Temat 7. Algorytmy wyszukiwania i porządkowania

Warto powtórzyć

1. Na czym polega iteracja?
2. W jaki sposób zapisujemy iterację w wybranym środowisku programowania?
3. W jaki sposób zapisujemy sytuację warunkową w wybranym środowisku programowania?
4. W jaki sposób przypisujemy zmiennej konkretną wartość, a jak wprowadzamy wartość zmiennej z klawiatury w wybranym środowisku programowania?

1. Wyszukiwanie elementu w zbiorze nieuporządkowanym

Algorytm wyszukiwania elementu w zbiorze nieuporządkowanym pokażemy na przykładzie wyszukiwania największej liczby spośród n liczb, zaczynając od prostego algorytmu dla zbioru dwuelementowego – wyboru większej z dwóch liczb. Następnie pokażemy, jak wyszukać dany element w zbiorze nieuporządkowanym.

1.1. Wybieranie większej z dwóch liczb

Zaczniemy od sformułowania zadania, zapisania jego specyfikacji oraz podania listy kroków algorytmu wyboru większej z dwóch liczb naturalnych.

Algorytm wyboru większej z dwóch liczb

Zadanie: Przedstaw w postaci listy kroków algorytm wyboru większej z dwóch liczb.
Dane: Dwie liczby naturalne: a, b ($a \neq b$).
Wynik: Wartość większej z dwóch liczb.

Lista kroków:

1. Zacznij algorytm.
2. Wprowadź wartość liczby *a*.
3. Wprowadź wartość liczby *b*.
4. Sprawdź, czy liczba *a* jest większa od liczby *b*. Jeśli tak, wyprowadź liczbę *a*, w przeciwnym przypadku wyprowadź liczbę *b*.
5. Zakończ algorytm.

Ćwiczenie 1. Analizujemy algorytm wyboru większej z dwóch liczb

1. Korzystając z podanej listy kroków, wykonaj kilkukrotnie algorytm dla różnych wartości zmiennych.
2. Zmodyfikuj listę kroków, aby zrealizować algorytm wyboru mniejszej z dwóch liczb.

Algorytm wyboru większej z dwóch liczb zaprogramujemy w środowiskach programowania Baltie (rys. 1a) i Scratch (rys. 1b) oraz w językach C++ (rys. 3a) i Python (rys. 3b). W algorytmie występuje sytuacja warunkowa , dlatego zastosujemy również instrukcję warunkową.

Baltie

Rys. 1a. *Fragment programu realizującego algorytm wyboru większej z dwóch liczb (Baltie) – ćwiczenie 2.*

Scratch

Rys. 1b. *Fragment programu realizującego algorytm wyboru większej z dwóch liczb (Scratch) – ćwiczenie 2.*

Ćwiczenie 2. Tworzymy program w środowiskach programowania Baltie lub Scratch wybierający większą z dwóch liczb

1. W wybranym środowisku programowania utwórz program wybierający większą z dwóch różnych liczb wprowadzanych z klawiatury, wyprowadzający na ekran odpowiedni komunikat i wynik (większą liczbę). Skorzystaj z podanej listy kroków.
2. Zapisz program w pliku pod nazwą *Większa z dwóch*.

Uwaga (Scratch): Dodatkowo zmień tło i postać duszka. Duszek powinien odwracać się w lewo lub w prawo, zależnie od tego, którą liczbę wprowadza lub wyprowadza (rys. 2.).

Scratch

Rys. 2. *Przykładowy ekran programu (Scratch) – ćwiczenie 2.*

C++

```
if (a > b)
    cout << "Wieksza jest pierwsza liczba " << a;
else
    cout << "Wieksza jest druga liczba " << b;
```

Rys. 3a. *Fragment programu realizującego algorytm wyboru większej z dwóch liczb (C++) – ćwiczenie 3.*

Python

```
if a > b:
    print("Większa jest pierwsza liczba", a)
else:
    print("Większa jest druga liczba", b)
```

Rys. 3b *Fragment programu realizującego algorytm wyboru większej z dwóch liczb (Python) – ćwiczenie 3.*

Ćwiczenie 3. Piszemy program w języku C++ lub Python wybierający większą z dwóch liczb

1. W wybranym języku programowania utwórz program wybierający większą z dwóch różnych liczb wprowadzanych z klawiatury, wyprowadzający na ekran odpowiedni komunikat i wynik (większą liczbę). Skorzystaj z podanej listy kroków.
2. Zapisz program w pliku pod nazwą *Większa z dwóch*.
3. Porównaj realizację algorytmu w programach utworzonych w środowisku Baltie i języku C++ lub w języku Scratch i języku Python. Co zauważasz? Uzasadnij odpowiedź.

1.2. Wyszukiwanie największego elementu w zbiorze nieuporządkowanym

Podczas rozwiązywania niektórych problemów (z różnych dziedzin nauki i życia codziennego) szukamy w danym zbiorze najmniejszego lub największego elementu, np. najmniejszej liczby, najwyższej osoby.

Chcemy znaleźć najwyższego ucznia wśród uczniów całej klasy.
W jaki sposób można to zrobić, czyli jaki zastosować algorytm wyszukiwania?

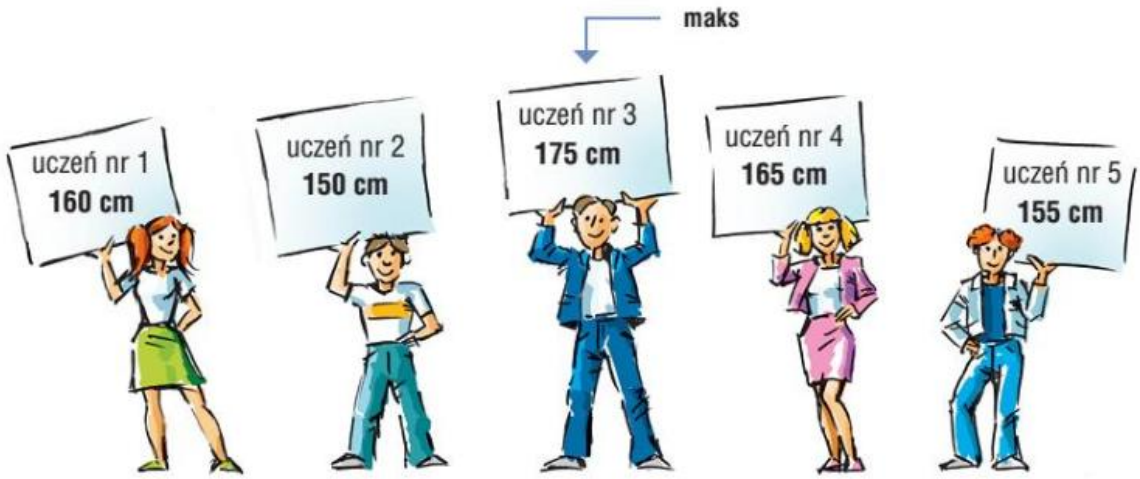
Aby wyszukać największą spośród *n* liczb, porównujemy kolejne liczby ze zbioru z największą znalezioną do tej pory liczbą (zapamiętaną osobno). Na początku algorytmu przyjmujemy, że pierwsza liczba ze zbioru jest największa. Porównania liczb powtarzamy, aż sprawdzimy wszystkie liczby ze zbioru.

Algorytm wyszukiwania największej liczby spośród *n* liczb

Zadanie: Znajdź największą liczbę spośród n liczb.
Dane: Liczba naturalna n oznaczająca liczbę elementów zbioru, n liczb naturalnych wprowadzanych kolejno i zapamiętywanych w zmiennej a.
Wynik: Wartość elementu największego: *maks*.
Lista kroków:

1. Zacznij algorytm.
2. Wprowadź liczbę elementów zbioru: n.
3. Wprowadź wartość pierwszej liczby: a.
4. Przyjmij, że liczbą największą jest pierwsza liczba ze zbioru – zmiennej *maks* przypisz wartość pierwszej liczby: $maks = a$.
5. Wprowadź wartość kolejnej liczby: a.
6. Porównaj kolejną liczbę ze zbioru z *maks*: czy $a > maks$?
7. Jeśli kolejna liczba jest większa od *maks*, to zmiennej *maks* przypisz wartość tej liczby: $maks = a$.
8. Powtarzaj $n - 1$ razy kroki 5., 6. i 7.
9. Wyprowadź wynik: *maks*.
10. Zakończ algorytm.

