Algorytmy i struktury danych

Proste algorytmy sortowania

Witold Marańda
maranda@dmcs.p.lodz.pl

1

Pojęcie sortowania

Sortowaniem nazywa się proces ustawiania zbioru obiektów w określonym porządku

Sortowanie stosuje się w celu ułatwienia późniejszego wyszukiwania elementów sortowanego zbioru (np. słowniki, spisy treści, książki telefoniczne, itp.)

Różnice w budowie algorytmów pozwalają podzielić metody sortowania na dwie klasy:

- sortowanie tablic – zwane sortowaniem wewnętrznym.
  Sortowanie wewnętrzne dotyczy sytuacji, gdy cały sortowany zbiór elementów (tablica) znajduje się w pamięci operacyjnej komputera i możliwy jest natychmiastowy dostęp do każdego elementu sortowanego zbioru.
- sortowanie plików – zwane sortowaniem zewnętrznym.
  Sortowanie zewnętrzne dotyczy sytuacji, gdy jedynie fragment sortowanego zbioru elementów (pliku) znajduje się w pamięci operacyjnej komputera i nie jest możliwy natychmiastowy dostęp do każdego elementu sortowanego zbioru

2

Pojęcie sortowania

Jeśli dany jest zbiór obiektów:

$$a_1, a_2, a_3, \ldots a_{1n}$$

(gdzie indeksy *1..n oznaczają położenie elementu w zbiorze)*

to sortowanie polega na permutowaniu (przestawianiu elementów w zbiorze), aż do momentu osiągnięcia uporządkowania:

$$a_{1(k)}, a_{2(k)}, a_{3(k)}, \ldots a_{1n(k)}$$

(gdzie indeksy *k oznacza liczbę wykonanych permutacji)*

takiego, że dla zadanej **funkcji porządkującej** f mamy:

$$f(a_{1(k)}),\ f(a_{2(k)}),\ f(a_{3(k)}),\ \ldots f(a_{1n(k)})$$

Wartość funkcji porządkującej nazywa się **kluczem sortowania**.

Wartości funkcji porządkującej (klucza) zwykle nie oblicza się, ale przechowuje w postaci jawnej jako składową każdego elementu sortowanego zbioru. Z tego powodu, najbardziej odpowiednią strukturą dla elementów zbioru jest struktura rekordu.

3

Klucz sortowania

W omawianych algorytmach sortowania używane będą obiekty następującego typu:

```
type obiekt = record
                klucz : integer;
                { pola danych obiektu }
              end
```

Pole rekordu klucz jest cechą pozwalającą na sortowanie obiektów, których właściwe dane znajdują się w pozostałych polach rekordu.

Przykłady obiektów do sortowania:

nazwiska osób sortowane według (klucza) wzrostu:

```
type obiekt = record
   wzrost : integer;
   nazwisko : array[1..20] of char;
end
```

nazwiska osób sortowane według (klucza) numer telefonu:

```
type obiekt = record
   telefon : integer;
   nazwisko : array[1..20] of char;
end
```

4

Efektywność metod sortowania

Ze względu na oszczędność miejsca, elementy sortuje się w tablicy, w której się aktualnie znajdują tj. *in situ*. Sortowanie składa się zasadniczo z dwóch rodzajów operacji:

- porównań elementów
- przesunięć elementów

Liczby porównań i przesunięć koniecznych do posortowania zbioru są dobrą miarą efektywności metod sortowania. Liczbę porównań koniecznych do posortowania zbioru oznacza się symbolem *Po*, a liczbę koniecznych przesunięć *Pr*.

Dla zbioru zawierającego *n*-elementów:

- dobre metody sortowania mają efektywność, tj liczbę *Po* i *Pr* rzędu **n*log(n),**
- proste metody sortowania mają efektywność, tj liczbę *Po* i *Pr* rzędu $\mathbf{n^2}$**.**

5

Proste metody sortowania

Proste metody sortowania, choć mało efektywne mają kilka zalet:

- są krótkie i łatwe do zrozumienia i dobrze ilustrują główne zasady sortowania,
- są dostatecznie szybkie dla małych zbiorów elementów.

