12. Metody sortowania prostego

Sortowanie to jeden z podstawowych problemów informatycznych. Jest wykorzystywane np. przy wyszukiwaniu elementu o określonych własnościach czy kompresji danych. Istnieje wiele różnych algorytmów sortowania o różnej efektywności. Chociaż współczesne oprogramowanie (w tym biblioteki języków programowania) dostarcza nam gotowe narzędzia do porządkowania, to zrozumienie pojęcia sortowania jest często uważane za pierwszy krok do opanowania algorytmiki. W tym temacie zajmiesz się metodami sortowania prostego.

Cele lekcji

- Poznasz metody sortowania prostego: sortowanie bąbelkowe (przez prostą zamianę) oraz sortowanie przez wstawianie.
- Przypomnisz sobie algorytm sortowania przez wybieranie poznany w szkole podstawowej.
- Porównasz metody sortowania prostego.
- Oszacujesz czasową złożoność obliczeniową algorytmów sortowania prostego.

Sortowanie

Przypomnijmy, że **sortowanie** to porządkowanie danych według określonych kryteriów. Porządkować można dowolny ciąg danych tego samego typu, np. liczby rosnąco, słowa alfabetycznie, dane osobowe według daty urodzenia. Musi być tylko jednoznacznie określony sposób porównania dwóch elementów danego typu.

Przykłady w tym temacie będą dotyczyć liczb całkowitych. Omówimy różne metody sortowania i utworzymy funkcje sortujące dane.

12.1. Sformułowanie problemu sortowania

W programach, które napiszemy, danymi wejściowymi będzie ciąg n losowych liczb całkowitych: $a_0, a_1, ..., a_{n-1}$, natomiast wynikiem – elementy tego ciągu uporządkowane niemalejąco. Liczby zapamiętamy w tablicy.

Oto specyfikacja problemu:

Specyfikacja

Dane: `A[0..n-1]` – tablica n liczb całkowitych.

Wynik: tablica `A` z uporządkowanymi elementami, `A[0]` ⩽ `A[1]` ⩽ ... ⩽ `A[n-1]`.

Funkcja `main` programu sortującego liczby (niezależnie od zastosowanego algorytmu sortowania) powinna uwzględniać: wylosowanie i wypisanie tablicy liczb, posortowanie elementów tablicy, wypisanie posortowanej tablicy liczb.

Do wylosowania tablicy liczb całkowitych użyjemy funkcji `Losuj`, natomiast do wypisania elementów tablicy – funkcji `Wypisz`. Kod źródłowy funkcji losującej oraz funkcji wypisującej elementy tablicy może wyglądać tak samo jak w **programach wyszukujących element w tablicy**. Rozmiar tablicy określimy za pomocą stałej `N`, którą zadeklarujemy przed funkcjami.

Fragment kodu źródłowego programu wyszukującego element w tablicy, s. 152

Funkcja `main` programu sortującego niemalejąco tablicę liczb może wyglądać następująco:

Fragment kodu źródłowego programu sortującego niemalejąco tablicę liczb – funkcja `main`

```
1.  int main()
2.  {
3.      int A[N];
4.      srand(time(NULL));
5.      Losuj(A);
6.      Wypisz(A);
7.      Sortuj(A);
8.      Wypisz(A);
9.      return 0;
10. }
```

W linii 3 jest deklarowana tablica `N` liczb całkowitych, a w linii 4 inicjowany generator liczb losowych. W funkcji `main` użyliśmy funkcji `Sortuj`, która będzie sortować tablicę liczb zgodnie ze specyfikacją problemu. Należy pamiętać o dodaniu w programie bibliotek `cstdlib` i `ctime` (w pierwszej jest dostępna funkcja `srand`, a w drugiej – funkcja `time`) oraz definicji stałej `N` określającej rozmiar tablicy.

12.2. Sortowanie bąbelkowe (przez prostą zamianę)

Sortowanie bąbelkowe (przez prostą zamianę)

Najprostszą metodą sortowania jest **sortowanie bąbelkowe**, nazywane też **sortowaniem przez prostą zamianę**. W metodzie tej porównywane są sąsiadujące ze sobą elementy. Jeżeli są ustawione w niewłaściwym porządku, to są zamieniane miejscami – stąd nazwa „przez prostą zamianę". Na przykład jeżeli chcemy posortować zbiór niemalejąco, a pierwszy z porównywanych elementów jest większy od drugiego – zamieniamy je miejscami.

