Algorytmy i struktury danych

Sortowanie

Aleksander Lamża
ZKSB · Instytut Informatyki
Uniwersytet Śląski w Katowicach

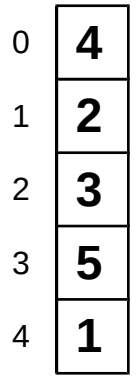
aleksander.lamza@us.edu.pl

Zawartość

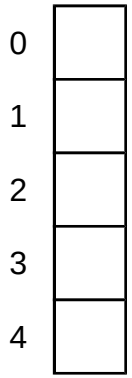
- Najbardziej toporny algorytm sortowania
- Sortowanie przez wstawianie
- Sortowanie przez wybieranie
- Sortowanie bąbelkowe

Najbardziej toporny algorytm sortowania

Mamy posortować taką tablicę (`tabN`):

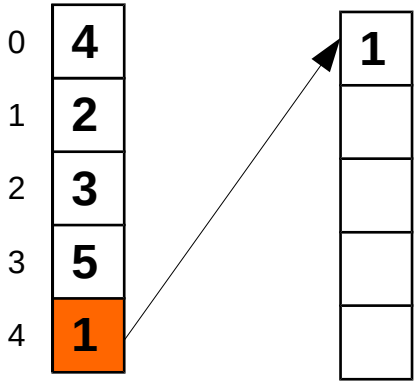


Tworzymy więc nową, pustą tablicę (`tabP`) o tej samej długości:

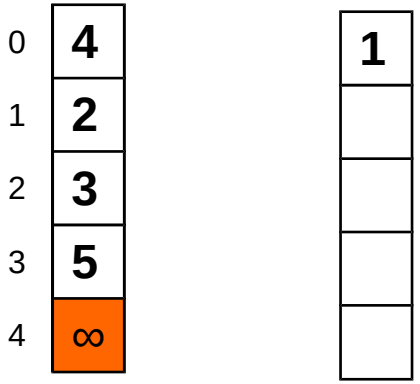


Najbardziej toporny algorytm sortowania

Następnie przeglądamy całą tablicę `tabN` i szukamy najmniejszego elementu. Kiedy go znajdziemy, wstawiamy w pierwszej dostępnej komórce tablicy `tabP`:

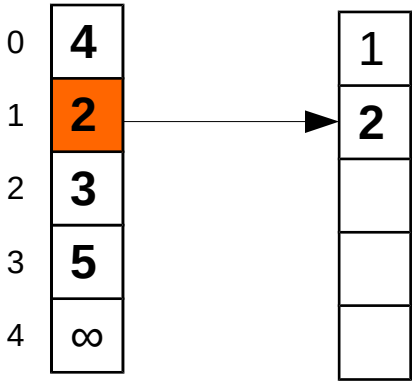


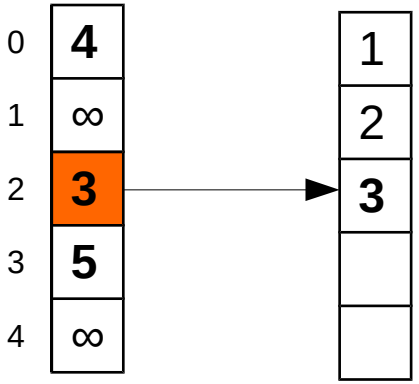
Następnie „zaznaczamy” w `tabN` miejsce, skąd wzięliśmy wartość:

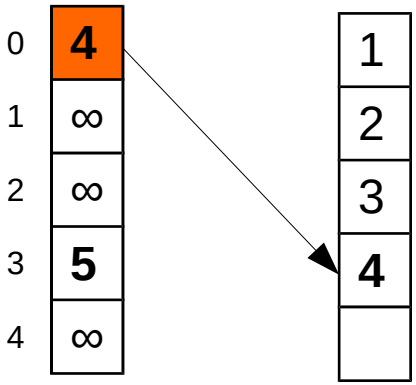


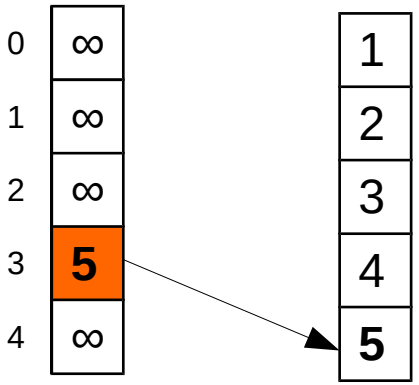
Najbardziej toporny algorytm sortowania

To samo powtarzamy dla kolejnych elementów tablicy `tabN`:









Najbardziej toporny algorytm sortowania

Jaka jest złożoność tego algorytmu?

Przeanalizujmy to na przykładzie tej pięcioelementowej tablicy.

Dla każdego elementu wstawianego do `tabP` trzeba przejrzeć całą tablicę `tabN` (a obie tablice mają taki sam rozmiar równy 5), więc:

$$T = 5 * 5$$

Mamy więc złożoność $\mathbf{O(N^2)}$.

Omawiany tu algorytm ma też dużą **złożoność pamięciową**,
ponieważ wymaga utworzenia drugiej tablicy o tym samym rozmiarze.

Sortowanie przez wstawianie

Algorytm przez wstawianie rozpoczyna się od sprawdzenia dwóch pierwszych elementów tablicy `tab`. Jeśli ich kolejność nie jest prawidłowa, zamieniamy je miejscami.
Kolejny – trzeci – element tablicy (`tab[2]`) jest wstawiany w odpowiednim miejscu. Jeśli jest mniejszy niż poprzednie elementy, musimy zrobić dla niego miejsce (czyli przesunąć większe elementy o jedną pozycję w prawo). W przeciwnym przypadku nie musimy zmieniać jego położenia.
To samo robimy dla kolejnych elementów.

Pseudokod tego algorytmu wygląda następująco:

```
for (i = 1; i < n; ++i)
   przesuń wszystkie elementy większe od tab[i] o jedną pozycję w prawo
   wstaw tab[i] w odpowiednim miejscu
```

www.sorting-algorithms.com/insertion-sort

Sortowanie przez wybieranie

Na początku jest wybierany najmniejszy element i zostaje on wymieniony z elementem znajdującym się na początku tablicy. Następnie jest odszukiwana najmniejszy z pozostałych elementów i jest wstawiany w kolejnej – drugiej – komórce tablicy. Powtarzamy te operacje dla wszystkich elementów do momentu, w którym wszystkie znajdą się w odpowiednich miejscach.

Pseudokod tego algorytmu wygląda następująco:

```
for (i = 0; i < n-1; ++i)
   wybierz najmniejszy spośród elementów tab[i]...tab[n-1]
   zamień wybrany element z tab[i]
```

www.sorting-algorithms.com/selection-sort

Sortowanie bąbelkowe

Wyobraźcie sobie tablicę jako naczynie wypełnione wodą z bąbelkami. Bąbelki to poszczególne elementy tablicy. Im lżejszy bąbelek (czyli element o mniejszej wartości), tym szybciej leci do góry.
Tablicę analizujemy od końca (dołu) do początku (góry) i zamieniamy ze sobą elementy, które zajmują nieprawidłowe miejsca.