We wszystkich omawianych metodach sortowana będzie tablica A, składająca się z n-elementów typu obiekt, zdefiniowanego wcześniej:

```
var A: array[1..n] of obiekt;
```

Zakłada się, że tablica A i stała n są dostępne (i nie są deklarowane) w procedurach realizujących sortowanie we wszystkich dalszych przykładach.

6

Sortowanie przez wstawianie

Metoda powszechnie stosowana przez grających w karty...

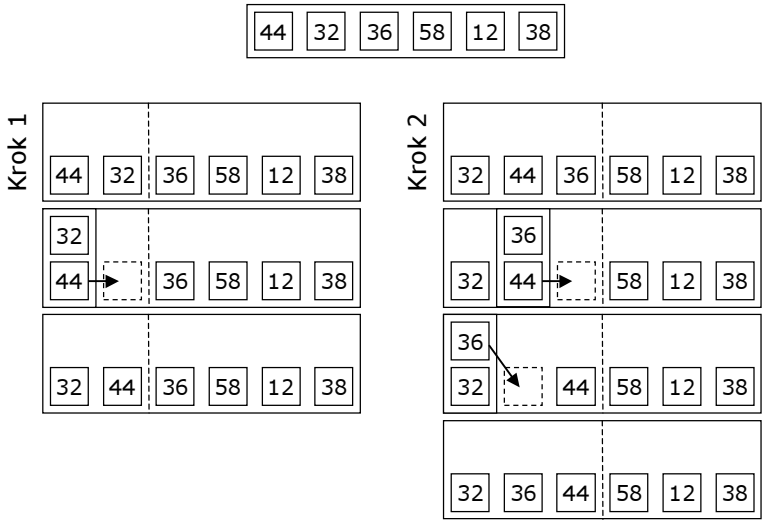
W każdym kroku (iteracji) metody, poczynając od elementu 2 do elementu *n*, porównuje się *i*-ty element z elementami poprzednimi i jeśli mają one większe klucze, przesuwa się je w prawo, robiąc miejsce na bieżący element.

W każdej iteracji, przesuwanie elementów kończy się z chwilą znalezienia elementu z kluczem mniejszym niż klucz danego elementu. Jeśli takiego elementu nie ma, to przesuwanie należy zakończyć po osiągnięciu „lewego" brzegu zbioru.