Nazwa „sortowanie bąbelkowe" powstała przez analogię do zjawiska wypływania na powierzchnię wody bańki powietrza: bańka o największej objętości wypływa jako pierwsza. Podczas sortowania największy element porządkowanego ciągu również „wypływa" jako pierwszy (jako pierwszy jest ustawiany na właściwym miejscu).

Algorytm sortowania bąbelkowego

Porządkowanie elementów tą metodą pokażemy na przykładzie pięcioelementowej tablicy, w której na kolejnych pozycjach znajdują się liczby: 6, 4, 9, 1, 5. Elementy uporządkujemy niemalejąco. Pierwszy krok ilustruje, jak umieścić właściwy element na ostatniej pozycji. W kolejnych krokach postępujemy analogicznie.

Krok 1

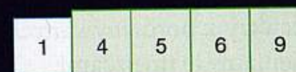
- Porównujemy pierwszy element z drugim. 6 > 4, więc zamieniamy liczby miejscami.
- Porównujemy drugi element z trzecim. 6 < 9, więc nie zamieniamy liczb miejscami.
- Porównujemy trzeci element z czwartym. 9 > 1, więc zamieniamy liczby miejscami.
- Porównujemy czwarty element z piątym. 9 > 5, więc zamieniamy liczby miejscami.
- W wyniku czterech porównań największy element tablicy (liczba 9) znalazł się na ostatniej pozycji.

W drugim kroku umieścimy właściwy element na przedostatniej pozycji. Przy porównaniach pomijamy ostatni element tablicy, ponieważ jest już na właściwym miejscu. W trzecim kroku przy porównaniach pomijamy dwa ostatnie elementy tablicy, w czwartym – trzy ostatnie itd. Poniżej przedstawione są wyniki otrzymane w kolejnych krokach.

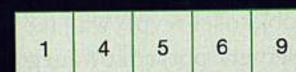
Krok 2 Porównujemy pary liczb: 4 i 6, 6 i 1 oraz 6 i 5. Liczba 6 znalazła się na właściwej, przedostatniej pozycji.

Krok 3 Porównujemy pary liczb 4 i 1 oraz 4 i 5. W wyniku tych operacji liczba 5 jest na właściwej pozycji.

Krok 4 Porównujemy parę liczb 1 i 4. Znalezienie drugiego elementu ciągu wskazuje jednocześnie najmniejszy element ciągu.



Wynikiem działania algorytmu jest tablica:



Oto zapis algorytmu porządkującego elementy tablicy niemalejąco metodą sortowania bąbelkowego:

```
dla i ← 1, 2, ..., n - 1 wykonuj
    dla j ← 0, 1, ..., n - i - 1 wykonuj
        jeśli A[j] > A[j+1] to
            pom ← A[j]
            A[j] ← A[j+1]
            A[j+1] ← pom
```

Dobra rada
Aby uporządkować elementy nierosnąco, zmień w instrukcji warunkowej znak > na <.

Zwróć uwagę na zakres zmiennych sterujących i oraz j poszczególnych pętli. Jedno wykonanie zewnętrznej pętli wyznacza największy element badanego ciągu – „wypłynięcie jednego bąbelka". Ciągu jednoelementowego nie musimy sortować, dlatego liczba powtórzeń zewnętrznej pętli jest o jeden mniejsza od liczby elementów tablicy. Liczba powtórzeń wewnętrznej pętli jest zależna od wartości zmiennej sterującej zewnętrznej pętli. W kolejnych iteracjach zewnętrznej pętli liczba powtórzeń wewnętrznej pętli zmniejsza się o jeden. Do zamiany miejscami dwóch liczb wykorzystaliśmy zmienną pomocniczą pom.

