AiSD

Rafał Włodarczyk

INA 4, 2025

Contents

1 Lecture I - Losowe sortowanie **1**

1.1 Worst-case analysis 1

1.2 Average-case analysis 1

1.3 Analiza losowego sortowania 1

1.4 Insertion Sort (A, n) 1

1.4.1 Worst-case analysis - Insertion Sort (A, n) 2

1.4.2 Average-case analysis - Insertion Sort (A, n) 3

1.5 Przykład złożoności 3

1 Lecture I - Losowe sortowanie

Definiujemy problem:

1. Input: $A = (a_1, \ldots, a_n), |A| = n$
2. Output: Permutacja tablicy wyjściowej $(a'_1, a'_2, \ldots, a'_n)$, takie że: $a'_1 \leqslant a'_2 \leqslant \cdots \leqslant a'_n$.

1.1 Worst-case analysis

$$T(n) = \max_{\text{wszystkie wejścia}} \{\#\text{operacji po wszystkich |n|-wejściach}\} \tag{1}$$

1.2 Average-case analysis

Zakładamy pewien rozkład prawdopodobieństwa na danych wejściowych. Z reguły myślimy o rozkładzie jednostajnym. Niech T - zmienna losowa liczby operacji wykonanych przez badany algorytm.

$$\mathbf{E}(T) - \text{wartość oczekiwana T} \tag{2}$$

Później możemy badać wariancję, oraz koncentrację.

1.3 Analiza losowego sortowania

Dla poprzedniego algorytmu zobaczmy, że: $n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left[\text{czyli } f(n) \sim g(n) \equiv \lim_{n\to\infty} \frac{f(n)}{g(n)} = 1\right]$. To jest tragiczna złożoność.

1.4 Insertion Sort (A, n)

$(A, n) = ((a_1, a_2, \ldots, a_n), n)$

```
for j = 2...n
{
    key = A[j]
    i=j-1
    while(i>0 && A[i]>key) {
        A[i+1] = A[i]
        i = i - 1
    }
    A[i+1] = key
}
```

Przykład: $A = (8, 2, 4, 9, 3, 6), n = 6$

- $8_i, 2_j, 4, 9, 3, 6 \quad j = 2, i = 1, key = 2$ while
- $2, 8_j, 4, 9, 3, 6$
- $2, 8_i, 4_j, 9, 3, 6 \quad j = 3, i = 2, key = 4$ while
- $2, 4, 8, 9, 3, 6$
- $2, 4, 8_i, 9_j, 3, 6 \quad j = 4, i = 3, key = 9$ no while
- $2, 4, 8, 9_i, 3_j, 6 \quad j = 5, i = 4, key = 3$ while
- $2, 3, 4, 8, 9, 6$
- $2, 3, 4, 8, 9_i, 6_j \quad j = 6, i = 5, key = 6$ while
- $2, 3, 4, 6, 8, 9$

```
| <= x | > x | x | ... |
| <= x | x | > x | ... |
```

Porównujemy element ze wszystkim co jest przed nim - wszystko przed j-tym elementem będzie posortowane. Insertion sort nie swapuje par elementów w tablicy, a przenosi tam gdzie jest jego miejsce.

1.4.1 Worst-case analysis - Insertion Sort (A, n)

Odwrotnie posortowana tablica powoduje najwięcej przesunięć. Ponieważ ustaliśmy że liczba operacji w while zależy od j, wtedy:

$$T(n) = \sum_{j=2}^{n} O(j-1) = \sum_{j=1}^{n-1} O(j) = O\left(\sum_{j=1}^{n-1} j\right) = \tag{3}$$

$$= O\left(\frac{1+n-1}{2} \cdot (n-1)\right) = O\left(\frac{(n-1)\cdot(n)}{2}\right) = O\left(\frac{n^2}{2}\right) = O(n^2) \tag{4}$$

c

1.4.2 Average-case analysis - Insertion Sort (A, n)

Policzmy dla uproszczenia, że na wejściu mamy n-elementowe permutacje, z których każda jest jednakowo prawdopodobna $p = \frac{1}{n!}$. Spróbujmy wyznaczyć $\mathbf{E}$, korzystając z inwersji permutacji. Wartość oczekiwana liczby inwersji w losowej permutacji wynosi:

$$\mathbf{E} \sim \frac{n^2}{4} \tag{5}$$

Pominęliśmy stałe wynikające z innych operacji niż porównywanie. W average-case będziemy około połowę szybiciej niż w worst-case.

Pseudokod bez przykładu jest słaby.

1.5 Przykład złożoności

Patrzymy na wiodący czynnik.

$$13n^2 + 91n\log n + 4n + 13^{10} = O(n^2) \tag{6}$$

$$= 13n^2 + O(n\log n) \tag{7}$$

Chcielibyśmy gdzie to konieczne, zapisać *lower order terms.*

Pytanie o dzielenie liczb - istnieją algorytmy, które ze względu na arytmetyczne właściwości liczb sprawiają, że mniejsze liczby mogą dzielić się dłużej niż większe. Podczas tego kursu nie omawiamy złożoności dla takich algorytmów.