Efektywność metody: Po, Pr ~ $\mathbf{n^2}$

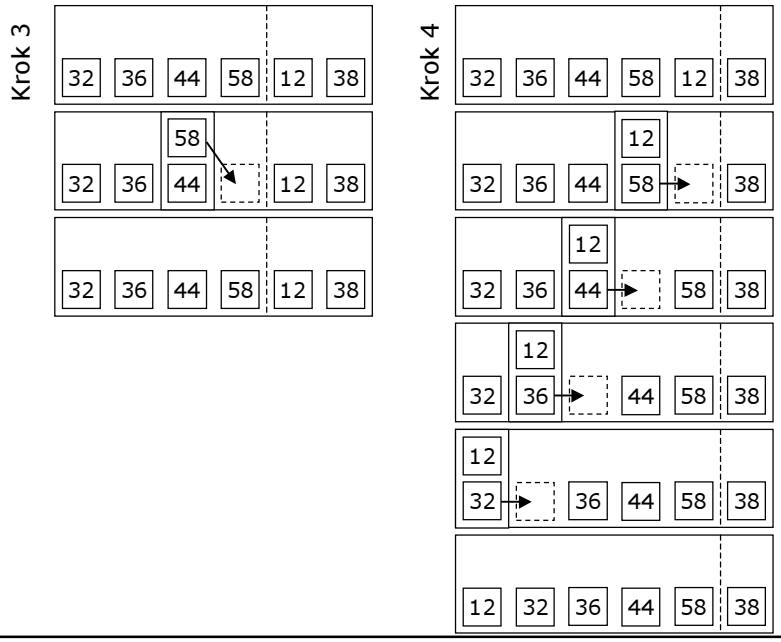
7

Sortowanie przez wstawianie – przykład 1/3



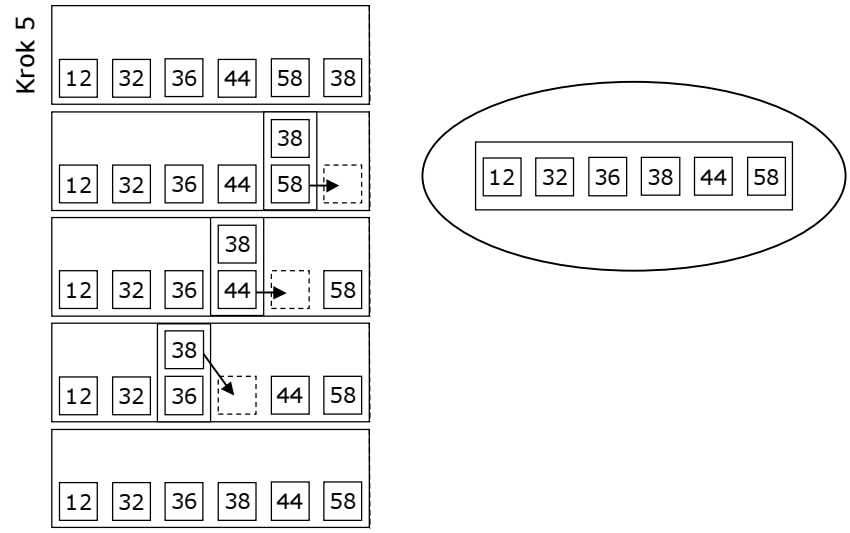
8

Sortowanie przez wstawianie – przykład 2/3



9

Sortowanie przez wstawianie – przykład 3/3



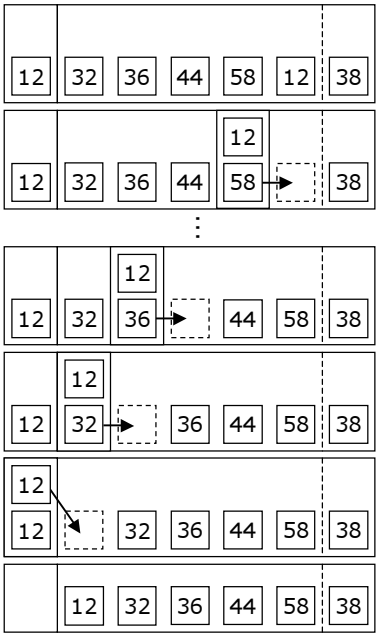
10

Sortowanie przez wstawianie – przykładowy program

```
PROCEDURE sort_wstawianie;
VAR
        i,j : integer;
        x : obiekt;
BEGIN
        FOR i:=2 TO n DO
        BEGIN
                x:=A[i];
                j:=i-1;
                WHILE (j>0) AND (x.klucz<A[j].klucz) DO
                BEGIN
                        A[j+1]:=A[j];
                        j:=j-1;
                END;
        A[j+1]:=x;
        END
END;
```

11

Sortowanie przez wstawianie (z wartownikiem)



Aby uniknąć wielokrotnego sprawdzania warunku `(j>0)` w pętli `while` (sprawdzanie osiągnięcia „lewego” brzegu zbioru) można umieścić kopię elementu x w tablicy A na pozycji 0, czyli tzw. wartownika.

```
PROCEDURE sort_wstawianie;
VAR
       i,j : integer;
       x : obiekt;
BEGIN
       FOR i:=2 TO n DO
       BEGIN
          x:=A[i];
          A[0]:=x;
          j:=i-1;
          WHILE x.klucz<A[j].klucz DO
          BEGIN
             A[j+1]:=A[j];
             j:=j-1;
          END;
       A[j+1]:=x;
       END
END;
```

Sortowanie przez wybieranie

W każdym kroku (iteracji) metody wyszukuje się najmniejszy element i zamienia go się z elementem na pierwszej pozycji. Operację tę powtarza się następnie dla *n-1* elementów i zamienia się element na drugiej pozycji. Iteracje takie powtarza się, aż zostanie tylko jeden element – o największym kluczu. Wykonuje się więc *n-1* iteracji.