Kod źródłowy funkcji Sortuj realizującej algorytm sortowania bąbelkowego zapisany w pseudokodzie może być następujący:

```
1.  void Sortuj(int A[])
2.  {
3.      int i, j, pom;
4.      for (i=1;i<N;i++)
5.          for (j=0;j<N-i;j++)
6.              if (A[j]>A[j+1])
7.              {
8.                  pom=A[j];
9.                  A[j]=A[j+1];
10.                 A[j+1]=pom;
11.             }
12. }
```

Fragment kodu źródłowego programu sortującego niemalejąco tablicę liczb – funkcja realizująca algorytm sortowania bąbelkowego

Dobra rada
Do zamiany wartości dwóch sąsiednich elementów tablicy możesz wykorzystać funkcję **swap**. Instrukcja z użyciem tej funkcji wyglądałaby następująco:
`swap(A[j],A[j+1]);`

Zwróć uwagę, że jeśli w jednej iteracji wewnętrznej pętli nie zostanie dokonana żadna zamiana liczb, to tablica jest już posortowana. Można więc zmodyfikować algorytm tak, aby kończył działanie, kiedy nie dokona żadnej zamiany. W praktyce nie stosuje się tego rozwiązania, ponieważ należy wykonywać dodatkowe porównanie przy każdej iteracji zewnętrznej pętli, a szanse na to, że zamiany zostaną zakończone znacząco wcześniej, są niewielkie.

Ćwiczenie 1

Napisz i przetestuj program sortujący niemalejąco tablicę liczb całkowitych algorytmem sortowania bąbelkowego.

12.3. Sortowanie przez wybieranie

Sortowanie przez wybieranie

Sortowanie przez wybieranie polega na wyszukaniu elementu, który powinien się znaleźć na danej pozycji, i zamianie miejscami tego elementu z elementem znajdującym się dotychczas na tej pozycji.

Jeśli np. chcemy posortować tablicę niemalejąco, w pierwszej iteracji poszukujemy najmniejszej liczby w całej tablicy i zamieniamy ją miejscami z pierwszym elementem tablicy. W drugiej iteracji poszukujemy liczby, która powinna się znaleźć w tablicy jako druga. Pomijamy pierwszy element tablicy, ponieważ jest już na właściwym miejscu. Po znalezieniu szukanej liczby zamieniamy ją miejscami z drugim elementem tablicy. W kolejnych krokach postępujemy podobnie. Może się zdarzyć, że kolejny poszukiwany element jest już na właściwej pozycji.

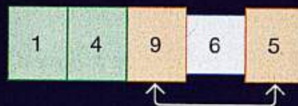
Algorytm sortowania przez wybieranie

Działanie algorytmu pokażemy dla tablicy, w której na kolejnych pozycjach znajdują się liczby: 6, 4, 9, 1, 5. Elementy uporządkujemy niemalejąco.

Krok 1 Szukamy najmniejszego elementu w całej tablicy. Jest nim liczba 1. Zamieniamy ją miejscami z pierwszą liczbą w tablicy – liczbą 6.

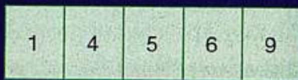
Krok 2 Szukamy najmniejszego elementu tablicy, pomijając element pierwszy. Szukaną liczbą jest 4 – znajduje się ona na właściwej pozycji.

Krok 3 Pomijamy pierwszy i drugi element tablicy i szukamy najmniejszej wśród pozostałych liczb – jest to liczba 5. Zamieniamy ją miejscami z trzecim elementem tablicy – liczbą 9.



Krok 4 Pomijamy trzy pierwsze elementy tablicy i szukamy liczby na przedostatnią pozycję. Jest to liczba 6, która znajduje się już na właściwym miejscu. Wyznaczenie przedostatniego elementu wskazuje jednocześnie element na ostatnią pozycję.

Wynikiem działania algorytmu jest tablica:



Zapis algorytmu porządkującego elementy tablicy niemalejąco metodą sortowania przez wybieranie może być następujący:

```
dla i ← 0, 1, ..., n – 2 wykonuj
    m ← i
    dla j ← i + 1, i + 2, ..., n – 1 wykonuj
        jeśli A[j] < A[m] to m ← j
    pom ← A[i]
    A[i] ← A[m]
    A[m] ← pom
```

Warto wiedzieć

Aby wyeliminować sytuację, w której liczbę zamieniamy samą ze sobą, trzeba by wprowadzić dodatkowe porównanie elementów przy każdej iteracji zewnętrznej pętli. Sytuacja taka dla dłuższych ciągów będzie jednak zachodzić rzadko.

