Zestaw 4

Proste algorytmy sortowania

Termin 17.12.2021

Zadanie 1. Implementacja prostych algorytmów sortowania

Zaimplementować poniższe proste algorytmy sortowania:

- Sortowanie bąbelkowe (`BubbleSort.cpp`, ang. *bubble sort*)
- Sortowanie przez wybieranie (`SelectionSort.cpp`, ang. *selection sort*)
- Sortowanie przez wstawianie (`InsertionSort.cpp`, ang. *insertion sort*)

Napisać szablon funkcji sortujący wektor `v`

```
template <class T> void sort(std::vector<T>& v);
```

Funkcja `main` wczytuje dane wejściowe ze **standardowego wejścia** do wektora typu `int` (`std::vector<int>`), sortuje elementy wektora wywołując funkcję `sort<int>()` i wypisuje elementy posortowanego wektora na **standardowe wyjście** przy użyciu pętli *for-each*:

```
for(const auto& i : v)
    std::cout << i << std::endl;
```

Na wejściu, w każdej linii znajduje się jedna liczba w postaci tekstowej. Na wyjściu, należy wypisać wczytane wartości w niemalejącej kolejności, po jednej liczbie w każdej linii. Proszę zapoznać się z dokumentem *Wskazówki i elementy języka C++* (wczytywanie danych, poprawa wydajności, pomiar czasu wykonania).

Czas sortowania

Zmierzyć czas działania wszystkich powyższych implementacji dla różnych rozmiarów danych wejściowych (mierzony czas powinien obejmować zakres od kilku sekund do kilku minut) i porównać wyniki.

Wyniki należy przedstawić na wykresie (w formie graficznej, np. używając programów `gnuplot`, `gnumeric`, `LibreOffice`), wykres powinien przedstawiać czas sortowania w funkcji rozmiaru danych. Dopasować zależność teoretyczną czasu wykonania.

Zadanie 2. Złożoność obliczeniowa

Zmodyfikować jedną wybraną implementację i dodać instrukcje zliczające dominujące operacje.

Obliczyć złożoność obliczeniową pesymistyczną, średnią i optymistyczną oraz miarę wrażliwości pesymistycznej. Sprawdzić dla losowych danych i tablicy posortowanej, czy liczba wykonanych operacji dominujących zgadza się oszacowaniami.

Zadanie 3. Generator (`Generator.cpp`)

Proszę napisać program, który przyjmuje dwa argumenty z linii komend - odpowiednio `n` i `max` - i wypisuje `n` (pseudo)**losowych liczb** z zakresu od `1` do `max` włącznie na standardowe wyjście.

Przykład:

```
./generator.x 10000 100000 > input_10k.txt
```

Zadanie dodatkowe A1 (2 pkt). Sortowanie przez zliczanie (`CountingSort.cpp`)

Zaimplementować sortowanie przez zliczanie (ang. *counting sort*). Założyć, że liczby są nieujemne i mniejsze od 10^6.

Zadanie dodatkowe A2 (1 pkt).

Zmodyfikować implementację algorytmu sortowania przez wybieranie tak, aby jednocześnie wybierać **minimum** i **maksimum** (`SelectionMinmax.cpp`).

Zadanie dodatkowe A3 (1 pkt).

Zamiast szablonu funkcji postaci

```
template <class T> void sort(std::vector<T>& v);
```

zaimplementować szablon funkcji z identycznym interfejsem jak funkcja biblioteczna `std::sort`,

```
template <typename RandomAccessIterator>
void sort(RandomAccessIterator begin, RandomAccessIterator end);
```

Andrzej Görlich
a.goerlich@outlook.com