

# 1. Zadania rozwiązywane bez użycia komputera

## 1.1. Analiza algorytmów

### Zadanie 1.

#### Wiązka zadań *Ciągi rekurencyjne*

Dana jest następująca funkcja rekurencyjna:

```

funkcja wynik( i )
    jeżeli  $i < 3$ 
        zwróć 1 i zakończ;
    w przeciwnym razie
        jeżeli  $i \bmod 2 = 0$ 
            zwróć  $wynik(i - 3) + wynik(i - 1) + 1$ 
        w przeciwnym razie
            zwróć  $wynik(i - 1) \bmod 7$ 
  
```

**Uwaga:** Operator mod oznacza resztę z dzielenia.

### 1.1.

Uzupełnij poniższą tabelę:

<i>i</i>	<i>wynik(i)</i>
2	1
3	
4	
5	
6	
7	
8	

### 1.2.

*Wykonaniem elementarnym* nazywać będziemy wykonanie *wynik(0)*, *wynik(1)* lub *wynik(2)*. Natomiast *złożonością elementarną wynik(i)* nazywamy liczbę *wykonań elementarnych* będących efektem uruchomienia *wynik(i)*. Złożoność elementarną *wynik(i)* oznaczamy przez *E(i)*.

Na przykład złożoność elementarna *wynik(4)* wynosi  $E(4) = 2$ , ponieważ wykonując *wynik(4)*, wywołamy *wynik(3)* i *wynik(1)* (wykonanie elementarne), a z kolei przy wykonaniu *wynik(3)* wywołamy *wynik(2)* (drugie wykonanie elementarne).

Uzupełnij poniższą tabelę:

$i$	$E(i)$
0	1
3	1
5	3
7	5
9	5
10	9

Okazuje się, że  $E(i)$  można opisać rekurencyjnym wyrażeniem, którego niekompletną postać podajemy poniżej. Uzupełnij brakujące miejsca tak, aby  $E(i)$  dawało poprawną złożoność elementarną  $wynik(i)$  dla każdego całkowitego nieujemnego  $i$ .

$$\begin{aligned}
 E(0) &= E(1) = E(2) = 1 \\
 E(i) &= E(\dots\dots\dots) + E(\dots\dots\dots) && \text{dla parzystego } i > 2 \\
 E(i) &= E(\dots\dots\dots) && \text{dla nieparzystego } i > 2
 \end{aligned}$$

### 1.3.

Naszym celem jest wyznaczenie największej liczby spośród wartości funkcji  $wynik(0)$ ,  $wynik(1)$ , ...,  $wynik(1000)$  bez konieczności rekurencyjnego wyznaczania kolejnych wartości. Poniżej prezentujemy niekompletny algorytm realizujący to zadanie.

```

W[0] ← 1
W[1] ← 1
W[2] ← 1
max_wart ← 1
dla  $i = 3, 4, \dots, 1\,000$  wykonuj
    jeżeli  $i \bmod 2 = 0$ 
         $W[i] \leftarrow \dots\dots\dots$ 
    w przeciwnym razie
         $W[i] \leftarrow \dots\dots\dots$ 
    jeżeli  $W[i] > max\_wart$ 
         $\dots\dots\dots$ 
zwróć  $max\_wart$ 

```

Uzupełnij brakujące miejsca w algorytmie tak, aby zwracał on największą liczbę spośród  $wynik(0)$ ,  $wynik(1)$ , ...,  $wynik(1000)$ .

### Komentarz do zadania

#### 1.1.

Do rozwiązania tego zadania stosujemy definicję rekurencyjną  $wynik(i)$ , wynikającą wprost z podanego pseudokodu:

$$\begin{aligned}
 wynik(0) &= wynik(1) = wynik(2) = 1 \\
 wynik(i) &= wynik(i-3) + wynik(i-1) + 1 \text{ dla parzystych } i > 2 \\
 wynik(i) &= wynik(i-1) \bmod 7 \text{ dla nieparzystych } i > 2
 \end{aligned}$$

A zatem:

- $wynik(3) = wynik(2) \bmod 7 = 1 \bmod 7 = 1$
- $wynik(4) = wynik(3) + wynik(1) + 1 = 1 + 1 + 1 = 3$

- $wynik(5) = wynik(4) \bmod 7 = 3 \bmod 7 = 3$
- $wynik(6) = wynik(5) + wynik(3) + 1 = 3 + 1 + 1 = 5$
- $wynik(7) = wynik(6) \bmod 7 = 5 \bmod 7 = 5$
- $wynik(8) = wynik(7) + wynik(5) + 1 = 5 + 3 + 1 = 9$

Zwróćmy uwagę, że w powyższym rozwiązaniu dla kolejnych argumentów (większych niż 2) funkcja *wynik* jest wywoływana wielokrotnie. Nie musimy jej jednak wielokrotnie wyznaczać, np. raz obliczony *wynik(3)* możemy wykorzystać przy każdym odwołaniu, bez ponownego obliczania.