W każdej iteracji zewnętrznej pętli jest znajdowany najmniejszy element badanego ciągu. Liczba powtórzeń wewnętrznej pętli zależy od wartości zmiennej sterującej zewnętrznej pętli. Rozpoczynamy od porównania elementu znajdującego się na pozycji, na którą wybierana jest liczba, z następnym elementem. Czasami liczba jest zamieniana sama ze sobą. Oto zapis funkcji sortującej tablicę liczb:

```
1.  void Sortuj(int A[])
2.  {
3.      int i, j, m, pom;
4.      for (i=0;i<N-1;i++)
5.      {
6.          m=i;
7.          for (j=i+1;j<N;j++)
8.              if (A[j]<A[m]) m=j;
9.          pom=A[i];
10.         A[i]=A[m];
11.         A[m]=pom;
12.     }
13. }
```

Fragment kodu źródłowego programu sortującego niemalejąco tablicę liczb – funkcja realizująca algorytm sortowania przez wybieranie

Ćwiczenie 2

Napisz i przetestuj program sortujący niemalejąco tablicę liczb całkowitych algorytmem sortowania przez wybieranie.

12.4. Sortowanie przez wstawianie

Sortowanie przez wstawianie

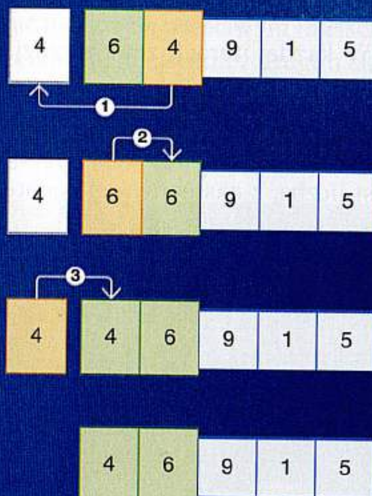
Metoda **sortowania przez wstawianie** polega na przeglądaniu kolejnych elementów ciągu i wstawianiu ich w odpowiednich miejscach już uporządkowanego podciągu (fragment ciągu po wstawieniu elementu powinien być uporządkowany). Aby wstawić element ciągu we właściwe miejsce, zapamiętuje się go w zmiennej pomocniczej. Następnie elementy posortowanego podciągu, które powinny się znaleźć za danym elementem, przesuwa się w prawo o jedną pozycję. Na końcu kopiuje się wartość ze zmiennej pomocniczej w zwolnione miejsce.

Algorytm sortowania przez wstawianie

Porządkowanie elementów tą metodą pokażemy na przykładzie pięcioelementowej tablicy, w której na kolejnych pozycjach znajdują się liczby: 6, 4, 9, 1, 5. Elementy uporządkujemy niemalejąco. Pierwszy krok ilustruje, jak uporządkować niemalejąco dwa elementy tablicy. Liczby przy strzałkach oznaczają kolejność wykonywania operacji.

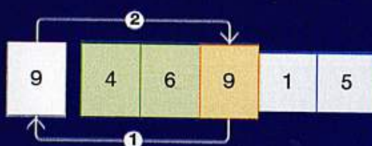
Krok 1

- Przyjmujemy, że pierwszy element tablicy jest pierwszym uporządkowanym elementem. Drugi element, liczbę 4, zapamiętujemy w zmiennej pomocniczej.
- Liczba 4 powinna się znaleźć przed liczbą 6, dlatego aby zrobić dla niej miejsce, przesuwamy liczbę 6 o jedną pozycję w prawo.
- Liczbę 4 ze zmiennej pomocniczej zapisujemy jako pierwszy element tablicy.
- Wynikiem jest uporządkowanie niemalejąco dwóch pierwszych elementów wyjściowej tablicy.

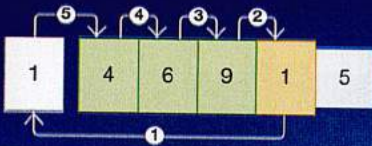


Postępując analogicznie, w drugim kroku uporządkujemy niemalejąco trzy elementy tablicy, w następnym – cztery itd. Poniżej znajduje się skrócony opis kolejnych kroków.