Algorytm wyszukiwania największej liczby spośród n liczb zastosujemy do znajdowania wzrostu najwyższego z n uczniów klasy (gdzie n to liczba uczniów klasy). Sprawdzimy działanie algorytmu dla $n = 5$ (tabela 1.). Na rysunku 4. podaliśmy wzrost uczniów w centymetrach, czyli kolejne wartości a.



Rys. 4. *Przykład zastosowania algorytmu wyszukiwania największej spośród* n *liczb*

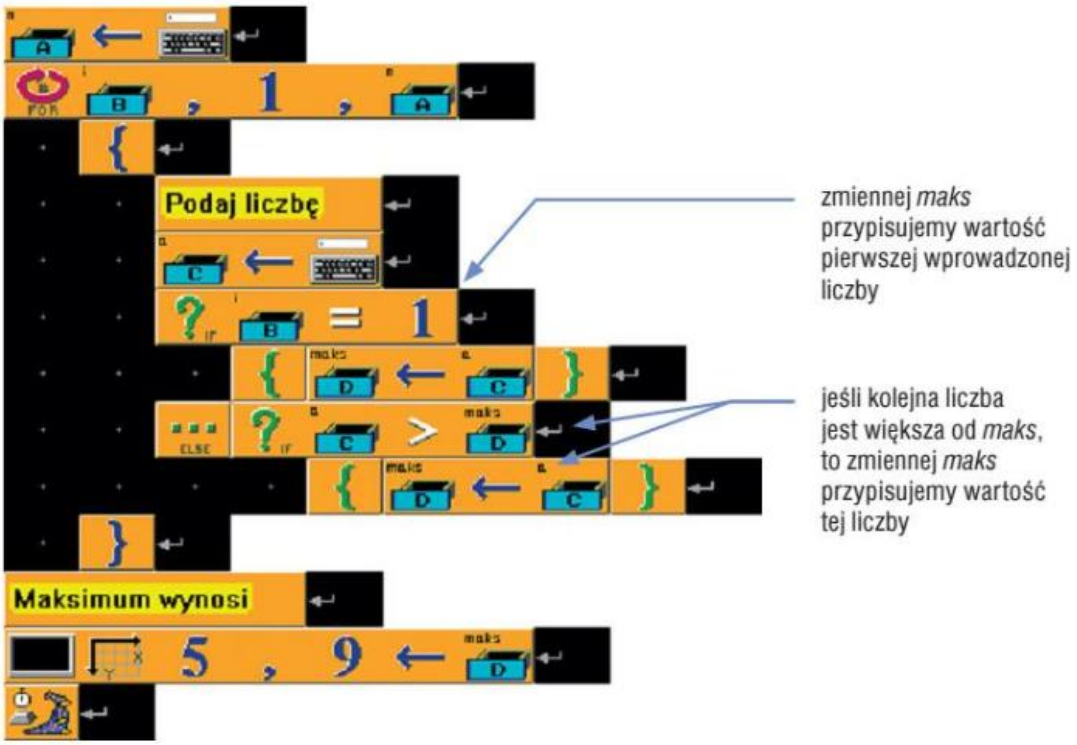
a	czy *a* > *maks*?	rezultat	działanie
wprowadź wzrost pierwszego ucznia: *a* = 160	–	–	zmiennej *maks* przypisz wzrost pierwszego ucznia: *maks* = 160
wprowadź wzrost drugiego ucznia: *a* = 150	porównaj wzrost drugiego ucznia z wartością *maks*: czy 150 > 160?	NIE	pozostaw *maks* bez zmian: *maks* = 160
wprowadź wzrost trzeciego ucznia: *a* = 175	porównaj wzrost trzeciego ucznia z wartością *maks*: czy 175 > 160?	TAK	zmiennej *maks* przypisz wzrost trzeciego ucznia: *maks* = 175
wprowadź wzrost czwartego ucznia: *a* = 165	porównaj wzrost czwartego ucznia z wartością *maks*: czy 165 > 175?	NIE	pozostaw *maks* bez zmian: *maks* = 175
wprowadź wzrost piątego ucznia: *a* = 155	porównaj wzrost piątego ucznia z wartością *maks*: czy 155 > 175?	NIE	pozostaw *maks* bez zmian: *maks* = 175
–	–	–	wzrost najwyższego ucznia to 175 cm.

Tabela 1. *Przykład wykonania algorytmu dla* n = *5*

Ćwiczenie 4. Sprawdzamy działanie algorytmu wyszukiwania największej liczby spośród *n* liczb

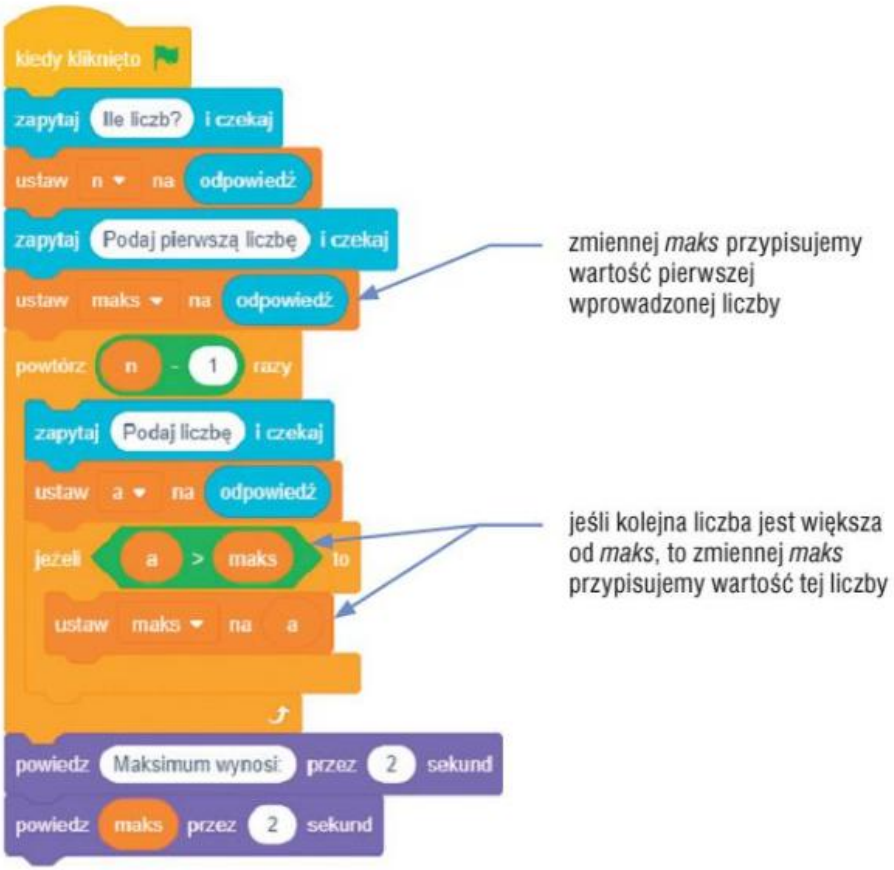
1. Przygotuj odpowiednie pomoce dydaktyczne (rys. 4.) i wykonaj wspólnie z koleżankami i kolegami z klasy algorytm wyszukiwania najwyższego spośród pięciu uczniów.
2. Sprawdź działanie algorytmu wyszukiwania największej liczby:
   a. dla zbioru liczb {70, 160, 155, 180, 185};
   b. dla zbioru liczb {8, -6, 23, 0, -50, 34, -23};
   c. dla dziesięciu wybranych liczb.

