Algorytmy i Struktury Danych
Sorting Algorithms

Vladyslav Slovinskyi

10 sierpnia 2022

Zawartość:

Katalog SortingAlgorithms zawiera implementacje poniższych prostych algorytmów sortowania:

- Sortowanie bąbelkowe;
- Sortowanie przez wybieranie;
- Sortowanie przez wstawianie;

I. Zestaw05:

1) `BubbleSort.cpp` - implementacja algorytmu sortowania bąbelkowego;
2) `SelectionSort.cpp` - implementacja algorytmu sortowania przez wybieranie;
3) `InsertionSort.cpp` - implementacja algorytmu sortowania przez wstawianie;
4) `ModifiedBubbleSort.cpp` - zmodyfikowana implementacja algorytmu sortowania bąbelkowego;

Jak uruchomic programy:

Katalog zawiera program Makefile do kompilacji powyzszych programow, a takze czyszczenia katalogu.

- Aby skompilowac projekt należy wykonać: `$ make`;
- Aby wyczyścić zawartośc katalogu (usunąć zbędne pliki), należy wykonać: `$ make clean`;

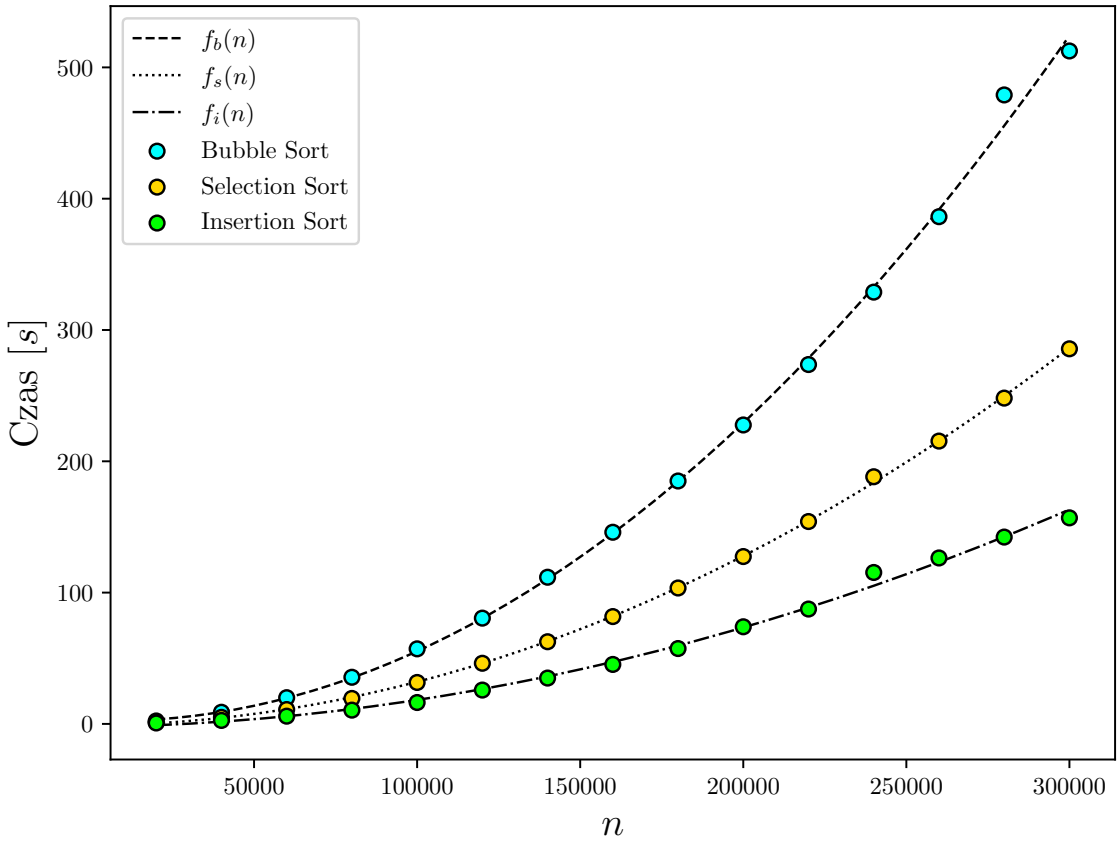
Czas sortowania

Za pomocą odpowiednich instrukcji został zmierzony czas działania wszystkich powyższych implementacji dla różnych rozmiarów danych wejściowych. Do pomiarów zostały dopasowane

zależności teoretyczne czasu wykonania. Złożoności pesymistyczne wymienionych wyżej prostych algorytmów sortowania wynoszą $O(n^2)$, w związku z czym dopasowywaliśmy zależności postaci:

$$f_{\langle x \rangle}(n) = an^2 + bn + c, \tag{1}$$

gdzie $\langle x \rangle \in \{b, s, i\}$, b – sortowanie bąbelkowe, s – sortowanie przez wybieranie, i – sortowanie przez wstawianie, n – rozmiar danych wejściowych. Pomiary oraz parametry dopasowania reprezentują rysunek 1 oraz tabela 1.



Rysunek 1: Wykres czasu działania wybranych implementacji prostych algorytmów sortowania (w sekundach) w funkcji rozmiaru danych wejściowych.

