AiSD Laboratorium - przygotowanie do projektu 3.

Poniższe zadania mają na celu przygotowanie do projektu 3. Rozwiązania albo będa pomocne albo wręcz będą częścią projektu 3.

W całym dokumencie zakładamy, że:

- podstawowym typem danych jest `T= unsigned short`
- dane przechwywane sa w kontenerze `std::vector<T>` z biblioteki `<vector>`
- jeżeli nie zaznaczono inaczej, dane przechowywane sa w zmiennej o nazwie `data`

Należy to rozumieć, że w odpowiednim miejscu znajdują się następujace linijki

```
#include <vector>
using T = short;
std::vector<T> data = {}
```

Dodatkowo wszytkie **własne** zmienne, typy, funkcje, klasy oraz struktury danych są umieszczone w przestrzeni nazw `AiSD`

Zadanie 0.

Przygotować projekt składajacy się z plików:

- `AiSD_sort_demo.cpp` -- jako driver do zadań
- `algorithms.h` oraz `algorithms.cpp` -- zawierający wszystkie funkcje etc., które bedą potrzebne w zadaniach

w którym:

- zdefinowane są `T` oraz `data`
- Wyswietlany jest komunikat przywitalny oraz rozmiar `data`

Sprawdzić czy wszystko się kompiluje...

Zadanie 1.

Dodać do projektu funkcje:

- `void print(const std::vector<T>& data)` -- wyświetla elementy tablicy rozdzielone spacjami
- `void print_pretty(const std::vector<T>& data)` -- wyświetla rozmiar tablicy oraz jej elementy jeden pod drugim w 8 (lub 10?) kolumnach
- (*) `std::ostring& operator<<(std::ostream& os, const std::vector<T>& data)` -- realizujacy to samo co `print`

Przetestować.

Zadanie 2.

Dodać do projektu funkcje lub klasę umożliwiajacą wypełnienie tablicy:

- wartościami z zakresu od `m` do `M`
- wartościami z zakresu od `m` do `M` w odwrotnej kojeności
- wartościami `N` losowymi wartościami

Przetestować.

Zadanie 3.

Dodać do projektu funkcje:

- `bool is_sorted(const std::vector<T>& data)` -- sprawdza czy tablica jest posortowana
- `bool is_sorted_reverse(const std::vector<T>& data)` -- sprawdza czy tablica jest posortowana odwrotnie

Funkcje zwracają wartośc logiczną `bool` opisującą stan tablicy.

Przetestować.

Zadanie 4.

Dodać do projektu funckję

```
    void InsertionSort(std::vector<T>& data);
```

która sortuje tablicę za pomocą algorytmu sortowania przez wstawianie.

Przetestować.

Zadanie 5.

W bibliotece `<algorithms>` dostępna jest funkcja `std::sort`. Przykład użycia podany jest na `cppreference.com`. Użyć funkcji `std::sort` do posortowania `data`

Zadanie 6.

Zaprojektować i wykonać eksperymenty porównujący czas działania funkcji `std::sort` oraz `InsertionSort`.

Oczekiwanym wynikiem są odpowiedzi na pytania:

- która z funkcji działa szybciej dla małych zbiorów, znacznie mały okreslić empirycznie
- która z funkcji działa szybciej dla dużych zbiorów, znacznie duzy okreslić empirycznie
- która z funkcji działa szybciej dla zbiorów posortowanych
- która z funkcji działa szybciej dla zbiorów posortowanych w odwrotnej kolejności
- która z funkcji działa szybciej dla zbiorów posortowanych w odwrotnej kolejności
- która z funkcji działa szybciej dla zbiorów o losowych elementach
- która z funkcji działa szybciej dla zbiorów w których niewiele (<25%) elementów nie jest na swoich miejscach
- która z funkcji działa szybciej dla zbiorów w których dużo (>75%) elementów nie jest na swoich miejscach

---

Zadania bonusowe

Zadanie 1.

Napisać funkcje `InsertionSortDiag` oraz `MergeSortDiag`, które dodatkowo

- zliczają wykonane porównania
- zliczają wykonane przypisania
- zlicają inne informacje, które można uznać za istotne dla algorytmu

Przeprowadzić eksperymenty ilustrujace zachowanie się algorytmu w przypadku optymistycznym, średnim i pesymistycznym. Wyniki eksperymentów zapisać do pliku.

Zadanie 2.

Napisać funkcję sortującą listy jedno- i dwustronnie wiązane. Proste rozwiązanie polega na przekopiowaniu zawartości listy do tablicy. Czy da się to zrobić bez kopiowania?

Zadanie 3.

Describe a `Theta(n log n )`-time algorithm that, given a set `S` of `n` integers and another integer `x`, determines whether or not there exists two elements of `S` whose sum is exactly `x`.

Zadanie 4.

A numeric array of length `N` is given. We need to design a function that finds all positive numbers in the array that have their opposites in it as well. Describe approaches for solving optimal worst case and optimal average case performance, respectively.

Zadanie 5.

Udowodnić (odszukac dowód w literaturze), że każdy algorytm sortujący za pomocą porównań potrzebuje $O(n \log n)$ porównań w przypadku średnim.

Które z algorytmów podanych na wykładzie mają optymalną złożoność?