Baltie



Rys. 5a. *Program realizujący algorytm wyszukiwania największej liczby w zbiorze n-elementowym (Baltie) – ćwiczenie 5.*

Scratch



Rys. 5b. *Program realizujący algorytm wyszukiwania największej liczby w zbiorze n-elementowym (Scratch) – ćwiczenie 5.*

Algorytm wyszukiwania największej liczby spośród *n* liczb zaprogramujemy w środowiskach programowania Baltie (rys. 5a) i Scratch (rys. 5b), a następnie w językach C++ (rys. 6a) i Python (rys. 6b). W algorytmie powtarzamy polecenia *n* razy, dlatego w każdym z programów zastosujemy odpowiednią instrukcję iteracyjną. Zastosujemy również instrukcję warunkową, ponieważ w algorytmie występują sytuacje warunkowe. Użyjemy również niezbędnych zmiennych.

C++

[*] Maksimum_n.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int main ()
5  {
6      int i, a, n, maks;
7      cout << "Ile liczb? ";
8      cin >> n;
9      for(i = 0; i < n ; i++)
10     {
11         cout << "Podaj liczbe " << i + 1 << ": ";
12         cin >> a;
13         if(i == 0)
14             maks = a;
15         else
16             if(a > maks)
17                 maks = a;
18     }
19     cout << "Maksimum wynosi: " << maks;
20     return 0;
21 }
```

zmiennej *maks* przypisujemy wartość pierwszej wprowadzonej liczby

jeśli kolejna liczba jest większa od *maks*, to zmiennej *maks* przypisujemy wartość tej liczby

Rys. 6a. *Program realizujący algorytm wyszukiwania największej liczby w zbiorze n-elementowym (C++) – ćwiczenie 6.*

Python

Maksimum_n.py - C:\Python\Maksimum_n.py (3.9.0)

File Edit Format Run Options Window Help

```
n = int(input("Ile danych? "))
maks = int(input("Podaj pierwszą liczbę: "))

for i in range(n-1):
    a = int(input("Podaj liczbę: "))
    if i == 0:
        maks = a
    else:
        if a > maks:
            maks = a

print("Maksimum wynosi:", maks)