	$a \pm \Delta a$	$b \pm \Delta b$	$c \pm \Delta c$
BubbleSort	$6.05 \times 10^{-9} \pm 3.11 \times 10^{-10}$	$-7.68 \times 10^{-5} \pm 1.02 \times 10^{-4}$	2.51 ± 7.12
SelectionSort	$3.13 \times 10^{-9} \pm 5.67 \times 10^{-11}$	$1.89 \times 10^{-5} \pm 1.87 \times 10^{-5}$	-1.18 ± 1.30
InsertionSort	$1.73 \times 10^{-9} \pm 1.48 \times 10^{-10}$	$3.33 \times 10^{-5} \pm 4.87 \times 10^{-5}$	-2.39 ± 3.38

Tabela 1: Parametry dopasowania funkcji (1) do danych pomiarowych.

Złożoność obliczeniowa

Zmodyfikowaliśmy implementację algorytmu sortowania bąbelkowego oraz dodaliśmy instrukcje zliczające operacje dominujące (w przypadku prostych algorytmów sortowania są to operacje porównania).

```
1  #include <iostream>
2  #include <vector>
3  #include <chrono>
4
5  template<class T>
6  void swap(T* x, T* y) {
7      T tmp = *x;
8      *x = *y;
9      *y = tmp;
10 }
11
12 template<class T>
13 void sort(std::vector<T>& v) {
14     int i, j, flag = 0;
15     for (i = 0; i < v.size() - 1; i++) {
16         for (j = v.size() - 1; j > i; j--) {
17             if (v[j] < v[j - 1]) {
18                 swap(&v[j], &v[j - 1]);
19                 flag = 1;
20             }
21         }
22         if (flag == 0) {
23             break;
24         }
25     }
26     for(const auto& i : v) {
27         std::cout << i << std::endl;
28     }
29 }
```

Implementację algorytmu sortowania reprezentuje powyższy rysunek. Na początek zajmiemy się funkcją `void swap(T* x, T* y)`. Złożoność obliczeniowa pesymistyczna instrukcji (7) – (9) wynosi $O(1)$. Z reguły sumowania złożoność funkcji `void swap(T* x, T* y)` wynosi:

$$O(\max(1,1,1)) = O(1).$$

Sprawdzenie warunku (17) jest $O(1)$. Czyli złożoność obliczeniowa instrukcji (17) oraz (18) jest $O(1)$. Z tego wynika, że wnętrze pętli (16) jest $O(1)$, a więc złożoność fragmentu (16) – (18) wynosi $O(n-1-i)$. Z kolei pętla (15) wykonuje się $n-1$ razy, w związku z czym złożoność obliczeniowa pesymistyczna całego algorytmu możemy obliczyć w następujący sposób:

$$W(n) = \sum_{i=0}^{n-1}(n-1-i) = n-1+n-2+\ldots+1 = \frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2}.$$

W notacji „duże O” mamy:

$$W(n) = O\left(\frac{n^2}{2} - \frac{n}{2}\right) = O\left(\frac{n^2}{2}\right) = O(n^2).$$

Gdy mamy do czynienia z wektorem posortowanym (przypadek optymistyczny), wtedy złożoność obliczeniowa wynosi $n-1$. W notacji „duże O” mamy:

$$B(n) = O(n-1) = O(n).$$

W celu określenia złożoności średniej (oczekiwanej) należy uśrednić ilość wszystkich możliwych zamian:

$$\frac{1+2+3\ldots(n-1-i)}{n-i} = \frac{n-i-1}{2}.$$

Po uwzględnieniu zewnętrznej pętli `for` (15):

$$A(n) = \sum_{i=0}^{n-1} \frac{n-1-i}{2} = \frac{1}{2}\sum_{i=0}^{n-1}(n-1) - \frac{1}{2}\sum_{i=0}^{n-1} i = \frac{n(n-1)}{2} - \frac{n(n-1)}{4} = \frac{n(n-1)}{4}.$$

W notacji „duże O” mamy:

$$A(n) = O\left(\frac{n(n-1)}{4}\right) = O\left(\frac{n^2}{4} - \frac{n}{4}\right) = O(n^2).$$

Miarę wrażliwości pesymistycznej algorytmu sortowania bąbelkowego określamy następująco:

$$\Delta(n) = \frac{n(n-1)}{2} - (n-1) = \frac{n^2-3n+2}{2},$$

z kolei w notacji „duże O” mamy $O(n^2)$. Poniższa tabela reprezentuje podsumowanie powyższych rozważań.

		$T(n)$	duże O
Złożoność pesymistyczna	$W(n)$	$\frac{1}{2}n(n-1)$	$O(n^2)$
Złożoność optymistyczna	$B(n)$	$n-1$	$O(n)$
Złożoność średnia	$A(n)$	$\frac{1}{4}n(n-1)$	$O(n^2)$
Miara złożoności pesymistycznej	$\Delta(n)$	$\frac{1}{2}(n^2-3n+2)$	$O(n^2)$

Tabela 2: Złożoność czasowa algorytmu sortowania bąbelkowego.

Liczba wykonanych operacji dominujących zgadza się z oszacowaniami, czyli dla losowych danych liczba operacji dominujących wynosi $\frac{1}{2}n(n-1)$, z kolei dla tablicy posortowanej – $(n-1)$.