### 1.2.

Najpierw rozwiążemy drugą część zadania, czyli podamy zależność rekurencyjną, określającą złożoność elementarną. Na podstawie tej zależności łatwo rozwiążemy pierwszą część zadania.

Ponieważ wykonania *wynik(0)*, *wynik(1)* oraz *wynik(2)* kończą się natychmiastowym zwróceniem wyniku, każdemu z nich odpowiada dokładnie jedno wykonanie elementarne. To daje nam warunek brzegowy rekurencji:

$$E(0) = E(1) = E(2) = 1.$$

Natomiast dla każdego  $i > 2$  sytuacja jest zależna od parzystości liczby  $i$ :

- Dla parzystych  $i > 2$  wykonanie *wynik(i)* pociąga za sobą wykonanie *wynik(i-1)* oraz *wynik(i-3)*. Zgodnie z przyjętymi oznaczeniami wykonanie *wynik(i-1)* wymaga  $E(i-1)$  wykonań elementarnych. Analogicznie *wynik(i-3)* wymaga  $E(i-3)$  wykonań elementarnych. Zatem *wynik(i)* pociąga za sobą  $E(i-1) + E(i-3)$  wykonań elementarnych. Stąd:

$$E(i) = E(i-1) + E(i-3) \text{ dla parzystego } i > 2.$$

- Z kolei wykonanie *wynik(i)* dla nieparzystego  $i > 2$  powoduje tylko wykonanie *wynik(i-1)*. Stąd:

$$E(i) = E(i-1) \text{ dla nieparzystego } i > 2.$$

Ostatecznie:

$$E(i) = \begin{cases} 1, & \text{dla } i \in \{0, 1, 2\}, \\ E(i-3) + E(i-1), & \text{dla parzystego } i > 2, \\ E(i-1), & \text{dla nieparzystego } i > 2, \end{cases}$$

Znając rekurencyjną formułę dla wyznaczania  $E(i)$ , możemy wypełnić podaną w zadaniu tabelkę, stosując tę formułę dla kolejnych  $i=3, 4, \dots, 10$ , podobnie jak w zadaniu 1 robiliśmy to dla funkcji *wynik*. Drobną różnicą polega jedynie na tym, że w zadaniu 1 formuła zapisana była w innej postaci, a mianowicie w postaci pseudokodu.

### 1.3.

Naturalnym sposobem uniknięcia kolejnych wywołań rekurencyjnych jest tablicowanie wyników, co zostało zasugerowane w podanym niekompletnym algorytmie. Jeśli wartości *wynik(0)*, *wynik(1)*, ..., *wynik(i-1)* przechowywane są w komórkach  $W[0]$ ,  $W[1]$ , ...,  $W[i-1]$  tablicy  $W$ , to wartość *wynik(i)* możemy wyznaczyć jako  $W[i-3] + W[i-1] + 1$  dla  $i$  parzystych oraz  $W[i] \bmod 7$  dla  $i$  nieparzystych. Aby wyznaczoną wartość zapamiętać w odpowiedniej ko-

mórcie tablicy  $W$ , użyjemy powyższych wyrażeń do uzupełnienia instrukcji podstawienia w podanym algorytmie.

Zmienna  $max\_wart$ , jak wskazuje nazwa, może być użyta do przechowywania największej wyznaczonej dotychczas wartości funkcji  $wynik$ , zatem aktualizujemy ją zawsze, gdy  $W[i] > max\_wart$ . Poniżej prezentujemy kompletne rozwiązanie:

```

 $W[0] \leftarrow 1$ 
 $W[1] \leftarrow 1$ 
 $W[2] \leftarrow 1$ 
 $max\_wart \leftarrow 1$ 
dla  $i = 3, 4, \dots, 1\,000$  wykonuj
    jeżeli  $i \bmod 2 = 0$ 
         $W[i] \leftarrow W[i-1] + W[i-3] + 1$ 
    w przeciwnym razie
         $W[i] \leftarrow W[i] \leftarrow W[i-1] \bmod 7$ 
    jeżeli  $W[i] > max\_wart$ 
         $max\_wart \leftarrow W[i]$ 
zwróć  $max\_wart$ 