input("\n\nAby zakończyć, naciśnij Enter")
```

zmiennej *maks* przypisujemy wartość pierwszej wprowadzonej liczby

jeśli kolejna liczba jest większa od *maks*, to zmiennej *maks* przypisujemy wartość tej liczby

Rys. 6b. *Program realizujący algorytm wyszukiwania największej liczby w zbiorze n-elementowym (Python) – ćwiczenie 6.*

Ćwiczenie 5. Tworzymy program w środowisku programowania Baltie lub Scratch realizujący algorytm wyszukiwania największej liczby w zbiorze *n*-elementowym

1. Umieść w obszarze roboczym polecenia pokazane na rysunku 5a lub 5b. Zapisz program w pliku pod nazwą *Maksimum_n*.
2. Uruchom program i sprawdź jego działanie dla różnych danych, w tym dla danych pokazanych na rysunku 4. Objaśnij działanie programu, m.in. uzasadnij użycie poszczególnych poleceń zgodnie z podaną listą kroków.
3. Odpowiedz na pytanie: *Dlaczego w pętli jest* n – 1 *powtórzeń?* Uzasadnij odpowiedź.

Ćwiczenie 6. Zapisujemy w języku C++ lub Python algorytm wyszukiwania największej liczby w zbiorze *n*-elementowym

1. Programy pokazane na rysunkach 6a i 6b realizują algorytm wyszukiwania największego elementu w zbiorze *n*-elementowym. Przepisz wybrany program i zapisz go w pliku pod nazwą *Maksimum_n*.
2. Uruchom program. Sprawdź jego działanie dla różnych danych, w tym dla danych pokazanych na rysunku 4. Objaśnij działanie programu, m.in. uzasadnij użycie poszczególnych poleceń zgodnie z podaną listą kroków.
3. Odpowiedz na pytanie: *Dlaczego w pętli jest* n – 1 *powtórzeń?* Uzasadnij odpowiedź.
4. Porównaj realizację algorytmu w programach utworzonych w środowisku Baltie i języku C++ lub w języku Scratch i języku Python. Co zauważasz? Uzasadnij odpowiedź.

1.3. Wyszukiwanie danego elementu w zbiorze nieuporządkowanym

Szukamy na półce sklepowej butów o rozmiarze 37. Ktoś jednak poprzestawiał buty i nie są one ustawione według rozmiarów. Jak sprawdzić, czy buty o rozmiarze 37 stoją na półce i na którym miejscu?

Aby stwierdzić, czy dany element istnieje w nieuporządkowanym zbiorze *n*-elementowym, musimy sprawdzić każdy z elementów zbioru, ponieważ może on znajdować się w dowolnym miejscu (nawet być ostatnim elementem). W szczególnym przypadku szukanego elementu może w zbiorze w ogóle nie być.

W przypadku poszukiwania na półce pary butów wśród par, które nie są uporządkowane według rozmiarów, będziemy po kolei sprawdzać każdą parę, aby znaleźć buty o rozmiarze 37 (zakładamy, że na półce jest jedna para butów o tym rozmiarze).

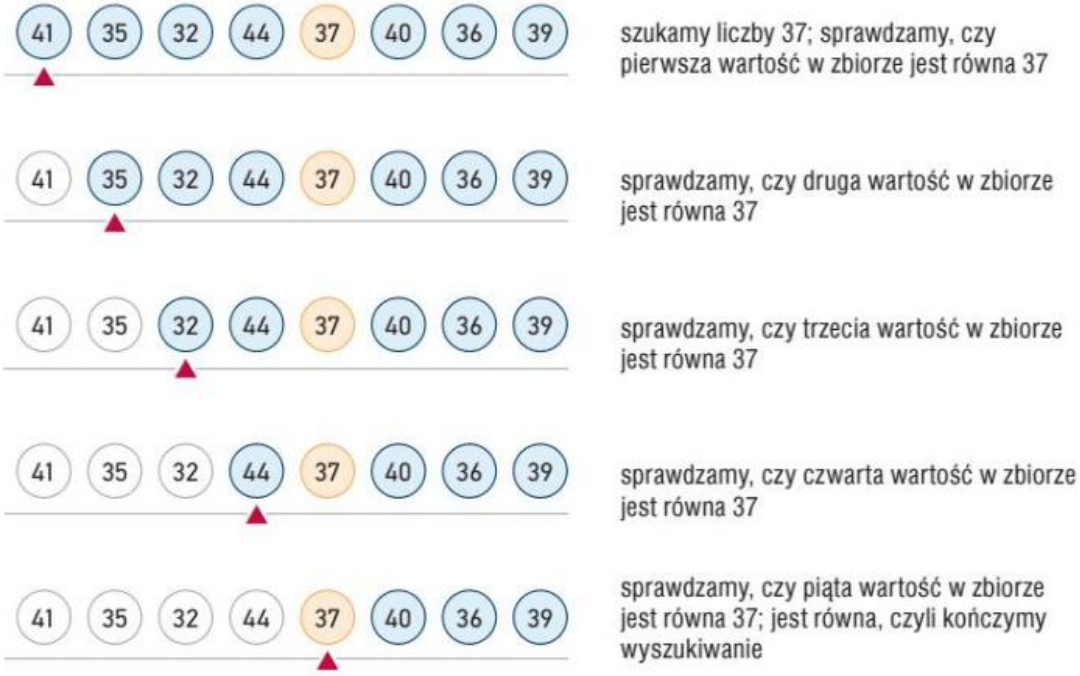
Opis algorytmu wyszukiwania elementu w zbiorze nieuporządkowanym (na przykładzie ośmiu liczb)

Dany jest nieuporządkowany zbiór ośmiu liczb {41, 35, 32, 44, 37, 40, 36, 39}. Poszukiwana liczba to 37. Zakładamy, że w zbiorze jest tylko jedna taka liczba. Przeglądamy wszystkie kolejne elementy zbioru, stosując przeszukiwanie liniowe.

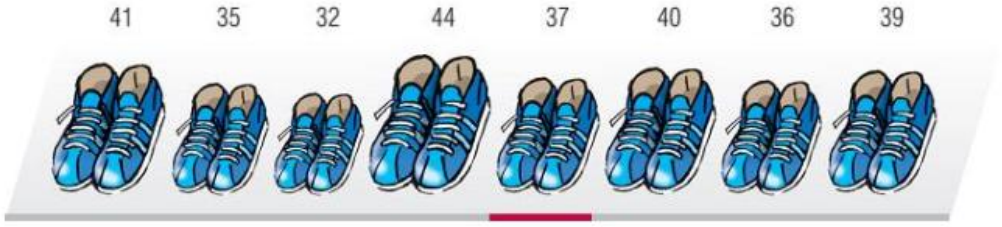
Zaczynamy od sprawdzenia, czy pierwsza liczba (na rysunku 7. zaznaczona strzałką) jest szukaną liczbą. Jeśli tak, kończymy poszukiwanie. Jeżeli nie – sprawdzamy kolejną liczbę. Algorytm kończy się, gdy znajdziemy liczbę równą szukanej, a jeśli jej nie ma – gdy dojdziemy do końca zbioru.

Uwaga

W sytuacji, kiedy szukany element występuje w zbiorze więcej niż jeden raz, algorytm kończy działanie po natknięciu się na niego po raz pierwszy.



Rys. 7. *Przykład wyszukiwania liczby w zbiorze nieuporządkowanym*



Rys. 8. *Zastosowanie algorytmu wyszukiwania elementu w zbiorze nieuporządkowanym – szukanie na półce butów o danym rozmiarze*

W algorytmie wyszukiwania elementu w zbiorze nieuporządkowanym szukamy danego elementu i równocześnie jego położenia w tym zbiorze.

Ćwiczenie 7. Sprawdzamy działanie algorytmu wyszukiwania liczby w zbiorze nieuporządkowanym

Przygotuj odpowiednie pomoce dydaktyczne (rys. 7.) i przedstaw wspólnie z koleżankami i kolegami z klasy algorytm wyszukiwania liczby w zbiorze nieuporządkowanym.

Wskazówka: Możesz zapisać liczby na kartkach i układać je na stole lub podłodze.

2. Wyszukiwanie elementu w zbiorze uporządkowanym metodą połowienia

Szukamy na półce sklepowej butów o rozmiarze 37. Buty są uporządkowane według rozmiarów (od najmniejszego do największego), zaczynając od lewej strony. Jak sprawdzić, czy buty o rozmiarze 37 stoją półce i na którym miejscu?

Wiemy, że buty zostały ułożone rosnąco według rozmiarów, zaczynając od lewej strony (rys. 10.). Szukamy butów o rozmiarze 37. Czy przeglądamy półkę od samego początku? Raczej nie. Zwykle odruchowo sprawdzamy, czy szukanych butów nie ma pośrodku. Jeśli trafimy na buty o innym rozmiarze, np. 39, to nie przeszukujemy prawej połowy półki, tylko będziemy szukać po lewej stronie – postępujemy zgodnie z **algorytmem wyszukiwania przez połowienie**.

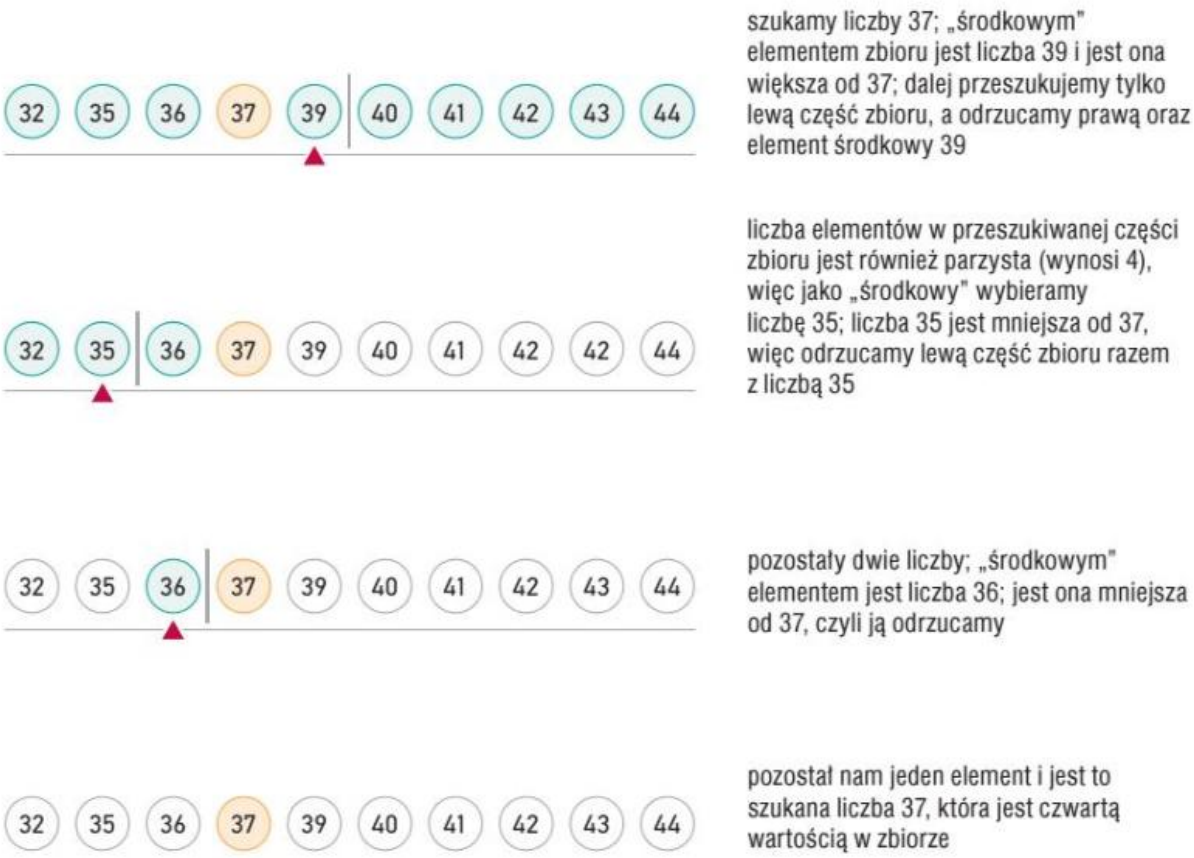
Algorytm wyszukiwania przez połowienie jest przykładem metody „dziel i zwyciężaj". Polega ona na dzieleniu przeszukiwanego zbioru na dwie części i zawężeniu przeszukiwania do jednej z tych części. Metoda podziału pomaga szybko znaleźć poszukiwany element, czyli zwyciężyć.

Opis algorytmu wyszukiwania przez połowienie (na przykładzie dziesięciu liczb)

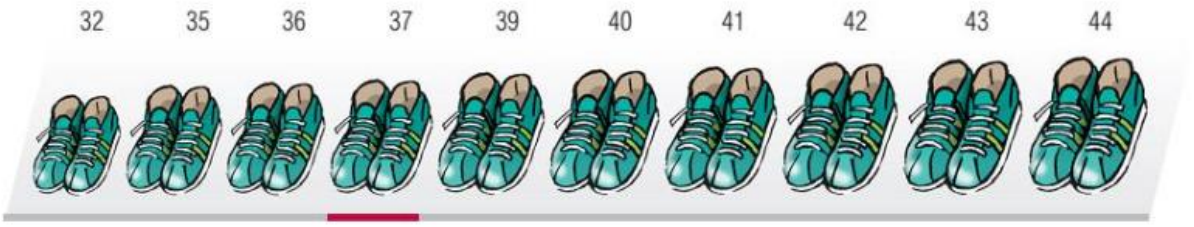
Dany jest uporządkowany w kolejności od najmniejszej do największej zbiór dziesięciu liczb {32, 35, 36, 37, 39, 40, 41, 42, 43, 44}. Poszukiwana liczba to 37. Jeżeli w zbiorze jest kilka takich samych wartości, algorytm znajdzie pozycję jednego z tych elementów (nie jest określone którego).