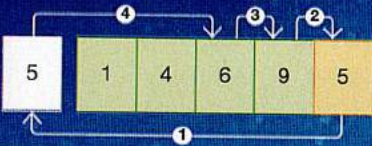
Krok 2 Zapamiętujemy liczbę 9 w zmiennej pomocniczej. Liczba 9 powinna się znaleźć za liczbami 4 i 6, dlatego nie trzeba zmieniać położenia tych liczb, a liczba 9 wraca na swoją pozycję.



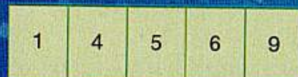
Krok 3 Zapamiętujemy liczbę 1 w zmiennej pomocniczej. Przesuwamy w prawo o jedną pozycję kolejno liczby: 9, 6 i 4. Zapisujemy liczbę 1 na pierwszym miejscu w tablicy.



Krok 4 Zapamiętujemy liczbę 5 w zmiennej pomocniczej. Przesuwamy liczbę 9, a następnie liczbę 6 o jedną pozycję w prawo. Wstawiamy liczbę 5 ze zmiennej pomocniczej w zwolnione miejsce.



Wynikiem działania algorytmu jest tablica:



Ponieważ na początku przyjmujemy, że pierwszy element tablicy jest uporządkowany, rozpoczynamy porządkowanie od drugiego elementu (o indeksie 1). Zapis algorytmu porządkującego liczby niemalejąco metodą sortowania przez wstawianie może wyglądać następująco:

```
dla i ← 1, 2, ..., n - 1 wykonuj
    pom ← A[i]    // zapisanie rozpatrywanego
                  // elementu w zmiennej pomocniczej
    j ← i - 1     // pozycja pierwszego elementu
                  // do ewentualnego przesunięcia
    dopóki j ≥ 0 oraz A[j] > pom wykonuj
        A[j+1] ← A[j] // przesunięcie elementu
        j ← j - 1     // pozycja kolejnego elementu
    A[j+1] ← pom // wstawienie elementu ze zmiennej
                 // pomocniczej na zwolnioną pozycję
```

Warunek pętli **dopóki** można uprościć, wprowadzając **wartownika**. Wówczas funkcję zmiennej pomocniczej (oraz wartownika) będzie pełnić element A[0]. Elementy sortowane będą miały indeksy od 1 do n (liczba elementów tablicy będzie wynosić $n + 1$). Nie trzeba będzie sprawdzać, czy indeks tablicy jest większy lub równy zero. Nawet gdy wstawiany będzie element najmniejszy, to w ostatnim obrocie pętli (dla $j = 0$) zostanie on porównany sam ze sobą i pętla zakończy działanie. Oto zmodyfikowana specyfikacja problemu i pseudokod:

Wartownik, s. 64

Specyfikacja

Dane: A[1..n] – tablica n liczb całkowitych.

Wynik: tablica A z uporządkowanymi elementami, A[1] ⩽ ... ⩽ A[n].

```
dla i ← 2, 3, ..., n wykonuj
    A[0] ← A[i]
    j ← i - 1
    dopóki A[j] > A[0] wykonuj
        A[j+1] ← A[j]
        j ← j - 1
    A[j+1] ← A[0]
```

A to ciekawe

Sortowanie w tańcu

Okazuje się, że można tłumaczyć metodę sortowania, dobrze się przy tym bawiąc. W różnych serwisach możemy obejrzeć, jak wyjaśniają zasady sortowania tancerze. Jeden z takich filmów dostępny jest pod linkiem https://www.youtube.com/watch?v=lyZQPjUT5B4.