```

## Rozwiązanie

### 1.1.

$i$	wynik( $i$ )
2	1
3	1
4	3
5	3
6	5
7	5
8	9

### 1.2.

Poprawne wartości  $E(i)$  dla argumentów podanych w tabeli

$i$	$E(i)$
0	1
3	1
5	2
7	3
9	5
10	8

Poprawna definicja rekurencyjna:

$$E(0) = E(1) = E(2) = 1$$

$$E(i) = E(i-1) + E(i-3) \text{ dla parzystego } i > 2$$

$$E(i) = E(i-1) \text{ dla nieparzystego } i > 2$$

## 1.3.

```

 $W[0] \leftarrow 1$ 
 $W[1] \leftarrow 1$ 
 $W[2] \leftarrow 1$ 
 $max\_wart \leftarrow 1$ 
dla  $i = 3, 4, \dots, 1000$  wykonuj
    jeżeli  $i \bmod 2 = 0$ 
         $W[i] \leftarrow W[i-1] + W[i-3] + 1$ 
    w przeciwnym razie
         $W[i] \leftarrow W[i-1] \bmod 7$ 
    jeżeli  $W[i] > max\_wart$ 
         $W[i] \leftarrow max\_wart$ 
zwróć  $max\_wart$ 

```

## Zadanie 2.

Wiązka zadań *Ułamki dwójkowe*

W systemach pozycyjnych o podstawie innej niż 10 można zapisywać nie tylko liczby całkowite, ale również rzeczywiste z pewną dokładnością. Na przykład w systemie dwójkowym cyfry po przecinku odpowiadają kolejnym potęgom  $1/2$  (jednej drugiej). Cyfra 1 na pierwszym miejscu po przecinku odpowiada  $1/2$ , na drugim miejscu —  $1/4$ , na trzecim —  $1/8$  i tak dalej.

Na przykład  $(0,101)_2 = 1/2 + 1/8 = 5/8 = 0,625_{10}$ . Podobnie jak w systemie dziesiętnym nie każda liczba daje się zapisać w ten sposób dokładnie — na przykład liczba  $1/3$  nie ma skończonego rozwinięcia w systemie dwójkowym (ani też w dziesiętnym). Można jednak stosunkowo łatwo wyznaczyć zadaną liczbę początkowych cyfr po przecinku dla każdej liczby rzeczywistej.

Następujący algorytm przyjmuje na wejściu liczbę rzeczywistą  $x$  należącą do przedziału  $[0, 1)$  oraz dodatnią liczbę całkowitą  $k$  i wypisuje  $k$  pierwszych cyfr liczby  $x$  w zapisie dwójkowym. Przeanalizuj algorytm i odpowiedz na podane pytania.

Dane:

$x$  — liczba rzeczywista,  $0 \leq x < 1$ ,  
 $k$  — liczba całkowita dodatnia.

Wynik:

zapis dwójkowy liczby  $x$  do  $k$ -tego miejsca po przecinku.

**funkcja** binarny( $x, k$ )

```

wypisz „0,”
 $y \leftarrow x$ 
dla  $i = 1, 2, \dots, k$  wykonuj
    (*) jeżeli  $y \geq 1/2$ 
        wypisz „1”
    w przeciwnym razie
        wypisz „0”
     $y \leftarrow y * 2$ 
    jeżeli  $y \geq 1$ 
         $y \leftarrow y - 1$ 

```

**2.1.**

Podaj liczbę wypisaną przez algorytm dla  $x = 0,6$ ,  $k = 5$  oraz wartości zmiennej  $y$  przy każdorazowym wykonaniu wiersza oznaczonego (\*).

Kolejne wykonanie (*)	Wartość zmiennej $y$
1	0,6
2	0,2
3	0,4
4	0,8
5	0,6

Liczba wypisana przez algorytm: 0,10011.

**2.2.**

Podaj przykład liczby  $x$ , dla której po wykonaniu funkcji  $\text{binarny}(x, 4)$  zmienna  $y$  ma wartość 0, a po wykonaniu funkcji  $\text{binarny}(x, 3)$  zmienna  $y$  nie jest równa 0.