Dzielimy zbiór liczb na dwie części, rozdzielone środkowym elementem. Zaczynamy od sprawdzenia tego elementu (liczby zaznaczonej strzałką). W przypadku, gdy liczba elementów zbioru jest parzysta, spośród dwóch środkowych elementów (w naszym przypadku liczb 39 i 40) jako „środkowy" wybieramy ten, który jest pierwszy w kolejności (czyli 39).

Jeśli środkowy element jest szukaną liczbą, kończymy wyszukiwanie. Jeśli środkowy element jest większy od poszukiwanej liczby, przeszukujemy podzbiór na lewo od elementu środkowego. Jeśli środkowy element jest mniejszy od poszukiwanej liczby, przeszukujemy podzbiór na prawo od elementu środkowego.



Rys. 9. *Przykład wyszukiwania liczby w zbiorze uporządkowanym metodą połowienia*



Rys. 10. *Zastosowanie algorytmu wyszukiwania elementu w zbiorze uporządkowanym metodą połowienia – szukanie na półce butów o danym rozmiarze*

W algorytmie wyszukiwania elementu w zbiorze uporządkowanym szukamy danego elementu i równocześnie jego położenia w tym zbiorze.

Ćwiczenie 8. Sprawdzamy działanie algorytmu wyszukiwania liczby w zbiorze uporządkowanym metodą połowienia

Przygotuj odpowiednie pomoce dydaktyczne (rys. 9.) i wykonaj wspólnie z koleżankami i kolegami z klasy algorytm wyszukiwania liczby w zbiorze uporządkowanym metodą połowienia.

Wskazówka: Możesz zapisać liczby na kartkach i układać je na stole lub podłodze.

3. Stosowanie metody połowienia do gry w zgadywanie liczby

Algorytm wyszukiwania przez połowienie wykorzystuje się w zabawie w zgadywanie liczby. Zabawa polega na odgadywaniu liczby z określonego zakresu. Jedna osoba wybiera liczbę, a druga osoba ją odgaduje. Osoba zgadująca podaje liczbę, a druga osoba udziela odpowiedzi: „za mała" lub „za duża". W ten sposób osoba zgadująca dzieli zbiór liczb, odrzucając część, w której są liczby większe lub mniejsze, aż odgadnie właściwą liczbę.

Ćwiczenie 9. Gramy w zgadywanie liczby

Zagraj z kolegą lub z koleżanką z klasy w zgadywanie liczby. Przyjmijcie zbiór liczb naturalnych z zakresu od 1 do 100.

Algorytm gry w zgadywanie liczby

Zadanie: Odgadnij liczbę w zbiorze liczb naturalnych z zakresu od 1 do 100, stosując algorytm gry w zgadywanie liczby.
Dane: Szukana liczba pamiętana w zmiennej *szukana*, liczby naturalne z zakresu od 1 do 100 podawane przez gracza i zapamiętywane kolejno w zmiennej *liczba*.
Wynik: Odgadnięta liczba: *szukana*.
Lista kroków:

1. Zacznij algorytm.
2. Zmiennej *szukana* przypisz liczbę, którą ma odgadnąć gracz.
3. Wprowadź liczbę podawaną przez gracza do zmiennej *liczba*.
4. Jeżeli *liczba* = *szukana*, idź do kroku 7.
5. Jeżeli *liczba* > *szukana*, wyświetl na ekranie napis „Za duża"; w przeciwnym przypadku wyświetl na ekranie napis „Za mała".
6. Idź do kroku 3.
7. Wyprowadź komunikat „Gratulacje! Odgadnięta liczba to: " i wartość zmiennej *szukana*.
8. Zakończ algorytm.

Algorytm gry w zgadywanie liczby zaprogramujemy w środowiskach programowania Baltie i Scratch, a następnie w języku C++ i języku Python.

Na początku programu należy zmiennej *szukana* przypisać wartość, którą gracz ma odgadnąć, oraz wprowadzać pierwszą liczbę podawaną przez gracza (kolejne będą podawane w pętli).

W algorytmie powtarzamy polecenia, dopóki wartość zmiennej *liczba* jest różna od wartości zmiennej *szukana*, czyli dopóki nie odgadniemy liczby równej szukanej, dlatego w programie wykorzystamy odpowiednią instrukcję iteracyjną. W języku Scratch zastosujemy polecenie **powtarzaj aż** (rys. 11b), a w środowisku programowania Baltie (rys. 11a) oraz w językach C++ (rys. 12a) i Python (rys. 12b) – instrukcję `while`. Zastosujemy również instrukcję warunkową i użyjemy niezbędnych zmiennych.

Ćwiczenie 10. Tworzymy w środowisku programowania Baltie lub Scratch grę w zgadywanie liczby

1. Zaprogramuj grę w zgadywanie liczby zgodnie z podaną listą kroków. W programie zastosuj polecenie przypisania wartości zmiennej *szukana* wartości zmiennej *liczba* wprowadzanej z klawiatury po uruchomieniu programu. Objaśnij działanie instrukcji `while` (rys. 11a) lub **powtarzaj aż** (rys. 11b).
2. Zapisz program w pliku pod nazwą *Zgadnij liczbę*.
3. Uruchom grę i sprawdź jej działanie dla różnych danych (wartości zmiennej *szukana*).
4. Zagraj z kolegą lub koleżanką w zgadywanie liczby. Wpisuj szukaną liczbę tak, aby przeciwnik jej nie widział.

Wskazówki:

Scratch
- Nie pokazuj wartości zmiennej *szukana* na ekranie.
- Zmień tło i postać duszka (rys. 13.).
- Dodatkowo po wyświetleniu napisu „Gratulacje! Odgadnięta liczba to" i wartości zmiennej *szukana* duszek ma zagrać w koszykówkę. W tym celu należy powtarzać wielokrotnie zmianę kostiumu.

Ćwiczenie 11. Zapisujemy w języku C++ lub Python algorytm gry w zgadywanie liczby

1. Zaprogramuj grę w zgadywanie liczby zgodnie z podaną listą kroków. W programie zastosuj polecenie przypisania wartości zmiennej *szukana* wartości zmiennej *liczba* wprowadzanej z klawiatury po uruchomieniu programu. Objaśnij działanie instrukcji `while` (rys. 12a lub 12b).
2. Zapisz program w pliku pod nazwą *Zgadnij liczbę*.
3. Uruchom grę i sprawdź jej działanie dla różnych danych (wartości zmiennej *szukana*).
4. Zagraj z kolegą lub koleżanką w zgadywanie liczby. Wpisuj szukaną liczbę tak, aby przeciwnik jej nie widział.
5. Porównaj realizację algorytmu w programach utworzonych w środowisku Baltie i języku C++ lub w języku Scratch i języku Python (w tym działanie instrukcji iteracyjnej). Co zauważasz? Uzasadnij odpowiedź.

Baltie

***Rys. 11a**. Fragment programu realizującego algorytm gry w zgadywanie liczby (Baltie) – ćwiczenie 10.*

Scratch

***Rys. 11b**. Fragment programu realizującego algorytm gry w zgadywanie liczby (Scratch) – ćwiczenie 10.*

C++