Oto kod źródłowy funkcji realizującej pseudokod podany jako ostatni:

Fragment kodu źródłowego programu sortującego niemalejąco tablicę liczb – funkcja realizująca algorytm sortowania przez wstawianie

```
1.  void Sortuj(int A[])
2.  {
3.      int i, j;
4.      for (i=2;i<=N;i++)
5.      {
6.          A[0]=A[i];
7.          j=i-1;
8.          while (A[j]>A[0])
9.          {
10.             A[j+1]=A[j];
11.             j--;
12.         }
13.         A[j+1]=A[0];
14.     }
15. }
```

Element tablicy znajdujący się na pozycji 0 pełni funkcję wartownika, a porządkowane liczby to elementy tablicy na pozycjach od 1 do `N`. W związku z tym tablica w funkcji `main` powinna być deklarowana jako `N+1`-elementowa:

```
int A[N+1];
```

Dodatkowo w programie należy tak zmodyfikować funkcje `Losuj` i `Wypisz`, aby pętle przeglądały elementy tablicy o indeksach od 1 do `N`.

Ćwiczenie 3

Napisz i przetestuj program sortujący niemalejąco tablicę liczb całkowitych algorytmem sortowania przez wstawianie.

12.5. Porównanie metod sortowania prostego

Operacje dominujące (elementarne), s. 164

Porównamy teraz przedstawione w tym temacie algorytmy sortowania. Dla algorytmów sortowania prostego **operacjami dominującymi (elementarnymi)** będą operacje porównania elementów ciągu.

Policzmy te operacje dla algorytmów sortowania bąbelkowego oraz sortowania przez wybieranie. Można to dosyć łatwo zrobić, ponieważ w zapisie tych algorytmów występują tylko pętle **for**, które w tych algorytmach są wykonywane tyle samo razy niezależnie od sortowanych danych (wylosowanych liczb). Zewnętrzna pętla wykonuje się $n - 1$ razy. Wewnętrzna pętla za pierwszym razem jest powtarzana także $n - 1$ razy, za każdym kolejnym – o jeden mniej. Korzystając ze wzoru na sumę wyrazów ciągu arytmetycznego, możemy obliczyć liczbę porównań. Jest ona równa:

$$(n-1)+(n-2)+\ldots+2+1=\frac{(n-1)+1}{2}\cdot(n-1)=\frac{n^2}{2}-\frac{n}{2}$$

Czasowa złożoność obliczeniowa, s. 164

Zatem **czasowa złożoność obliczeniowa** jest kwadratowa: $O(n^2)$.

Czy algorytmy sortowania bąbelkowego i sortowania przez wybieranie są równorzędne? Zwróć uwagę, że nie liczyliśmy wszystkich operacji, a jedynie dominujące operacje porównania. Pamiętajmy, że algorytmy jeszcze zamieniają elementy tablicy, wykonując operacje przypisania wartości. Algorytm sortowania bąbelkowego realizuje to w wewnętrznej pętli, a sortowania przez wybieranie – w zewnętrznej, więc zdecydowanie mniej razy. Można powiedzieć, że ten drugi sposób porządkowania jest sprawniejszy.

Trudniej będzie oszacować liczbę porównań w algorytmie sortowania przez wstawianie. Wewnętrzna pętla **while** wykonuje się różną liczbę razy w zależności od danych – relacji pomiędzy liczbami w tablicy. Załóżmy sytuację pesymistyczną, kiedy za każdym razem przesuwamy wszystkie uporządkowane już elementy, czyli dany element jest wstawiany na pierwszą pozycję (tablica jest odwrotnie uporządkowana). Wówczas algorytm wykona dokładnie tyle samo porównań co algorytm sortowania bąbelkowego lub przez wybieranie. **Złożoność pesymistyczna** jest równa $\frac{n^2}{2}-\frac{n}{2}$, czyli $O(n^2)$. Dla większej liczby elementów ryzyko wylosowania liczb odwrotnie uporządkowanych jest bliskie zeru. Dla losowych ciągów będzie przesuwanych średnio około połowy elementów, więc sortowanie przez wstawianie wykona rzędu $\frac{n^2}{4}$ porównań. **Złożoność oczekiwana (średnia)** jest więc także kwadratowa: $O(n^2)$.

Złożoność pesymistyczna, s. 165

Złożoność oczekiwana (średnia), s. 165

Sortowanie przez wstawianie jest nieznacznie sprawniejsze w porównaniu z dwoma pozostałymi metodami sortowania prostego. Należy jednak pamiętać, że algorytm sortowania przez wstawianie, przesuwając elementy, wykonuje wiele operacji przypisania wartości (więcej niż sortowanie przez wybieranie, ale mniej niż sortowanie bąbelkowe).