Musi to być liczba, która ma dokładnie cztery cyfry po przecinku w zapisie dwójkowym. Możliwe odpowiedzi to: 1/16 (0,0625), 3/16 (0,1875), 5/16 (0,3125), ..., 15/16 (0,9375)

**2.3.**

W systemie trójkowym używa się cyfr 0, 1 i 2. Cyfra 1 na pierwszym miejscu po kropce oznacza  $1/3$ , zaś 2 oznacza  $2/3$ . Na drugim miejscu są to odpowiednio  $1/9$  i  $2/9$ , na trzecim —  $1/27$  i  $2/27$  i tak dalej, z kolejnymi potęgami trójki w mianownikach.

Poniżej podany jest algorytm wypisujący dla zadanej liczby rzeczywistej  $x$  z przedziału  $[0,1)$  oraz liczby całkowitej dodatniej  $k$  pierwsze  $k$  cyfry zapisu  $x$  w systemie trójkowym. Uzupełnij luki tak, aby algorytm działał prawidłowo.

**funkcja** trójkowy( $x$ ,  $k$ )

wypisz „0,”

$y \leftarrow x$

dla  $i = 1, 2, \dots, k$  wykonuj

jeżeli  $y \geq 2/3$

wypisz „2”

jeżeli .....

wypisz „1”

jeżeli .....

wypisz „0”

$y = y * 3$

jeżeli  $y \geq 2$

.....

jeżeli  $y \geq 1$

.....

**funkcja** trójkowy( $x$ ,  $k$ )

wypisz „0,”

$y \leftarrow x$

dla  $i = 1, 2, \dots, k$  wykonuj

jeżeli  $y \geq 2/3$

wypisz „2”

jeżeli  $y \geq 1/3$  oraz  $y < 2/3$

wypisz „1”

jeżeli  $y < 1/3$

wypisz „0”

$y \leftarrow y * 3$

jeżeli  $y \geq 2$

$y \leftarrow y - 2$  // prawidłową odpowiedzią jest również  $y \leftarrow y - 1$

jeżeli  $y \geq 1$

$y \leftarrow y - 1$

**Komentarz do zadania****2.1.**

Algorytm zaczyna od wypisania zera i przecinka dziesiętnego. Następnie zaczyna się główna pętla: w pierwszej iteracji  $y = 0,6$ , a zatem pierwszą po przecinku cyfrą jest 1. Mnożymy  $y$  przez 2 i jeśli przekroczy 1, odejmujemy 1. Ponieważ  $0,6 \cdot 2 = 1,2$ , to po tej iteracji zmienna  $y$

przybierze wartość 0,2. W kolejnym obrocie pętli wypiszemy cyfrę 0 (jako że  $0,2 < 0,5$ ), po czym podwoimy  $y$ , otrzymując 0,4. Kontynuując w ten sposób działania, dojdziemy do odpowiedzi takiej jak wzorcowa.

Można przy okazji zauważyć, że po czwartej iteracji  $y$  znowu przybierze wartość 0,6, a zatem dalsze kroki algorytmu, gdybyśmy wykonali ich więcej, byłyby identyczne z pierwszymi czterema. Widać stąd, że  $(0,6)_2 = 0,1001100110011\dots$

## 2.2.

Wartość zmiennej  $y$  równa zero oznacza po prostu, że już wszystkie dalsze cyfry dwójkowe liczby  $x$  są zerami, innymi słowy, że rozwinięcie dwójkowe  $x$  się skończyło.

W zadaniu pytamy zatem o liczbę, która po trzech iteracjach ma jeszcze dalsze niezerowe cyfry dwójkowe do wypisania, ale po czterech już nie ma. Musi to więc być liczba, która w rozwinięciu dwójkowym ma dokładnie cztery cyfry po przecinku. Najmniejszą taką liczbą jest  $(0,0001)_2$ , czyli  $1/16 = 0,0625$ . Innymi możliwymi odpowiedziami są na przykład  $(0,0011)_2 = 3/16 = 0,1875$  albo  $(0,1111)_2 = 15/16 = 0,9375$ .