```
while (liczba != szukana)
{
    if (liczba > szukana)
        cout << "Za duza" << endl;
    else
        cout << "Za mala" << endl;
    cout << "Podaj liczbe od 1 do 100: ";
    cin >> liczba;
}
```

***Rys. 12a**. Fragment programu realizującego algorytm gry w zgadywanie liczby (C++) – ćwiczenie 11.*

Python

```
while liczba != szukana:
    if liczba > szukana:
        print("Za duża")
    else:
        print("Za mała")
    liczba = int(input("Podaj liczbę od 1 do 100: "))
```

***Rys. 12b**. Fragment programu realizującego algorytm gry w zgadywanie liczby (Python) – ćwiczenie 11.*

Scratch

Rys. 13. *Przykładowy ekran programu realizującego algorytm gry w zgadywanie liczby (Scratch) – ćwiczenia 10. i 12.*

Gra będzie ciekawsza, jeśli gracze nie będą znali wcześniej szukanej liczby. Prawie w każdym języku programowania występuje polecenie umożliwiające generowanie liczb losowych.

Ćwiczenie 12. Modyfikujemy grę w zgadywanie liczby w środowisku programowania Baltie lub Scratch

1. Otwórz program *Zgadnij liczbę* zapisany w ćwiczeniu 10.
2. Zmodyfikuj grę, tak aby komputer wybierał losowo szukaną liczbę z przedziału od 1 do 100. Dodaj zliczanie wykonanych prób: utwórz nową zmienną *licznik*.
   Na początku programu ustaw jej wartość na 1, a w pętli zwiększaj o 1. Po napisie „Gratulacje! Odgadnięta liczba to" i wartości zmiennej *szukana* wyświetl napis „Liczba wykonanych prób to" i wartość zmiennej *licznik*.
3. Zapisz program w pliku pod nazwą *Zgadnij liczbę_los*.

Wskazówki:

Baltie Aby w środowisku programowania Baltie wygenerować liczby losowe z danego przedziału, należy zastosować element **Liczba losowa**  .

Na przykład  generuje liczby całkowite z przedziału <0, 100), czyli liczby: 0, 1, 2, …, 99.

W poleceniu  zmiennej *szukana* przypisujemy wartość losową od 1 do 100.

Scratch Aby w języku Scratch wygenerować liczby losowe z danego przedziału, należy zastosować element **losuj liczbę** z grupy **Wyrażenia**, np.

– wygeneruje losowo liczbę naturalną z przedziału ⟨1, 100⟩.

W poleceniu 

zmiennej *szukana* przypisujemy wartość losową od 1 do 100.

Ćwiczenie 13. Modyfikujemy grę w zgadywanie liczby w języku C++ lub Python

1. Otwórz program *Zgadnij liczbę* zapisany w ćwiczeniu 11.
2. Zmodyfikuj grę, tak aby komputer wybierał losowo szukaną liczbę z przedziału od 1 do 100.
3. Zapisz program w pliku pod tą samą nazwą.

Wskazówki:

C++ W języku C++ liczby losowe generuje funkcja `rand()`.
Na przykład `rand() % 100 + 1` – wygeneruje losowo liczbę naturalną z przedziału ⟨1, 100⟩.
Na początku programu musimy dołączyć bibliotekę, w której zdefiniowano funkcję `randint()`.

```
#include <stdlib.h>
...
szukana = rand() % 100 + 1;
```

Python W języku Python liczby losowe generuje funkcja `randint()`.
Na przykład `randint(1, 100)` – wygeneruje losowo liczbę naturalną z przedziału ⟨1, 100⟩.
Na początku programu musimy dołączyć bibliotekę, w której zdefiniowano funkcję `randint()`.

