Algorytmy i struktury danych

Aldona Świrad

Projekt

Znalezienie liczby trójelementowych kombinacji liczb z zadanego ciągu, których suma jest równa zadanej liczbie M

Rzeszów, 2021

Spis treści

1. Wstęp

Projekt napisany w języku programowania C++, w środowisku Code::Blocks IDE, .realizowany w ramach przedmiotu „Algorytmy i struktury danych” na kierunku Inżynieria i analiza danych, semestr I, grupa 8.

2. Opis problemu

Problemem do rozwiązania w danym projekcie, jest znalezienie takiej liczby trójelementowych kombinacji liczb z zadanego ciągu, których suma jest równa zadanej liczbie M. Wprowadzając pewien ciąg liczb należy znaleźć wśród nich takie ciągi trzech liczb (ang.triplets), których elementy sumują się do liczby jakiej zażąda użytkownik. Ciągi trójelementowe nie mogą się duplikować. Należy także podać liczbę kombinacji takich ciągów.

2.1. Opis podstaw teoretycznych zagadnienia

Do rozwiązania problemu znalezienia ciągów trójelementowych wykorzystano tablice statyczne i dynamiczne, pętle „for",oraz instrukcje warunkowe „if". Dość problematycznym aspektem okazało się pozbycie powtarzających się ciągów.trzech elementów. Pomocne w tym zakresie było wykorzystanie wektorów, klas, obiektów oraz specjalnych funkcji. W celu sprawnego testowania algorytmu, za pomocą instrukcji warunkowej „switch" stworzony został program z przejrzystym menu, gdzie użytkownik sam może zdecydować o źródle pochodzenia danych wejściowych. Dane wyjściowe są zapisywane w pliku.

2.2. Opis szczegółów implementacji problemu

Do przeprowadzenia testów badających czas pracy algorytmów wykorzystana została funkcja clock z biblioteki time.h. Testowanie algorytmu pod względem danych rozłożone zostało na 3 sposoby, gdzie w każdym pojawia się inna metoda wprowadzania danych wejściowych.

Możliwości generowania danych wejściowych dla algorytmu.

1. Użytkownik sam wpisuje dane wejściowe
2. Dane wejściowe są pobierane z wcześniej przygotowanego pliku.
3. Dane wejściowe są generowane losowo, zapisywane do pliku, a następnie z niego wczytywane.

Pierwszy algorytm można podzielić na 2 części. Pierwsza część zajmuje się znajdowaniem ciągów trójelementowych. Zbudowana jest ona z 3 zagnieżdżonych pętli „for", gdzie każda kolejna przeszukuje po kolei elementy zadanego ciągu. Dzięki temu znajdowane są kombinacje ciągów trzech elementów.