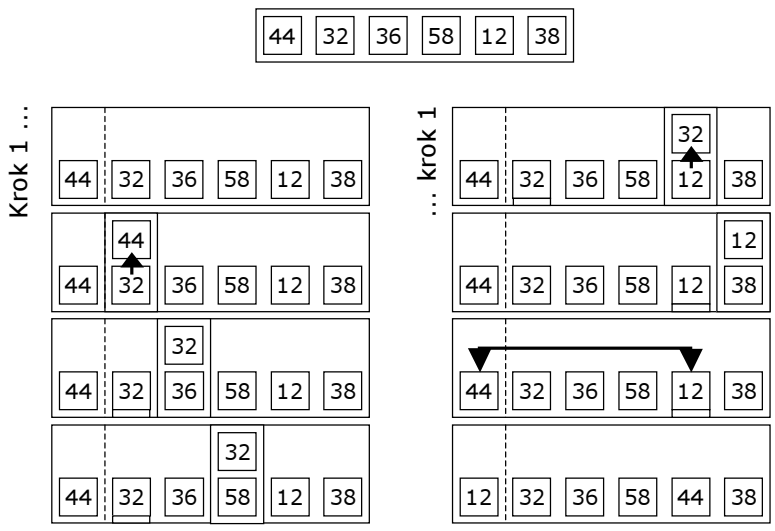
Metoda sortowania przez wybieranie jest w pewnym sensie odwrotna do metody sortowania przez wstawianie:

- w metodzie wstawiania w każdym kroku rozważa się tylko *jeden*, kolejny element *ciągu źródłowego* i wszystkie elementy *tablicy wynikowej* aby znaleźć miejsce wstawienia nowego elementu,
- w metodzie wybierania rozważa się *wszystkie* elementy *tablicy źródłowej*, aby znaleźć element z najmniejszym kluczem i ustawić go jako kolejny element *ciągu wynikowego*.

Efektywność metody: Po ~ $\mathbf{n^2}$, Pr ~ n*log(n)