```
from random import randint
szukana = randint(1, 100)
```

4. Porządkowanie elementów zbioru metodą przez wybieranie

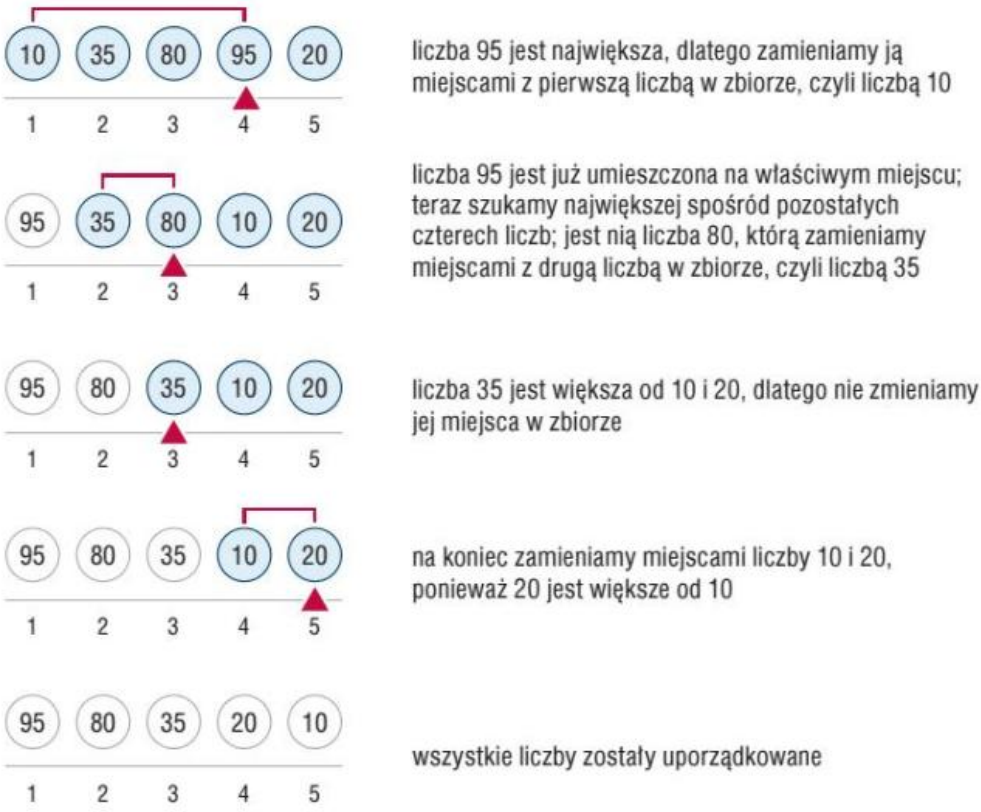
Czasem porządkujemy (w terminologii informatycznej – **sortujemy**) zbiory przedmiotów, np. ustawiamy książki na półce – od największej do najmniejszej lub alfabetycznie według nazwisk autorów, ustalamy kolejność wyświetlania plików według nazw lub dat utworzenia. Na danych uporządkowanych (posortowanych) szybciej wykonuje się różne operacje (np. wyszukiwanie).

{ Chcemy uporządkować wszystkich uczniów od najwyższego do najniższego.
Czy można do tego celu wykorzystać algorytm wyszukiwania najwyższego ucznia? }

Istnieją różne algorytmy wyszukiwania i porządkowania. Do porządkowania elementów często stosujemy metodę **porządkowania przez wybieranie**.

Aby uporządkować elementy od największego do najmniejszego, stosujemy algorytm wyboru elementu największego. Jeśli chcemy uporządkować elementy od najmniejszego do największego, stosujemy algorytm wyboru elementu najmniejszego.

Porządkowanie zbioru liczb metodą przez wybieranie polega na wyszukiwaniu w nim liczby największej (dla porządku malejącego) lub najmniejszej (dla porządku rosnącego), zamienieniu jej miejscami z liczbą, która jest aktualnie na początku zbioru, a następnie powtarzaniu tych czynności z pominięciem liczb już uporządkowanych (rys. 14.).



Rys. 14. *Przykład porządkowania liczb metodą przez wybieranie – od największej do najmniejszej*

Algorytm porządkowania metodą przez wybieranie wykorzystamy do ustawienia uczniów od najwyższego do najniższego.

Algorytm porządkowania metodą przez wybieranie

Zadanie: Uporządkuj dziesięciu uczniów według wzrostu (od ucznia najwyższego do najniższego), stosując sortowanie przez wybieranie.
Dane: Dziesięciu uczniów (ustawionych w przypadkowej kolejności).
Wynik: Dziesięciu uczniów ustawionych w kolejności od najwyższego do najniższego.
Lista kroków:

1. Zacznij algorytm.
2. Znajdź najwyższego ucznia. Zapamiętaj, w którym miejscu stoi.
3. Zamień miejscami ucznia najwyższego z uczniem stojącym na pierwszym miejscu.
4. Powtarzaj kroki 2. i 3. (z pominięciem uczniów już ustawionych na miejscach początkowych), aż uporządkujesz wszystkich uczniów.
5. Zakończ algorytm.

Ćwiczenie 14. Porządkujemy uczniów według wzrostu metodą przez wybieranie

Wybierz z klasy dziesięciu uczniów. Ustaw wybranych uczniów według wzrostu (od najwyższego do najniższego). Postępuj zgodnie z podaną listą kroków.

Ćwiczenie 15. Porządkujemy książki metodą przez wybieranie

Napisz listę kroków algorytmu porządkowania książek od najgrubszej do najcieńszej.

5. Porządkowanie elementów zbioru metodą przez zliczanie

Algorytm porządkowania przez zliczanie pokażemy na przykładzie porządkowania liczb całkowitych. Przed rozpoczęciem porządkowania musimy znać najmniejszą i największą wartość w zbiorze.

Porządkowanie metodą przez zliczanie polega na zliczaniu wystąpień poszczególnych wartości w zbiorze, np. ile mamy jedynek, dwójek, trojek, czwórek, piątek, …, dwudziestek – w przypadku zbioru liczb zawierających wartości od 1 do 20. Następnie wypisujemy daną liczbę tyle razy, ile wynosi liczba jej wystąpień w zbiorze. W ten sposób otrzymujemy uporządkowany zbiór.

Opis algorytmu porządkowania elementów zbioru metodą przez zliczanie (na przykładzie siedmiu liczb)

Dany jest zbiór liczb {5, 6, 5, 2, 3, 5, 3}, który zawiera liczby całkowite z przedziału od 2 do 6 (czyli najmniejszą liczbą jest liczba 2, a największą 6).

Do zliczania wystąpień wszystkich liczb z tego zakresu musimy przygotować pięć liczników do zliczania wystąpień liczb: 2, 3, 4, 5 i 6. Na rysunku 15. liczniki mają formę pudełek. Na początku wszystkie pudełka liczników są puste (mają wartość 0), ale gdy zaczniemy przeglądać zbiór, wartość liczników będzie się zmieniać.

Liczniki przygotowujemy dla wszystkich liczb z przedziału od 2 do 6, nawet dla liczb z tego przedziału, których nie ma w naszym zbiorze. Algorytm porządkowania liczb metodą przez zliczanie zaprogramujemy w tematach 8. (C++) i 9. (Python).

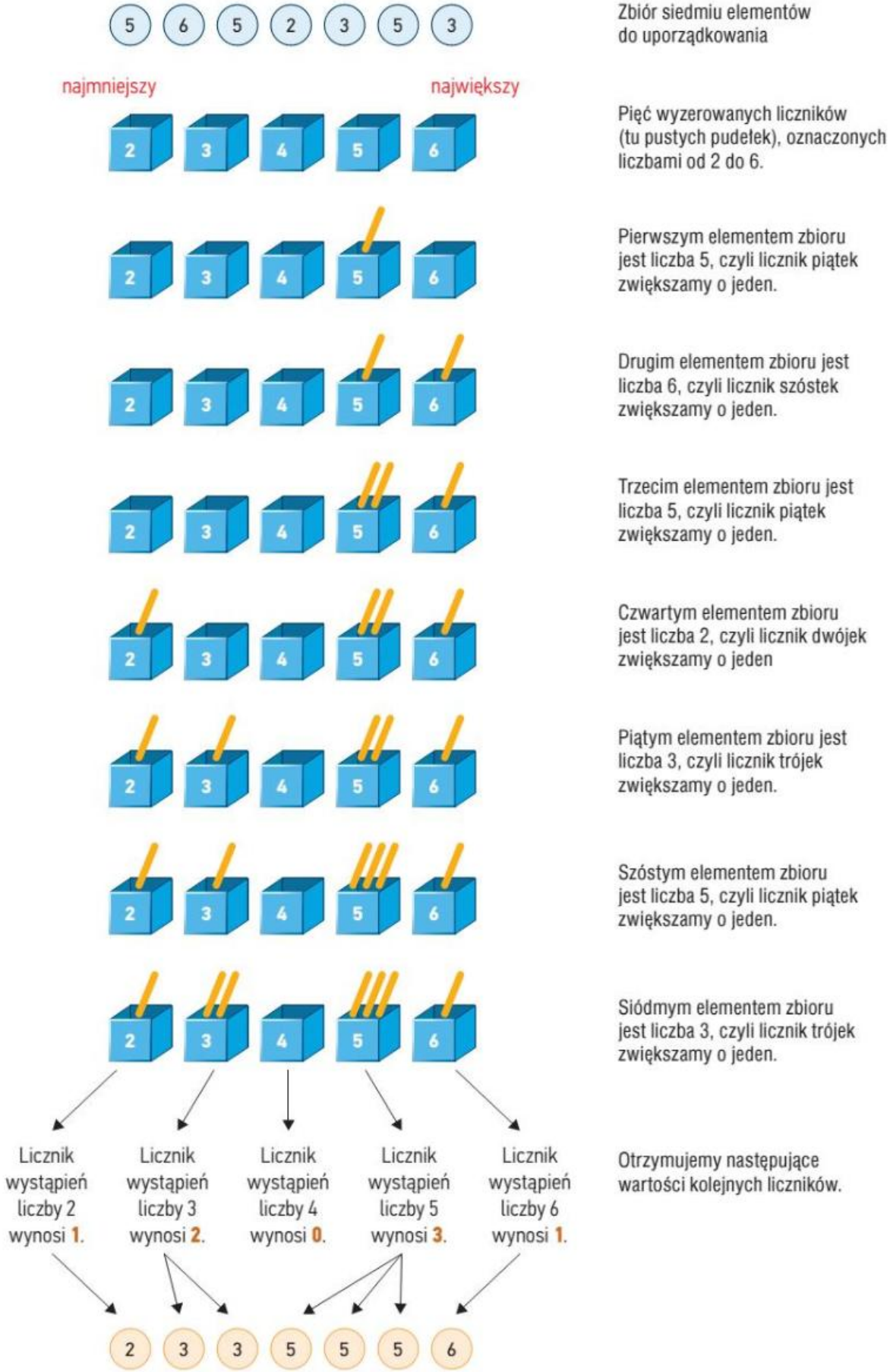
Ćwiczenie 16. Porządkujemy liczby od najmniejszej do największej metodą przez zliczanie

Przygotuj odpowiednie pomoce dydaktyczne (pudełka lub kubeczki i patyczki lub ołówki, a następnie wykonaj wspólnie z koleżankami i kolegami z klasy algorytm porządkowania liczb metodą przez zliczanie dla zbioru składającego się z dziesięciu elementów: {–2, 5, 2, 0, 4, –2, 4, –1, –2, 5}. Liczby uporządkuj od najmniejszej do największej.

Wskazówki:

- Pudełek (kubeczków) ma być tyle, ile jest liczb całkowitych z zakresu ⟨*najmniejszy*, *największy*⟩, gdzie najmniejszy to najmniejsza liczba w zbiorze, a *największy* – *największa*.
- Zbiór zaczynamy przeglądać od pierwszego elementu i kolejno wrzucamy patyczki (ołówki) do kubeczków oznaczonych daną liczbą.

Aby zbiór liczb był uporządkowany od największej do najmniejszej, wystarczy zbiór liczników przejrzeć odwrotnie – od prawej strony do lewej.



Rys. 15. *Przykład porządkowania liczb metodą przez zliczanie*

Warto zapamiętać