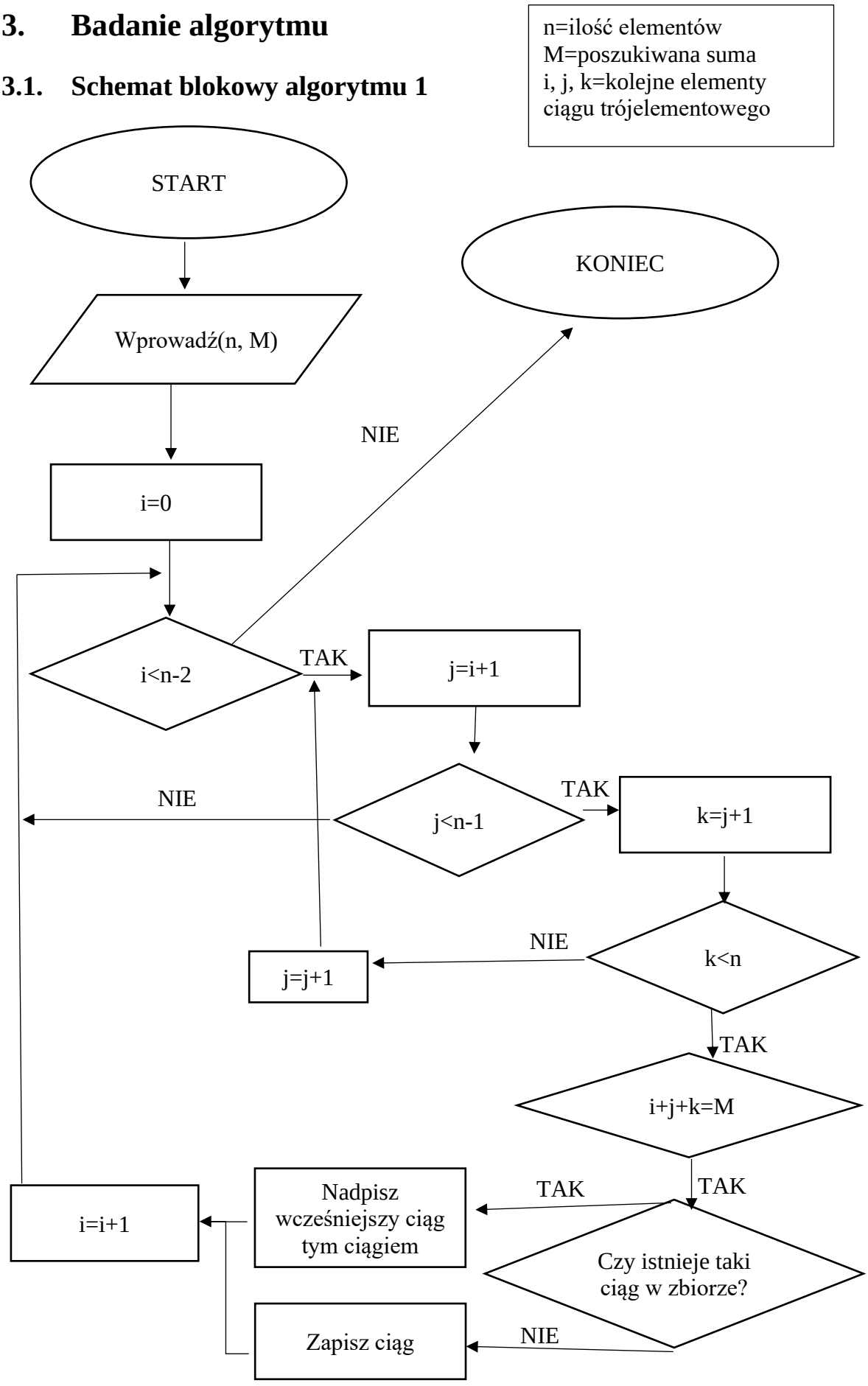
Niestety tak znalezione ciągi przy niekorzystnym ułożeniu danych wejściowych mają tendencję do duplikowania się, co nie jest pożądane. Druga część algorytmu, zagnieżdżona w pierwszej, przeciwdziała takiej sytuacji. Utworzone zostały klasy, aby powtarzające się ciągi trójelementowe móc zapisać jako obiekty. Ciąg znaleziony przez część pierwszą algorytmu zostaje przekonwertowany na typ string , a następnie zapisany jako obiekt. Tak utworzony obiekt zostaje zapisany do zbioru obiektów i wektora obiektów, po czym następuje kolejne przejście pętli. Kolejny znaleziony ciąg poddawany jest sprawdzeniu czy istnieje ciąg identyczny do niego w zbiorze. Jeżeli w zbiorze zostanie znaleziony duplikat to jest on nadpisywany poprzez taki sam nowy ciąg. Jeżeli znaleziony ciąg się nie powtarza jest on dopisywany do wektora.

Drugi algorytm został zbudowany z 2 pętli. Najpierw tablica musi być posortowana. Użyta została do tego funkcja sort (nie skupiałam się na konkretnym sortowaniu, więc użyłam gotowej funkcji). Następnie zastosowana została pętla „for", która wykona się n-2 razy wyznaczając pierwszy element ciągu trójelementowego. Dwa kolejne elementy będą kolejno drugim (lub kolejnym) i ostatnim (lub wcześniejszym) wyrazem. Jeśli taka kombinacja da sumę zażądaną przez użytkownika zostanie zapisana w taki sam sposób jak w części drugiej pierwszego algorytmu. Jeśli natomiast taka kombinacja nie spełni wymagań to, jeżeli dała sumę mniejszą niż oczekiwano, inkrementacji zostaje poddany drugi (lub kolejny) element, gdyż był mniejszy, a jeżeli dała sumę większą, dekrementacji podlega ostatni (lub wcześniejszy) element, gdyż był największy.

