Algorytmy w C# - podstawowe

Podstawy Programowania

Utwórz aplikacje konsolową, która będzie zawierała wszystkie poniższe zadania związane z algorytmami w C#. Każde zadanie ma wyświetlać odpowiednie komunikaty, np.:
Podaj liczbę, którą chcesz sprawdzić, czy jest liczbą pierwszą:
14
Liczba 14 nie jest liczbą pierwszą

Zapisz działanie aplikacji w zrzucie ekranu przechwytując cały ekran wraz z zapisanym kodem w tle. Nazwy zrzutów ekranów podpisz następującym schematem: *konsola1.jpg, konsola2.jpg…*

Po każdym zadaniu, ma być czyszczona cała konsola, aby następne zadanie zaczynało na świeżej konsoli. Na koniec spakuj **cały projekt** wraz z **zrzutami ekranu** oraz **dokumentacją**, która zawiera informacje: *w jakim środowisku programistycznym był pisany kod, w jakim języku programowania, na jakim systemie operacyjnym był zapisywany kod*. Archiwum ma nosić nazwę wg. następującego schematu: *imie_nazwisko_algorytmy.zip*

Jako tablicę liczb zastosuj następujący zbiór: **[4, 12, 5, 1, 8, 30, 182, 3, 6, 14, 28, 50, 13]**

Sprawdzanie, czy liczba jest liczbą pierwszą

Liczba pierwsza to taka liczba całkowita większa od 1, która dzieli się wyłącznie przez 1 oraz przez samą siebie.

Zasada działania algorytmu:

1. Jeśli liczba podana jest mniejsza niż 2, nie jest ona liczbą pierwszą
2. Jeżeli liczba jest większa, bądź równa 2, algorytm sprawdza, czy dana liczba dzieli się bez **reszty** przez jakąkolwiek liczbę w zakresie od 2 do **pierwiastka kwadratowego z tej liczby**. Jeśli znajdzie taki dzielnik, liczba nie jest pierwsza. Jeśli nie znajdzie żadnego dzielnika, oznacza to, że liczba jest pierwsza.

Algorytm Euklidesa

Algorytm Euklidesa służy do wyznaczania największego wspólnego dzielnika (NWD) dwóch liczb.

Zasada działania algorytmu:

1. Wejście: Dwie liczby całkowite *a* oraz *b*
2. Sprawdzanie: Dopóki *b* nie jest równe zero:
    1. Obliczamy resztę z dzielenia *a* przez *b*
    2. Przypisujemy wartość *b* do *a*, a resztę z dzielenia tych liczb do *b*

3. Wyjście: Kiedy *b* stanie się równe 0, liczba *a* jest największym wspólnym dzielnikiem.

Szyfr Cezara

Szyfr Cezara to jeden z najstarszych i najprostszych szyfrów, opierający się na przesunięciu liter w alfabecie o stałą liczbę miejsc. Dla przykładu, jeśli przesuniemy litery o 3 miejsca, to 'A' stanie się 'D', 'B' stanie się 'E', i tak dalej. Przy końcu alfabetu, litery są "owijane" wokół, co oznacza, że po 'Z' wracamy do 'A'.

Zasada działania algorytmu:

1. Wejście: Tekst do zaszyfrowania i klucz (liczba), który określa, o ile miejsc w alfabecie przesuniemy litery.
2. Przesunięcie liter: Dla każdej litery w tekście, zamieniamy ją na literę, która jest przesunięta o wartość klucza w alfabecie.
3. Obsługa końców alfabetu: Jeśli przesunięcie wykracza poza 'Z', zaczynamy od początku alfabetu ('A').

Znajdowanie liczb Fibonacciego

Ciąg Fibonacciego to sekwencja liczb, w której każda liczba jest sumą dwóch poprzednich, gdzie pierwsze liczby w ciągu to **0** oraz **1**. Ogólnie dla $n \geq 2$, ciąg Fibonacciego jest definiowany jako:

$$F(n) = F(n-1) + F(n-2)$$

Zasada działania algorytmu:

1. Rozpocznij od dwóch pierwszych elementów
2. Aby obliczyć kolejne elementy, użyj wzoru wypisanego wyżej
3. Program przestaje liczyć dalej, kiedy osiągnie element *n*, który został podany przez użytkownika.

Rozkład liczby na czynniki pierwsze

Rozkład liczby na czynniki pierwsze polega na podzieleniu jej na możliwie najmniejsze liczby pierwsze, które po pomnożeniu dają liczbę wyjściową. Na przykład rozkład liczby 60 na czynniki pierwsze to $60=2^2 \cdot 3 \cdot 5$, ponieważ 2, 3 i 5 są liczbami pierwszymi, a $60=2 \cdot 2 \cdot 3 \cdot 5$.