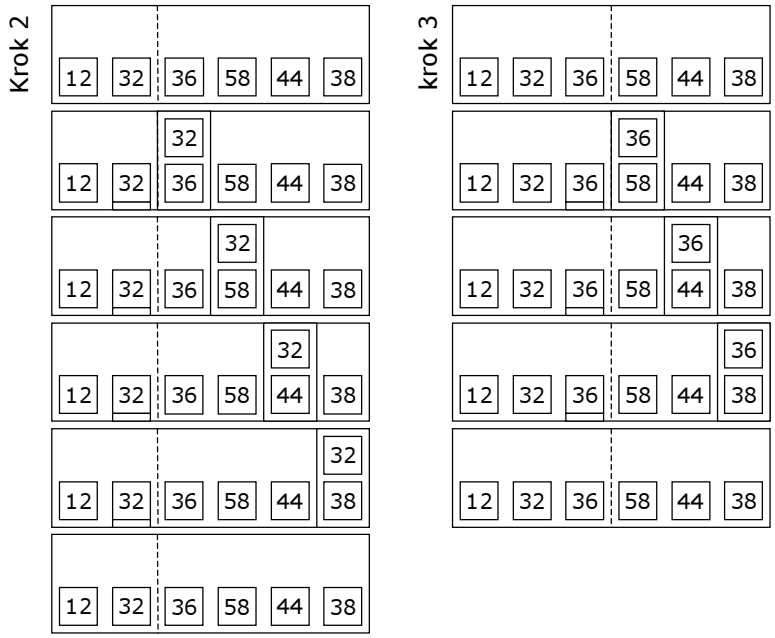
13

Sortowanie przez wybieranie – przykład 1/3



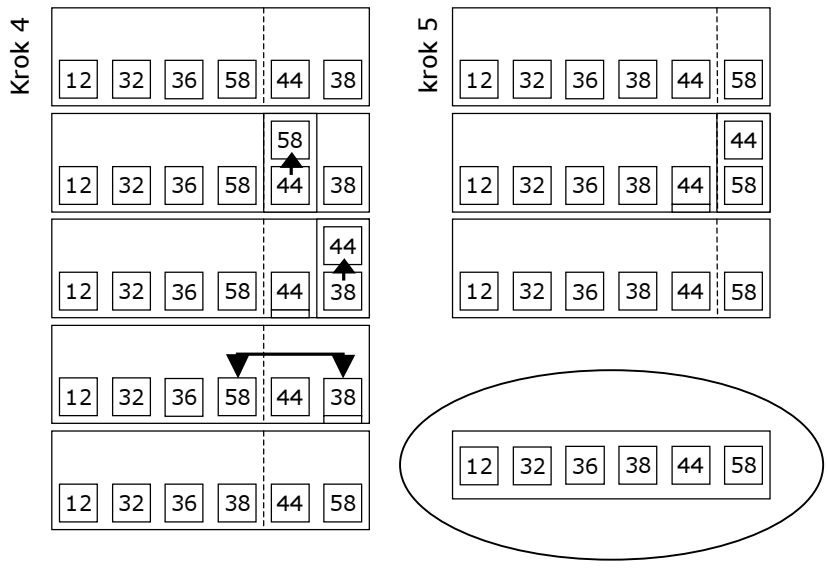
14

Sortowanie przez wybieranie – przykład 2/3



15

Sortowanie przez wybieranie – przykład 3/3



16

Sortowanie przez wybieranie – przykładowy program

```
PROCEDURE sort_wybieranie;
VAR
   i, j, k: integer;
   x: obiekt;
BEGIN
   FOR i:=1 TO n-1 DO
   BEGIN
      k:=i;
      x:=A[i];
      FOR j:=i+1 TO n DO
         IF A[j].klucz<x.klucz THEN
         BEGIN
            k:=j;
            x:=A[j];
         END;
      A[k]:=A[i];
      A[i]:=x;
   END
END;
```

17

Sortowanie przez wybieranie

Efektywność metody: Po ~ $\mathbf{n^2}$, Pr ~ n*log(n)

Metoda sortowania przez wybieranie jest zwykle nieco lepsza niż sortowanie przez wstawianie, ze względu na mniejszą liczbę przesunięć *Pr*. Jest to szczególnie istotne, gdyż zwykle operacja przesunięcia jest bardziej czasochłonna od operacji porównania, która dotyczy tylko kluczy, a nie wszystkich pól sortowanych elementów.

Jednakże w przypadkach, gdy kluczą są juz posortowane lub prawie posortowane, metoda sortowania przez wstawianie okazuje się szybsza.

18

Sortowanie przez zamianę (*sortowanie bąbelkowe)*

W każdym kroku (iteracji) metody porównuje się klucze wszystkich sąsiadujących elementów i jeśli trzeba zamienia się elementy miejscami (gdy ich kolejność nie jest zgodna z kluczem). W ten sposób po *n-1* przejściach przez zbiór dochodzi się do stanu uporządkowania.