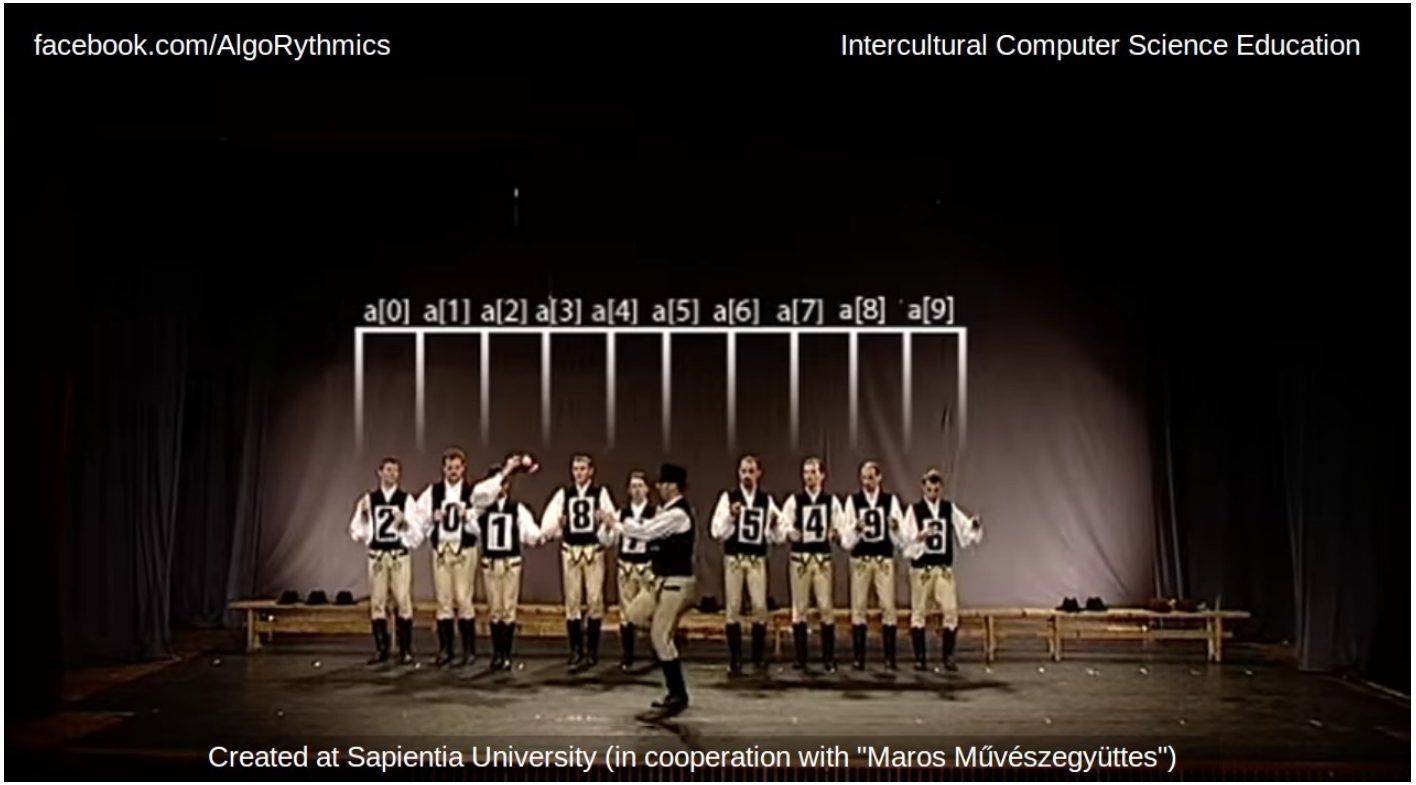
Pseudokod tego algorytmu wygląda następująco:

```
for (i = 0; i < n-1; ++i)
   for (j = n-1; j > i; --j)
      jeśli kolejność tab[j] i tab[j-1] jest nieprawidłowa, zamień je miejscami
```

www.sorting-algorithms.com/bubble-sort

Artystyczne wizje sortowania

W zrozumieniu algorytmów sortowania może Wam pomóc zespół AlgoRythmics:



http://www.youtube.com/user/AlgoRythmics

Zadanie

Waszym zadaniem jest zaimplementowanie omówionych algorytmów sortowania i sprawdzenie ich wydajności dla trzech przypadków:

- dane losowe,
- dane prawie uporządkowane,
- dane ułożone rosnąco,
- dane ułożone malejąco.

Musicie więc przygotować cztery funkcje inicjujące tablicę:

```
void wypelnijLosowo(int* tab)

void wypelnijPrawieRosnaco(int* tab)

void wypelnijMalejaco(int* tab)

void wypelnijRosnaco(int* tab)
```

Przyda się też funkcja wyświetlająca tablicę:

```
void wyswietl(int* tab)
```