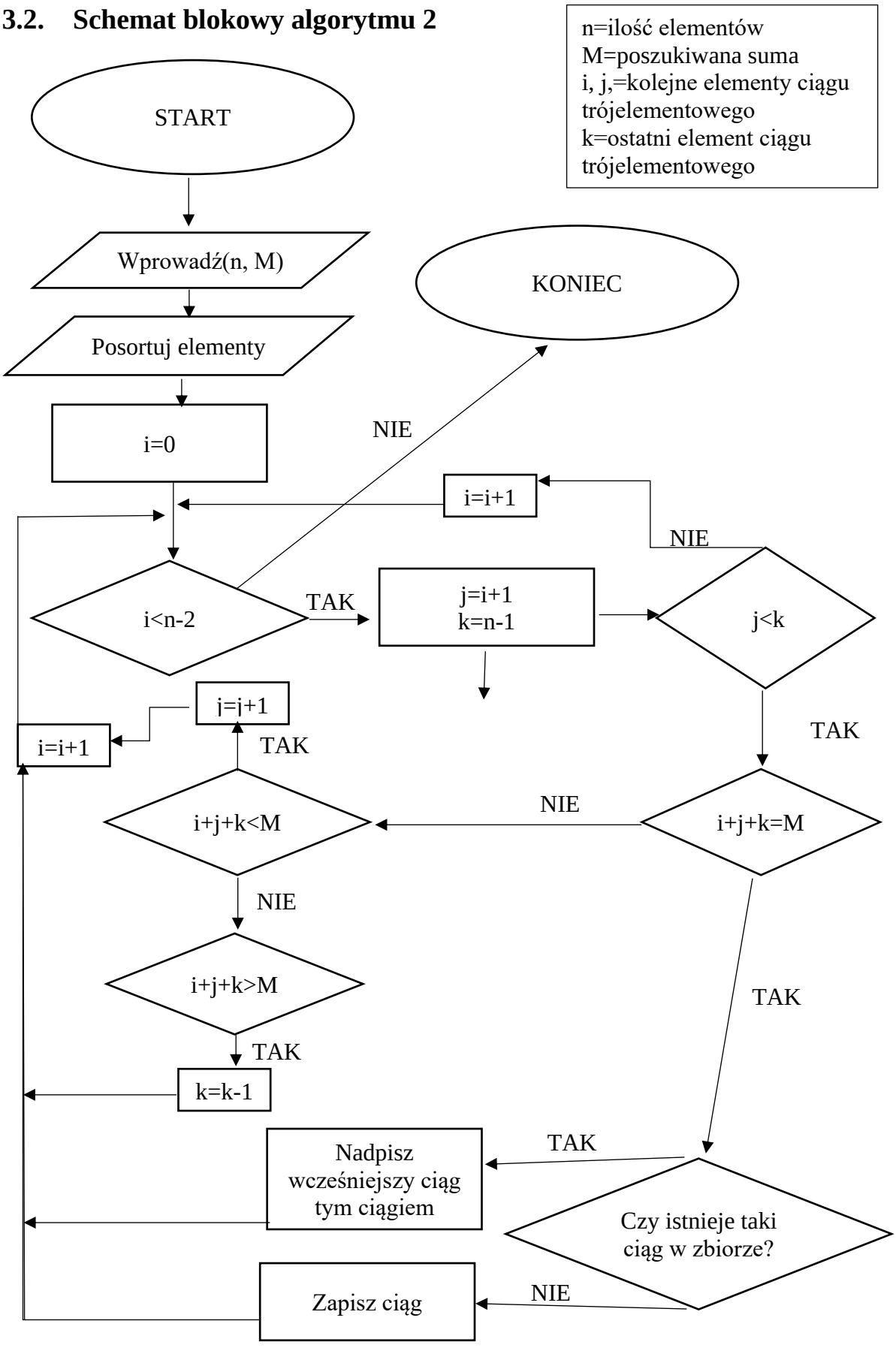
Liczba kombinacji obliczana jest poprzez sprawdzenie ilości elementów w wektorze, gdyż obie wartości są tożsame.

3. Badanie algorytmu

3.1. Schemat blokowy algorytmu 1



3.2. Schemat blokowy algorytmu 2



3.3. Pseudokod

Algorytm 1

Weź pierwszy (lub kolejny) element ciągu.

Weź drugi (lub kolejny) element ciągu

Weź trzeci (lub kolejny) element ciągu

Jeśli suma elementów jest równa M, sprawdź czy jest to duplikat

Jeśli jest to duplikat nadpisz poprzedni ciąg

Jeśli nie jest to duplikat, zapisz ciąg

Jeśli suma elementów nie jest równa M, wróć do ostatniej (lub wcześniejszej) pętli, inkremetując jej iterator

Wypisz znalezione kombinacje oraz określ ich liczbę

Algorytm 2

Weź pierwszy (lub kolejny) element ciągu.