## 2.3.

Algorytm *binarny*( $x, k$ ) opiera się na następującym pomysśle: jeśli liczba  $x$  jest nie mniejsza od  $1/2$ , to jej pierwszą cyfrą dwójkową po przecinku musi być 1. Jeśli teraz pomnożymy liczbę  $x$  przez 2, to odpowiada to przesunięciu całego rozwinięcia o jedno miejsce w lewo. Druga po przecinku cyfra liczby  $x$  to pierwsza cyfra po przecinku  $2x$ , czyli możemy ją wyznaczyć podobnie jak poprzednio: sprawdzając, czy jest większa lub równa  $1/2$ . Oczywiście musimy najpierw pominąć stojącą przed przecinkiem część całkowitą — odejmujemy zatem 1, jeśli liczba przekroczyła 1 w czasie mnożenia.

Bardzo podobną technikę stosujemy w algorytmie *trojkowy*( $x, k$ ). Tutaj pierwsza cyfra po przecinku powinna być równa 2, jeśli liczba  $x$  jest nie mniejsza od  $2/3$ , 1 — jeśli  $x$  należy do przedziału  $[1/3, 2/3)$ , a 0 — jeśli  $x$  jest mniejsze od  $1/3$ . Następnie dokonujemy przesunięcia w lewo, mnożąc liczbę przez 3, po czym usuwamy z niej część całkowitą. Istotną różnicą jest to, że teraz część całkowita liczby może wynosić 0, 1 lub 2, zatem musimy odjąć 1 lub 2, zależnie od potrzeby:

```
dla i=1, 2, ..., k wykonuj
    jeżeli  $y \geq 2/3$ 
        wypisz „2”
    jeżeli  $y \geq 1/3$  oraz  $y < 2/3$ 
        wypisz „1”
    jeżeli  $y < 1/3$ 
        wypisz „0”
     $y \leftarrow y * 3$ 
    jeżeli  $y \geq 2$ 
         $y \leftarrow y - 2$ 
    jeżeli  $y \geq 1$ 
         $y \leftarrow y - 1$ 
```

Zamiast ostatnich czterech wierszy — odpowiadających pomijaniu części całkowitej — można było w oryginalnym algorytmie napisać tak:

```

jeżeli  $y \geq 2$ 
     $y \leftarrow y - 1$ 
jeżeli  $y \geq 1$ 
     $y \leftarrow y - 1$ 

```

Jeśli liczba będzie większa niż 2, algorytm najpierw odejmie 1, a następnie zauważy, że liczba wciąż jest większa niż 1, i odejmie znów jedynkę. Warto jeszcze wspomnieć, że gdyby luki nie narzucały konstrukcji algorytmu, można byłoby użyć krótszego zapisu:

```

dopóki  $y \geq 1$  wykonuj
     $y \leftarrow y - 1$ 

```

Miałoby to ten sam efekt: odrzucenie z liczby  $y$  jej części całkowitej.

### Zadanie 3.

#### Wiązka zadań *Ciekawe mnożenia*

Dana jest następująca funkcja rekurencyjna:

Dane:

$x$  — liczba całkowita,  
 $n$  — dodatnia liczba całkowita.

```

funkcja  $F(x, n)$ 
    jeżeli  $n = 1$ 
        podaj wynik  $x$  i zakończ
    w przeciwnym razie
        jeżeli  $n \bmod 3 = 0$ 
             $k \leftarrow F(x, n \text{ div } 3)$ 
        (*) podaj wynik  $k * k * k$  i zakończ
        w przeciwnym razie
        (**) podaj wynik  $x * F(x, n - 1)$  i zakończ

```

**Uwaga:** „div” jest operatorem dzielenia całkowitego.

#### 3.1.

Podaj wszystkie wywołania rekurencyjne funkcji  $F$  oraz obliczany po każdym wywołaniu wynik, jeśli na początku wywołamy  $F(2, 10)$ .

wywołanie	wynik
$F(2, 10)$	1024
$F(2, 9)$	512
$F(2, 3)$	8
$F(2, 1)$	2

#### 3.2.

Uzupełnij tabelę o brakujące elementy:



$x$	$n$	wynik $F(x, n)$
2	2	4
2	3	8
3	4	81
2	5	32
2	8	256
2	10	1024

**3.3.**

Uzupełnij tabelę, podając łączną liczbę mnożeń wykonanych w wierszach oznaczonych (\*) i (\*\*) po wywołaniu  $F$  dla podanych argumentów  $x$  i  $n$ :

$x$	$n$	Liczba operacji mnożenia
2	2	1
2	3	2
3	4	3
4	7	4
4	8	5
4	9	4

**3.4.**

Podaj, która z poniższych funkcji określa liczbę wszystkich operacji mnożenia wykonywanych przez powyższy algorytm dla argumentu  $n$  będącego potęgą trójki ( $n = 3^m$  dla pewnego nieujemnego  $m$ ):