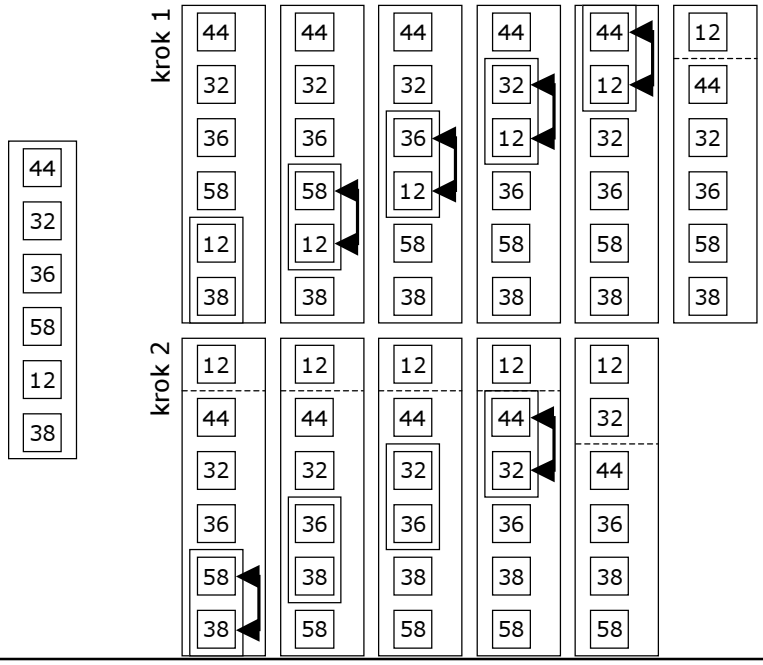
Popularna nazwa metody „sortowanie bąbelkowe" wynika z podobieństwa przemieszczania się elementów „w górę" tabeli, jak bąbelków powietrza w wodzie (jeśli tabela jest ustawiona tak jak w przykładach, tj. kolejne permutacje zbioru są kolumnami.

Efektywność metody: Po, Pr ~ $\mathbf{n^2}$

Sortowanie bąbelkowe jest najmniej efektywne ze wszystkich omawianych metod, lecz ma bardzo prostą implementację. Jej stosowanie ma sens tylko gdy sortowany zbiór jest bardzo mały.

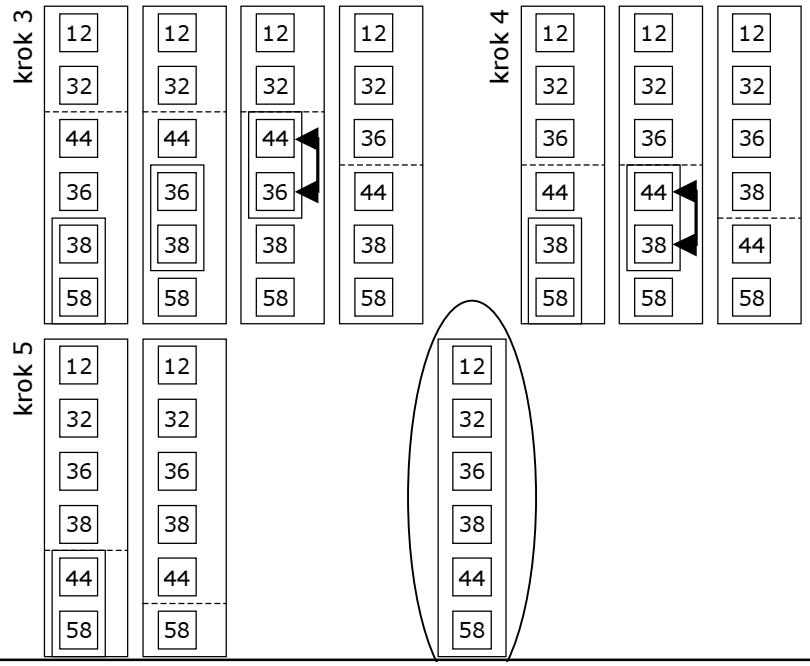
19

Sortowanie przez zamianę – przykład 1/2



20

Sortowanie przez zamianę – przykład 2/2



21

Sortowanie przez zamianę – przykładowy program

```
PROCEDURE sort_zamiana;
VAR
   i, j: integer;
   x: obiekt;
BEGIN
   FOR i:=2 TO n DO
      FOR j:=n DOWNTO i DO
         IF A[j-1].klucz>A[j].klucz THEN
         BEGIN
            x:=A[j-1];
            A[j-1]:=A[j];
            A[j]:=x;
         END
END;
```

22