Weź drugi (lub kolejny) element ciągu

Weź ostatni (lub wcześniejszy) element ciągu

Dopóki drugi (lub kolejny) element jest mniejszy od ostaniego

Jeśli suma elementów jest równa M, sprawdź czy jest to duplikat i wróć do kroku pierwszego

Jeśli jest to duplikat nadpisz poprzedni ciąg

Jeśli nie jest to duplikat, zapisz ciąg

Jeśli suma elementów jest mniejsza od M, inkrementuj drugi (lub kolejny) element i wróć do kroku pierwszego

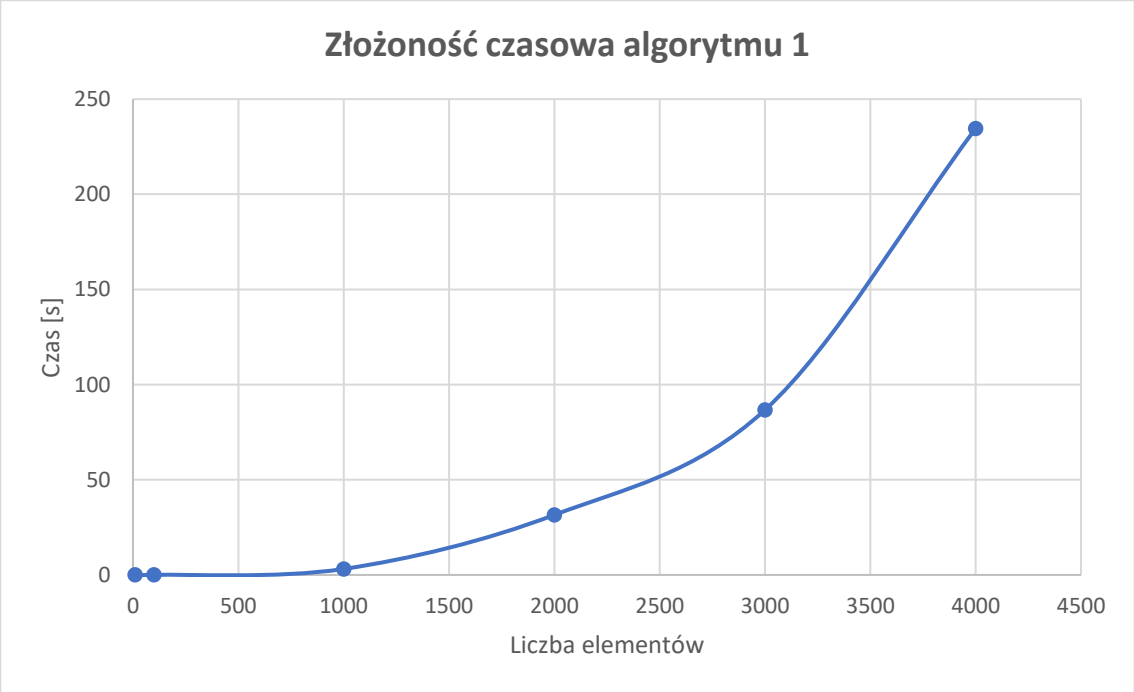
Jeśli suma elementów jest większa od M, dekrementuj ostatni (lub wcześniejszy) element i wróć do kroku pierwszego

Wypisz znalezione kombinacje oraz określ ich liczbę

3.4. Rezultaty testów

Liczba elementów	Czas wykonania zadania przez algorytm[s]
10	0,001
100	0,015
1000	3,133
2000	31,532
3000	86,684
4000	234,612

