, co nie występuje w tradycyjnym MergeSort.
- **MergeSort bez optymalizacji:** Tradycyjny **MergeSort** działa przez rekurencyjne dzielenie tablicy na coraz mniejsze fragmenty, aż do osiągnięcia jednoelementowych lub pustych podtablic, a następnie scalanie ich w sposób uporządkowany. Ta metoda, choć może prowadzić do dużej liczby porównań, jest bardzo przewidywalna i skuteczna, ponieważ każdy podział i scalenie jest dokładnie zaplanowane i nie wymaga dodatkowych porównań poza tymi niezbędnymi do scalania.

- **Złożoność**: W najlepszym przypadku, gdy dane wejściowe są już całkowicie posortowane (jeden duży rosnący podciąg), SmartMergeSort może teoretycznie osiągnąć złożoność bliską O(n), gdzie n jest liczbą elementów. W porównaniu, MergeSort zawsze będzie miał złożoność O(n log n), niezależnie od uporządkowania danych wejściowych.

4) 1. QuickSort vs DualPivotQuickSort

Małe Zbiory Danych (n od 10 do 50)

Porównania i Zamiany

- **DualPivotQuickSort** wykazuje mniejszą liczbę porównań niż **QuickSort** w małych zbiorach danych dla wszystkich wartości k. To może wynikać z efektywniejszego podziału danych na więcej niż dwa podzbiory, co pozwala na szybsze zmniejszanie rozmiaru problemu .
- Liczba zamian w **DualPivotQuickSort** również jest generalnie niższa, co sugeruje, że jego strategia dwóch osi pivotowych może skutkować mniejszą koniecznością przestawiania elementów, niż ma to miejsce w przypadku standardowego QuickSort.

Wnioski dla Małych Zbiorów Danych

- W kontekście małych zbiorów danych **DualPivotQuickSort** jest bardziej wydajny, zarówno pod względem liczby porównań, jak i zamian, co sugeruje jego preferencję dla optymalizacji operacji na małą skalę.

Duże Zbiory Danych (n od 1000 do 50000)

Porównania i Zamiany

- Podobnie jak w przypadku małych zbiorów danych, **DualPivotQuickSort** zachowuje niższą liczbę porównań i zamian w dużych zbiorach danych dla wszystkich wartości k . To potwierdza skalowalność i efektywność DualPivotQuickSort w obróbce dużych ilości danych.
- Spójność w efektywności **DualPivotQuickSort** w dużych zbiorach danych podkreśla jego zdolność do efektywnego zarządzania większymi zestawami elementów, prawdopodobnie dzięki redukcji głębokości rekursji i lepszemu podziałowi danych.

Wnioski dla Dużych Zbiorów Danych

- **DualPivotQuickSort** wykazuje znaczącą przewagę nad tradycyjnym **QuickSort** w kontekście zarówno małych, jak i dużych zbiorów danych, co czyni go bardziej pożądanym wyborem dla aplikacji wymagających szybkiego i efektywnego sortowania.
- **DualPivotQuickSort jest bardziej wydajny:**

DualPivotQuickSort używa dwóch osi pivot (punkty podziału) zamiast jednej, co pozwala na równoczesne dzielenie tablicy na trzy podzbiory (elementy mniejsze niż pierwszy pivot, elementy pomiędzy pivotami i elementy większe niż drugi pivot). Ta metoda potencjalnie redukuje liczbę porównań i zamian, ponieważ każdy podział jest dokładniejszy i wymaga mniej kroków do osiągnięcia ostatecznego uporządkowania. Skuteczność tej strategii jest widoczna zarówno w małych, jak i dużych zbiorach danych, z mniejszą liczbą wymaganych operacji porównania i zamiany, co przekłada się na ogólną efektywność.

2. Stała C dla DualPivotQuickSort:

Małe Zbiory Danych (n od 10 do 50)

Dla małych zbiorów danych, wartość stałej **C** w **DualPivotQuickSort** waha się w stosunkowo wąskim zakresie, co sugeruje, że algorytm działa efektywnie nawet dla niewielkich ilości danych . Stała **C** w tych przypadkach pozostaje niska, co wskazuje na to, że liczba porównań między kluczami jest bliska optymalnej w stosunku do ilości danych i ich logarytmicznej skali wzrostu.

Duże Zbiory Danych (n od 1000 do 50000)

Dla dużych zbiorów danych, obserwujemy podobną efektywność **DualPivotQuickSort**, gdzie wartość stałej **C** utrzymuje się na niskim poziomie, co potwierdza jego wydajność w skalowaniu do większych zbiorów danych . To wskazuje, że zwiększenie rozmiaru danych nie prowadzi do nieproporcjonalnego wzrostu liczby porównań, co jest kluczowe dla utrzymania ogólnej wydajności sortowania.

Wnioski

- **Skuteczność DualPivotQuickSort:** Stała **C** dla **DualPivotQuickSort** utrzymuje się na relatywnie niskim poziomie w całym zakresie rozmiarów danych, co świadczy o jego ogólnej skuteczności i bliskości do teoretycznego optimum złożoności $O(n\ln(n))$.