Ćwiczenie 4

Napisz program sortujący niemalejąco tablicę n liczb całkowitych algorytmem sortowania przez wstawianie, a następnie zmodyfikuj go tak, aby liczył wykonywane porównania. Przetestuj działanie programu dla różnych wartości n. Porównaj otrzymane wyniki z wartością $\frac{n^2}{4}$.

Zapamiętaj

Jeśli sortujemy niewielką liczbę elementów, nie ma znaczenia, której metody sortowania użyjemy. Wszystkie omówione w tym temacie algorytmy sortowania działają wolno przy porządkowaniu dużych zbiorów danych.

Podsumowanie

- Sortowanie bąbelkowe (przez prostą zamianę) polega na porównywaniu sąsiednich elementów ciągu i ewentualnej ich zamianie tak, aby spełniały relację porządkującą.
- W sortowaniu przez wybieranie w kolejnych krokach znajduje się najmniejszy element sortowanego ciągu i przenosi ten element na odpowiednią pozycję ciągu wynikowego (przez zamianę elementów miejscami).
- Sortowanie przez wstawianie polega na wstawianiu kolejnych elementów porządkowanego ciągu w już uporządkowany fragment ciągu.
- Dla algorytmów sortowania prostego operacją dominującą jest porównywanie elementów.
- Dla algorytmów sortowania prostego czasowa złożoność obliczeniowa (zarówno oczekiwana, jak i pesymistyczna) jest kwadratowa: $O(n^2)$.

Zadania

* **1** Napisz program, który uporządkuje liczby w tablicy tak, aby pierwszym elementem tablicy była liczba z najmniejszą cyfrą jedności, a ostatnim – liczba z największą cyfrą jedności. Jeśli dwie liczby mają taką samą cyfrę jedności, uznaj je za równe.

* **2** Napisz program, który posortuje niemalejąco tablicę liczb całkowitych. Jako pierwszy powinien zostać wyznaczony ostatni element tablicy, następnie przedostatni itd. Podczas rozwiązywania zadania skorzystaj z algorytmu sortowania przez wybieranie.

* **3** Zmodyfikuj omówiony w temacie program realizujący algorytm sortowania bąbelkowego tak, aby sprawdzał, czy w danej iteracji wewnętrznej pętli któreś z elementów zostały zamienione miejscami. Algorytm powinien kończyć działanie, jeżeli w danej iteracji nie było ani jednej zamiany elementów.

** **4** Napisz program sortujący liczby całkowite nieujemne według sumy ich cyfr: na pierwszym miejscu powinna się znaleźć liczba, której suma cyfr jest najmniejsza.

** **5** Napisz program sortujący liczby całkowite nieujemne według liczby jedynek w ich zapisie binarnym: na pierwszym miejscu powinna się znaleźć liczba z największą liczbą jedynek w zapisie binarnym.

** **6** Napisz program porządkujący liczby w tablicy tak, aby najpierw znalazły się w niej liczby nieparzyste, a potem liczby parzyste.

** **7** Napisz program, który uporządkuje niemalejąco tablicę liczb, wykorzystując algorytm sortowania przez wstawianie. Miejsce do wstawienia elementu program powinien znajdować metodą przeszukiwania binarnego.
Uwaga: Przesunięcie elementów nadal będzie liniowe.

*** **8** Napisz program, który uporządkuje liczby całkowite w tablicy tak, aby pierwszym elementem tablicy była liczba z najmniejszą cyfrą jedności, a ostatnim – liczba z największą cyfrą jedności. Jeśli dwie liczby mają taką samą cyfrę jedności, uznaj je za równe. Rozwiąż zadanie tak, aby złożoność czasowa algorytmu była nie większa niż $10n$.

*** **9** Napisz program losujący tablicę liczb całkowitych z zakresu od 2 do 1000 oraz sortujący je według najmniejszego czynnika pierwszego. Najpierw w tablicy powinny się znaleźć liczby podzielne przez 2, potem podzielne przez 3 (a niepodzielne przez 2), następnie podzielne przez 5 (a niepodzielne przez 2 i 3) itd.