- $lmnozen(n) = n \operatorname{div} 2$
  - $lmnozen(n) = \log_2 n$
  - $lmnozen(n) = 2 \cdot \log_3 n$
  - $lmnozen(n) = 1 + \sqrt{n}$
- $lmnozen(n) = 2 \cdot \log_3 n$

**Zadanie 4.****Wiązka zadań Silniowy system pozycyjny**

Pojęcie *silni* dla liczb naturalnych większych od zera definiuje się następująco:

$$n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot (n-1) \cdot n$$

Silniowy system pozycyjny to pozycyjny sposób zapisu liczb naturalnych, w którym mnożniki dla kolejnych pozycji są definiowane przez silnie kolejnych liczb naturalnych, tzn.

$$(x)_i = (x_n x_{n-1} x_{n-2} \dots x_2 x_1)_i = x_n \cdot n! + x_{n-1} \cdot (n-1)! + \dots + x_2 \cdot 2! + x_1 \cdot 1!$$

W systemie silniowym współczynnik  $x_i$ , który odpowiada mnożnikowi  $i!$ , spełnia zależność  $0 \leq x_i \leq i$ .

Zapis każdej liczby w silniowym systemie pozycyjnym jest jednoznaczny, tzn. każdą liczbę naturalną można zapisać tylko w jeden sposób i każdą liczbę naturalną można zapisać dokładnie w jeden sposób.

**Uwaga:** W poniższych zadaniach będziemy mieć do czynienia tylko z takimi liczbami, dla których współczynniki  $x_i$  spełniają zależność  $0 \leq x_i \leq 9$ .

### Przykład

$$(1220)_! = 1 \cdot 4! + 2 \cdot 3! + 2 \cdot 2! + 0 \cdot 1! = 24 + 12 + 4 + 0 = 40.$$

#### 4.1.

Uzupełnij tabelę. Zamień zapis liczby w systemie silniowym na jej zapis w systemie dziesiętnym.

liczba w systemie silniowym	liczba w systemie dziesiętnym
$(310)_!$	20
$(2011)_!$	51
$(54211)_!$	711

#### 4.2.

Podaj zapis w systemie silniowym największej liczby, jaką można w tym systemie zapisać na pięciu pozycjach.

Największa liczba 5-cyfrowa zapisana w zapisie silniowym to  $(54321)_!$

#### 4.3.

Zamiana zapisu liczby w systemie dziesiętnym na zapis w systemie silniowym może przebiegać według następującego schematu: Szukamy największej liczby  $k$ , której silnia nie przekracza liczby  $x$ . Pierwsza jej cyfra to wynik dzielenia całkowitego  $x$  przez  $k!$ . Kolejne cyfry zapisu silniowego (zaczynając od cyfr najbardziej znaczących) otrzymujemy przez wyznaczanie wyników dzielenia liczby  $x$  przez  $(k-1)!$ ,  $(k-2)!$ , ...,  $2!$ ,  $1!$ . Po wyznaczeniu cyfry  $x_i$ , odpowiadającej współczynnikowi  $i!$ , zmniejszamy wartość  $x$  o liczbę odpowiadającą cyfrze  $x_i$ , czyli  $x_i \cdot i!$ . Oznacza to, że  $x$  przyjmuje wartość  $x \bmod k!$ .

### Przykład

$x$	$k$	$x \div k!$	$x \bmod k!$
1548	6	2	108
108	5	0	108
108	4	4	12
12	3	2	0
0	2	0	0
0	1	0	0

Liczba dziesiętna 1548 w zapisie silniowym:  $(204200)_!$

Wykonaj zamianę liczby 5489 z systemu dziesiętnego na silniowy zgodnie z opisanym powyżej algorytmem. Uzupełnij poniższą tabelkę oraz podaj zapis silniowy liczby 5489.

$x$	$k$	$x \div k!$	$x \bmod k!$
5489	7	1	449
449	6	0	449
449	5	3	89
89	4	3	17
17	3	2	5
5	2	2	1
1	1	1	0

Liczba dziesiętna 5489 w zapisie silniowym: ..... $(1033221)_!$ .....

#### 4.4.

Poniżej przedstawiono algorytm z lukami, który zamienia zapis liczb z systemu dziesiętnego na system silniowy. Uzupełnij luki w tym algorytmie.

