

Wstęp.

Zadania po wykładzie o prostych algorytmach sortowania i analizie kosztu zamortyzowanego.

Lista zadań

1. Wykorzystując koszt zamortyzowany uzasadnij/udowodnij, że średnia złożoność operacji `add()` dodającej nowy element na końcu listy dla `ArrayList` wynosi $O(1)$.
2. Korzystając z kosztu zamortyzowanego uzasadnij/udowodnij, że średnia złożoność metody `nextPermutation()` wynosi $O(1)$.
3. Zaimplementuj wersję sortowania przez wstawianie. Algorytm ma sortować malejąco, a posortowana część ma rosnąć od prawej (od końca tablicy). Pokaż stan tablicy po każdym kroku algorytmu dla tablicy:
76,71, 5, 57,12,50,20,93,20,55,62,3
4. Zaimplementuj wersję sortowania przez wybór. Algorytm ma sortować malejąco, a posortowana część ma rosnąć od prawej (od końca tablicy). Pokaż stan tablicy po każdym kroku algorytmu dla tablicy:
76,71, 5, 57,12,50,20,93,20,55,62,3
5. Zaimplementuj wersję sortowania bąbelkowego. Algorytm ma sortować malejąco, a posortowana część ma rosnąć od lewej strony (od początku tablicy). Pokaż stan tablicy po każdym kroku algorytmu dla tablicy:
76,71, 5, 57,12,50,20,93,20,55,62,3
6. Zaimplementuj ShakerSort, wersję sortowania bąbelkowego wprowadzoną podczas wykładu.
 - a. Bez dalszych ulepszeń
 - b. Z wybranymi dwoma ulepszeniami przedstawionymi na wykładzie.
7. Pokaż stan tablicy po każdym kroku powyższego algorytmu ShakerSort. Algorytm ma sortować rosnąco. Przykładowa tablica:
76,71, 5, 57,12,50,20,93,20,55,62,3
8. Zaimplementuj algorytm sortowania poprzez generowanie kolejnych permutacji n -elementowego ciągu wejściowego i sprawdzania, czy ta permutacja jest uporządkowana. Dla jakiego n czas działania Twojego komputera dla najgorszego przypadku jest już powyżej 1 minuty?
9. Zaimplementować sortowania zwariowane polegające na „przetasowaniu” ciągu wejściowego i sprawdzeniu, czy po tej operacji powstał ciąg uporządkowany. Dla jakiej długości ciągu wejściowego n czas działania Twojego komputera jest już powyżej 1 minuty?