- Wybrany element w zbiorze nieuporządkowanym możemy znaleźć, korzystając z algorytmu przeszukiwania liniowego.
- Wybrany element w zbiorze uporządkowanym możemy znaleźć, korzystając z algorytmu wyszukiwania przez połowienie, który jest przykładem realizacji metody **dziel i zwyciężaj**.
- Zbiór elementów możemy uporządkować, stosując algorytmy porządkowania, np. metodą przez wybieranie lub metodą przez zliczanie.
- W porządkowaniu metodą przez wybieranie wykorzystujemy algorytm wyszukiwania elementu (najmniejszego lub największego) w zbiorze nieuporządkowanym.

Pytania i polecenia

1. Wyjaśnij na przykładzie, w jaki sposób znajduje się największą spośród siedmiu liczb.
2. Dlaczego w programach na rysunkach 5a i 5b oraz na rysunkach 6a i 6b wprowadzaną wartość przypisujemy najpierw zmiennej *maks*?
3. Dlaczego w programie wyszukującym liczbę największą stosujemy instrukcję iteracyjną i warunkową?
4. Na czym polega algorytm wyszukiwania przez połowienie?
5. Jakie może być zastosowanie algorytmu wyszukiwania przez połowienie? Pokaż na przykładzie.
6. Dysponujemy listą dwustu uczestników maratonu. Lista jest uporządkowana według numerów startowych, ale numery nie są kolejnymi liczbami naturalnymi. Jak szybko znaleźć na tej liście nazwisko zawodnika o konkretnym numerze?
7. Jaką liczbę najlepiej podać jako pierwszą w grze w zgadywanie liczby w przypadku zbioru liczb od 1 do 100? Uzasadnij odpowiedź.
8. Na czym polega algorytm porządkowania metodą przez wybieranie?
9. Dlaczego do ustawienia uczniów według wzrostu zastosowaliśmy metodę porządkowania przez wybieranie? Uzasadnij odpowiedź.
10. W jaki sposób porządkuje się liczby metodą przez zliczanie? Wyjaśnij na przykładzie.
11. Dlaczego w porządkowaniu metodą przez zliczanie należy przygotować liczniki dla wszystkich liczb z danego zakresu, nawet jeśli niektóre liczniki pozostaną zerowe? Uzasadnij odpowiedź.
12. Wskaż przynajmniej dwie różnice między porządkowaniem metodą przez wybieranie a metodą przez zliczanie.

Zadania

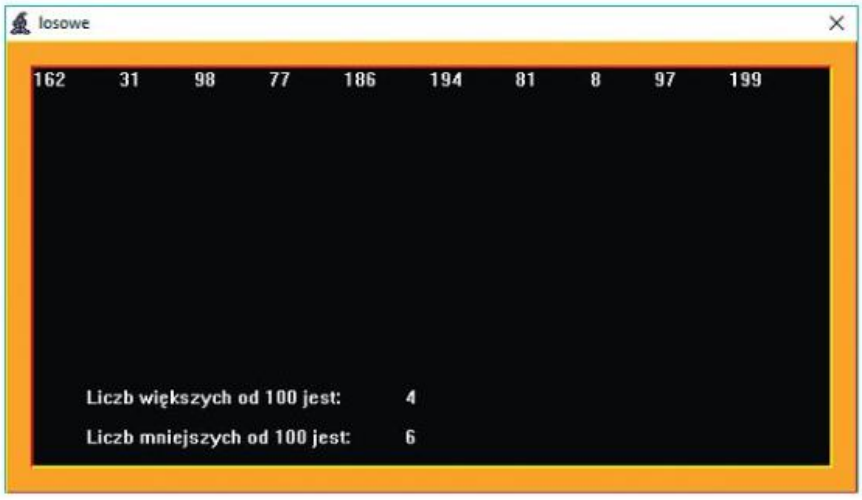
Uwagi:

- Listy kroków możesz zapisać w zeszycie przedmiotowym lub w edytorze tekstu.
- Schematy blokowe możesz rysować w zeszycie przedmiotowym, w edytorze tekstu z wykorzystaniem **Kształtów**, w edytorze grafiki **Paint** lub w wybranym darmowym programie pobranym z Internetu.

1. Narysuj schemat blokowy algorytmu znajdowania większej z dwóch liczb.
2. Zmodyfikuj program zapisany w ćwiczeniu 4., uwzględniając przypadek, gdy wprowadzone z klawiatury liczby są równe. Zapisz program w pliku pod tą samą nazwą.

3. Utwórz program do znajdowania największej z trzech liczb. Zapisz program w pliku pod nazwą *Największa z trzech*.
   Wskazówka: Pamiętaj, aby po sprawdzeniu, czy pierwsza liczba jest większa od drugiej, sprawdzić, czy wybrana z pierwszego porównania liczba jest większa od trzeciej.
4. Na podstawie listy kroków pokazanej w punkcie 1. tego tematu napisz listę kroków algorytmu znajdowania najmniejszej liczby spośród dziesięciu liczb. Sprawdź działanie algorytmu dla danych: 5, 78, 15, 0, 1, 100, 155, 50, 22, 30.
5. Zmodyfikuj program *Maksimum_n* zapisany w ćwiczeniu 5. lub 6. tak, aby wyszukiwał najmniejszy element ze zbioru *n*-elementowego. Zapisz program w pliku pod nazwą *Minimum_n*.
6. Dany jest uporządkowany zbiór liczb: 5, 8, 15, 20, 25, 30, 35, 70, 88, 90, 100, 120. Poszukiwana liczba to 70. Przedstaw na rysunku (podobnym do rys. 9.) sposób znalezienia liczby 70.
7. Dany jest zbiór liczb: 12, 3, 8, 2,12, 5, 8, 4, 10, 4. Przedstaw na rysunku (podobnym do rys. 15.) uporządkowanie tego zbioru metodą przez zliczanie. Ile liczników musisz przygotować? Ile liczników pozostało pustych?
8. Utwórz w środowisku programowania Baltie lub Scratch program, który będzie generował losowo dziesięć liczb z zakresu 1-200 i zliczał oddzielnie liczby większe od 100 i mniejsze 100. Zastosuj trzy zmienne: *liczba*, *licznik1* i *licznik2*. Liczby powinny być wyświetlane na ekranie w trakcie ich generowania. Na koniec powinny zostać wyświetlone wylosowane liczby, odpowiednie komunikaty i wyniki (rys. 16a i 16b). Zapisz program w pliku pod nazwą *Losowe*.

Baltie



Rys. 16a. *Ekran z danymi i wynikami (Baltie) – zadanie 8.*

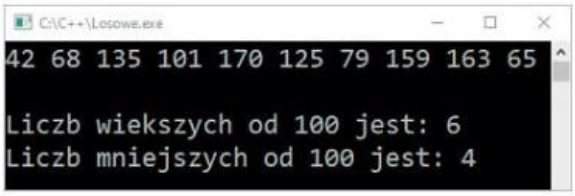
Scratch

Rys. 16b. *Ekran wyświetlania danych i wyników (Scratch) – zadanie 8.*

Dla zainteresowanych

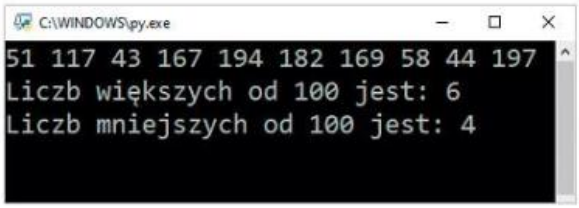
9. Napisz listę kroków i zbuduj schemat blokowy algorytmu znajdowania najniższego z trzech uczniów.
10. Napisz listę kroków algorytmu wyszukiwania przez połowienie. Skorzystaj z opisu metody podanego w punkcie 2. tego tematu.
11. Napisz w języku programowania C++ lub Python program, który będzie generował losowo dziesięć liczb z zakresu 1-200 i zliczał oddzielnie liczby większe od 100 i mniejsze od 100. Zastosuj trzy zmienne: *liczba*, *licznik1* i *licznik2*. Liczby powinny być wyświetlane na ekranie w trakcie ich generowania. Na koniec powinny zostać wyświetlone odpowiednie komunikaty i wyniki. Zapisz program w pliku pod nazwą *Losowe*.

C++



***Rys. 17a**. Ekran z danymi i wynikami (C++) – zadanie 11.*

Python



***Rys. 17b**. Ekran z danymi i wynikami (Python) – zadanie 11.*