#### Specyfikacja

Dane:

$x$  — liczba całkowita dodatnia zapisana w systemie dziesiętnym,

Wynik:

$s$  — napis reprezentujący liczbę  $x$  zapisaną w systemie silniowym.

$silnia \leftarrow 1$

$k \leftarrow 1$

**dopóki** ( $silnia < x$ ) **wykonuj**

$k \leftarrow k + 1$

$silnia \leftarrow silnia * k$

**jeżeli** .....  
 $silnia \leftarrow silnia \div k$

$k \leftarrow k - 1$

$s \leftarrow ""$

**dopóki** ( $k > 0$ ) **wykonuj**

$cyfra \leftarrow \dots\dots\dots$

$s \leftarrow s \circ \text{tekst}(cyfra)$

$x \leftarrow \dots\dots\dots$

$silnia \leftarrow \dots\dots\dots$

$k \leftarrow k - 1$

**Przykładowa poprawna odpowiedź :**

$silnia \leftarrow 1$

$k \leftarrow 1$

**dopóki** ( $silnia < x$ ) **wykonuj**

$k \leftarrow k + 1$

$silnia \leftarrow silnia * k$

**jeżeli**  $silnia > x$

$silnia \leftarrow silnia \div k$

$k \leftarrow k - 1$

$s \leftarrow ""$

**dopóki** ( $k > 0$ ) **wykonuj**

$cyfra \leftarrow x \div silnia$

$s \leftarrow s \circ \text{tekst}(cyfra)$

$x \leftarrow x - cyfra * silnia$  lub  $x \leftarrow x \bmod silnia$

$silnia \leftarrow silnia \div k$

$k \leftarrow k - 1$

#### Uwaga

$\text{tekst}(x)$  oznacza funkcję zamieniającą liczbę  $x$  na jej zapis tekstowy

$""$  oznacza napis pusty

$u \circ v$  oznacza sklejenie dwóch napisów:  $u$  oraz  $v$

**Zadanie 5.****Wiązka zadań Sortowanie przez wstawianie na dwa sposoby**

Sortowanie przez wstawianie polega na powtarzaniu operacji wstawiania elementu do już uporządkowanego ciągu. Aby znaleźć w uporządkowanym ciągu miejsce, w które należy wstawić nowy element, można stosować różne strategie. Poniższy algorytm znajduje to miejsce metodą wyszukiwania binarnego.

**Specyfikacja**

Dane:

$n$  — liczba naturalna oznaczająca długość ciągu,

$A[1..n]$  — ciąg liczb całkowitych zapisanych w tablicy

Wynik:

$A[1..n]$  — tablica liczb całkowitych, w której liczby zostały ustawione w porządku niemalejącym

**Algorytm**

```

dla  $j = n - 1, n - 2, \dots, 1$  wykonuj
   $x \leftarrow A[j]$ 
   $p \leftarrow j$ 
   $k \leftarrow n + 1$ 
  dopóki  $k - p > 1$  wykonuj
     $i \leftarrow (p + k) \text{ div } 2$ 
    jeżeli  $x \leq A[i]$ 
       $k \leftarrow i$ 
    w przeciwnym razie
       $p \leftarrow i$ 
  dla  $i = j, j + 1, \dots, p - 1$  wykonuj
     $A[i] \leftarrow A[i + 1]$ 
   $A[p] \leftarrow x$ 

```

**5.1.**

Przeanalizuj działanie powyższego algorytmu dla ciągu 12, 4, 3, 10, 5, 11 o długości  $n = 6$  i podaj, ile razy zostaną wykonane instrukcje  $k \leftarrow i$  i  $p \leftarrow i$  dla kolejnych wartości  $j$  zamieszczonych w tabeli.

Wartość $j$	Liczba wykonań	
	$k \leftarrow i$	$p \leftarrow i$
5	1	0
4	1	1
3	2	0
2	1	1
1	0	3

**5.2.**

Uzupełnij luki w poniższym algorytmie sortowania przez wstawianie tak, aby znajdowanie miejsca na kolejny wstawiany element było realizowane metodą wyszukiwania liniowego.

**Specyfikacja**

Dane:

$n$  — liczba naturalna oznaczająca długość ciągu,  $A[1..n]$  — ciąg liczb całkowitych zapisanych w tablicy.

Wynik:

$A[1..n]$  — tablica liczb całkowitych, w której liczby zostały ustawione w porządku niemalejącym.

