Alicja i Bogdan w naleśnikarni

Dominik Lau

27 października 2022

Pełny algorytm

```
def sortuj(A[1.. .n],n):
   if A posortowane:
      return // gotowe

   if n == 1:
      return // gotowe

   m = znajdz_najwiekszy(A, n) // (1.)
   flip(A, 1,m) // (2.)
   flip(A, 1, n) // (3.)
   sortuj(a[2...n], n-1) // (4.)
```

A - tablica naleśników
n - ilość naleśników

(1.) m - indeks największego naleśnika
(2.) wywrocenie naleśników na stosie, palce wkładamy za m-tym naleśnikiem
(3.) wywrócenie całego stosu naleśników
(4.) dalej sortujemy stos n-1 naleśników

Najgorsze cztery naleśniki

$A = [2, 4, 3, 1]$
mamy 5 przełożeń:
$A = [2, 4, 3, 1] \to [4, 2, 3, 1] \to [1, 3, 2, 4] \to [3, 1, 2, 4] \to [2, 1, 3, 4] \to [1, 2, 3, 4]$

Dlaczego algorytm nie jest optymalny

przykładowa sytuacja
mamy $A = [5, 2, 3, 4, 1]$
Algorytm będzie działał tak
$A = [5, 2, 3, 4, 1] \to [1, 4, 3, 2, 5] \to [4, 1, 3, 2, 5] \to [2, 3, 1, 4, 5] \to [3, 2, 1, 4, 5] \to [1, 2, 3, 4, 5]$
czyli wykona 5 przewrotów, jest natomiast rozwiązanie wymagające 4 przewrotów
$A = [5, 2, 3, 4, 1] \to [1, 4, 3, 2, 5] \to [2, 3, 4, 1, 5] \to [4, 3, 2, 1, 5] \to [1, 2, 3, 4, 5]$

algorytm wykonuje zatem nieoptymalną ilość przewrotów
ogółem algorytm wykonuje w najgorszym przypadku (czyli takim, gdzie naleśniki zostaną posortowane dopiero przy ostatnim ruchu $P(n) = 2n - 2$ przewrotów

Złożoność obliczeniowa

rozmiar danych = ilość naleśników = n
sprawdzenie, czy dane są posortowane O(n)
przewrót k elementów = O(k) więc w najgorszym przypadku przewrót = O(n)
znalezienie największego również O(n)
zatem $T(n) = T(n-1) + O(n) = O(n^2)$