Zasada działania algorytmu:

1. Wejście: Liczba *n*, którą chcemy rozłożyć na czynniki pierwsze.
2. Algorytm dzieli *n* przez najmniejsze liczby pierwsze, zaczynając od 2, aż liczba *n* stanie się równa 1. Aby zwiększyć efektywność, możemy sprawdzać podzielność tylko do pierwiastka kwadratowego z liczby, ponieważ jeśli liczba ma dzielnik większy od swojego pierwiastka, to drugi dzielnik musi być mniejszy.

Znajdowanie najmniejszego oraz największego elementu w zbiorze

Znajdowanie najmniejszego i największego elementu w zbiorze liczb to bardzo proste, ale użyteczne zadanie. Aby znaleźć te elementy, iterujemy przez zbiór, porównując każdy element z aktualnie najmniejszym i największym elementem, które zainicjujemy odpowiednimi wartościami na początku.

Zasada działania algorytmu:

1. Zakładamy, że pierwszy element zbioru jest zarówno najmniejszy, jak i największy.
2. Dla każdego elementu porównujemy go z bieżącym najmniejszym i największym elementem. Jeśli jest mniejszy od bieżącego najmniejszego, aktualizujemy najmniejszy element. Jeśli jest większy od bieżącego największego, aktualizujemy największy element.
3. Po przetworzeniu wszystkich elementów zwracamy znalezione wartości.

NIE STOSUJEMY METOD WBUDOWANYCH ZWRACAJĄCYCH MAKSYMALNĄ LUB MINIMALNĄ WARTOŚĆ Z TABLICY LICZB!

Sortowanie bąbelkowe

Sortowanie bąbelkowe (ang. *bubble sort*) to prosty algorytm sortowania, który działa, porównując sąsiadujące ze sobą elementy w zbiorze i zamieniając je miejscami, jeśli są w złej kolejności. Proces ten jest powtarzany wielokrotnie, aż cała lista zostanie posortowana.

Zasada działania algorytmu

1. Algorytm przechodzi przez zbiór elementów i porównuje każdą parę sąsiadujących elementów.
2. Jeśli element na lewej pozycji jest większy niż element na prawej pozycji, to zamienia je miejscami.
3. Każda iteracja kończy się "wypchnięciem" największego (lub najmniejszego, w zależności od porządku) elementu na odpowiednie miejsce. Kolejne iteracje poruszają się coraz krócej, ponieważ największe elementy "osiadają" na końcu zbioru.
4. Proces jest powtarzany aż do momentu, gdy w trakcie jednej pełnej iteracji nie dojdzie do żadnej zamiany.

NIE STOSUJEMY ŻADNYCH METOD SORTUJĄCYCH ELEMENTY W TABLICY!

Sortowanie przez wybór

Sortowanie przez wybór (ang. *selection sort*) to prosty algorytm sortowania, który działa poprzez znajdowanie najmniejszego (lub największego) elementu w zbiorze i zamienianie go miejscami z pierwszym elementem. Proces jest powtarzany dla każdego podzbioru elementów, aż cały zbiór zostanie posortowany.

Zasada działania algorytmu:

1. Algorytm przechodzi przez zbiór elementów i znajduje najmniejszy element.
2. Po znalezieniu najmniejszego elementu, zamienia go z pierwszym elementem w zbiorze.
3. Proces jest powtarzany dla pozostałej części zbioru (pomijając już posortowane elementy), czyli algorytm ponownie szuka najmniejszego elementu w pozostałych elementach i zamienia go z następnym w kolejności miejscem.
4. Algorytm kończy pracę, gdy wszystkie elementy zostaną posortowane.

NIE STOSUJEMY ŻADNYCH METOD SORTUJĄCYCH ELEMENTY W TABLICY!

Sortowanie przez wstawianie

Sortowanie przez wstawianie (ang. *insertion sort*) to prosty algorytm sortowania, który działa poprzez budowanie posortowanego zbioru elementów, jeden element na raz. Algorytm wybiera element z nieposortowanej części zbioru i wstawia go na odpowiednie miejsce w części posortowanej.

Zasada działania algorytmu:

1. Lista elementów jest podzielona na dwie części – posortowaną i nieposortowaną. Na początku posortowana część składa się tylko z pierwszego elementu.
2. Algorytm bierze pierwszy element z nieposortowanej części i wstawia go w odpowiednie miejsce w posortowanej części.
3. Powtarza ten proces dla każdego kolejnego elementu z nieposortowanej części, aż cała lista zostanie posortowana.
4. Proces kończy się, gdy wszystkie elementy znajdują się w posortowanej części.

NIE STOSUJEMY ŻADNYCH METOD SORTUJĄCYCH ELEMENTY W TABLICY!