**Algorytm:**

**dla**  $j = n - 1, n - 2, \dots, 1$  **wykonuj**  
 $x \leftarrow \dots\dots\dots$   
 $i \leftarrow \dots\dots\dots$   
**dopóki**  $(i \leq n)$  **i**  $(x > A[i])$  **wykonuj**  
 $A[i - 1] \leftarrow A[i]$   
 $i \leftarrow i + 1$   
 $\dots\dots\dots \leftarrow x$

Uzupełnione fragmenty algorytmu:

**dla**  $j = n - 1, n - 2, \dots, 1$  **wykonuj**  
 $x \leftarrow A[j]$   
 $i \leftarrow j + 1$   
**dopóki**  $(i \leq n)$  **i**  $(x > A[i])$  **wykonuj**  
 $A[i - 1] \leftarrow A[i]$   
 $i \leftarrow i + 1$   
 $A[i - 1] \leftarrow x$

**5.3.**

Porównaj dwa algorytmy sortowania przez wstawianie: taki, w którym miejsce dla wstawianego elementu jest znajdowane metodą wyszukiwania binarnego, i taki, w którym jest ono znajdowane metodą wyszukiwania liniowego. Zaznacz znakiem X w odpowiedniej kolumnie, które zdanie jest prawdziwe (P), a które jest fałszywe (F).

Oba algorytmy dla ciągu 12, 4, 3, 10, 5, 11 wykonują

	P	F
jednakową liczbę porównań między elementami ciągu liczb.		<b>X</b>
jednakową liczbę przesunięć elementów w tablicy.	<b>X</b>	
tyle samo powtórzeń pętli zewnętrznej w algorytmie.	<b>X</b>	
jednakową liczbę instrukcji podstawienia wartości do zmiennej $x$ .	<b>X</b>	

**Zadanie 6.****Wiązka zadań *Od szczegółu do ogółu***

Rozważmy następujący algorytm:

Dane:

$k$  — liczba naturalna,  
 $A[1..2^k]$  — tablica liczb całkowitych.

**Algorytm 1:**

```

 $n \leftarrow 1$ 
dla  $i=1,2,\dots,k$  wykonuj
     $n \leftarrow 2 \cdot n$ 
 $s \leftarrow 1$ 
dopóki  $s < n$  wykonuj
     $j \leftarrow 1$ 
    dopóki  $j < n$  wykonuj
        (*)      jeżeli  $A[j] > A[j+s]$ 
        (**)       $\text{zamień}(A[j], A[j+s])$ 
         $j \leftarrow j+2 \cdot s$ 
     $s \leftarrow 2 \cdot s$ 
zwróć  $A[1]$ 

```

**Uwaga:** Funkcja  $\text{zamień}(A[j], A[j+s])$  zamienia miejscami wartości  $A[j]$  oraz  $A[j+s]$ .

**6.1.**

Prześledź działanie algorytmu 1 dla podanych poniżej wartości  $k$  i początkowych zawartości tablicy  $A$ . W każdym wierszu poniższej tabeli wpisz końcową zawartość tablicy  $A$ .

k	Początkowa zawartość tablicy $A[1\dots 2^k]$	Końcowa zawartość tablicy $A[1\dots 2^k]$
2	[4, 3, 1, 2]	[1, 4, 3, 2]
2	[2, 3, 4, 1]	[1, 3, 2, 4]
3	[1, 2, 3, 4, 5, 6, 7, 8]	[1, 2, 3, 4, 5, 6, 7, 8]
3	[8, 7, 6, 5, 4, 3, 2, 1]	[1, 8, 7, 6, 5, 4, 3, 2]
3	[4, 5, 6, 1, 8, 3, 2, 4]	[1, 5, 4, 6, 2, 8, 3, 4]

**6.2.**

Wskaż, które z poniższych zdań są prawdziwe (P), a które fałszywe (F), wstawiając znak X w odpowiedniej kolumnie:

	P	F
Po zakończeniu działania algorytmu 1 komórka $A[2^k]$ zawiera największą z liczb $A[1], \dots, A[2^k]$ .		X
Po zakończeniu działania algorytmu 1 spełniona jest nierówność $A[i] \leq A[i+1]$ dla każdego $i$ , takiego że $1 \leq i \leq 2^k$ .		X
Po zakończeniu działania algorytmu 1 komórka $A[1]$ zawiera najmniejszą z liczb $A[1], \dots, A[2^k]$ .	X	