Tabela 1. Złożoność czasowa algorytmu 1

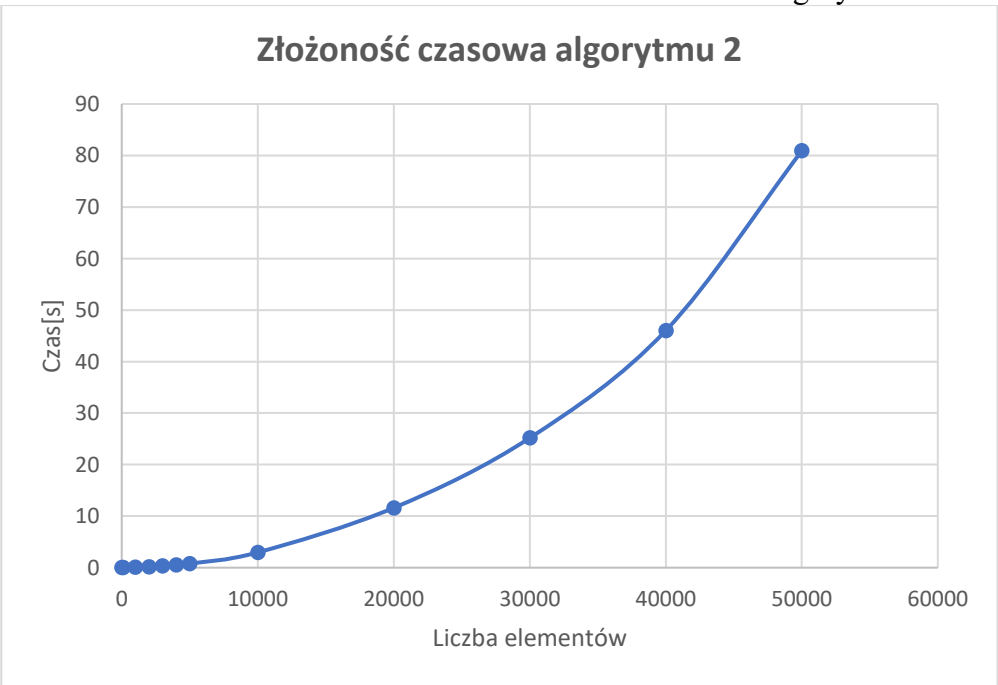


Wykres 1. Złożoność algorytmów

Z testów wynika, że dany algorytm posiada złożoność czasową typu O(n^3).

Liczba elementów	Czas wykonania zadania przez algorytm 2 [s]
10	0,002
100	0,006
1000	0,078
2000	0,121
3000	0,298
4000	0,525
5000	0,792
10000	2,96
20000	11,606
30000	25,181
40000	46,021
50000	80,895

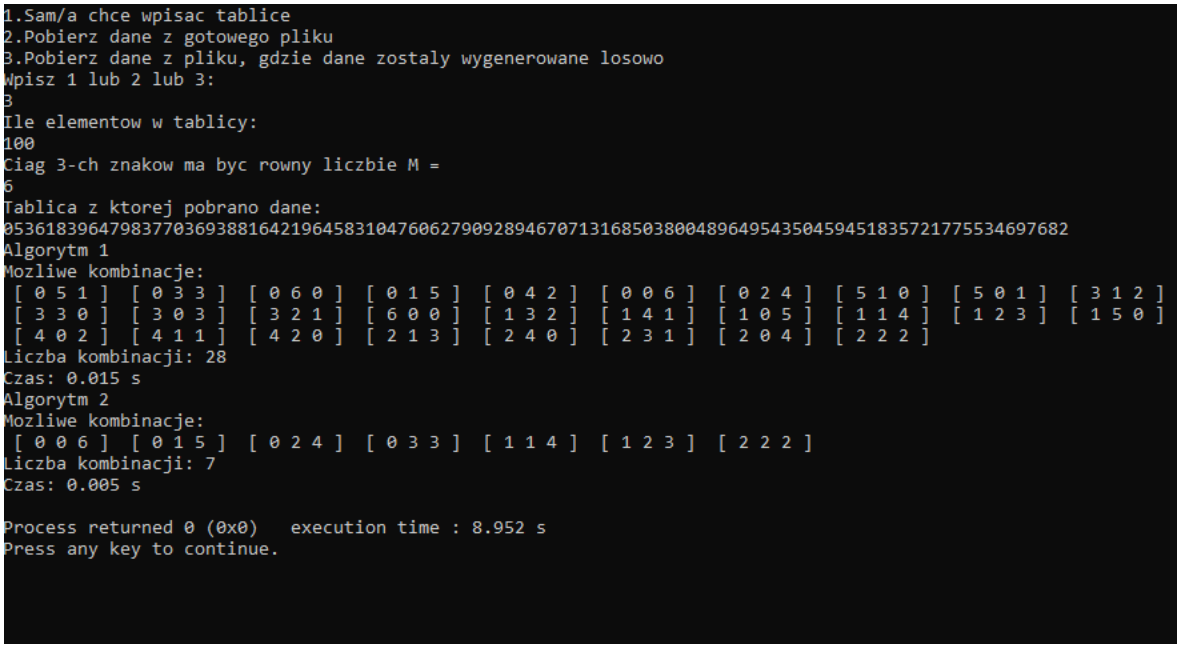
Tabela 2. Złożoność czasowa algorytmu 2



Wykres 2. Złożoność czasowa algorytmu 2

Z testów wynika, że dany algorytm posiada złożoność czasową typu O(n^2).

Oba algorytmy zwracają dobre wyniki, lecz ich interpretacja może być różna. Algorytm 1 zwraca przy dużej liczbie elementów możliwe kombinacje ułożenia liczb w kombinacjach ciągów trójelementowych, natomiast algorytm 2 zwraca jedynie poszczególne wersje danych kombinacji, przez co ich ilość jest mniejsza.



Rysunek 1.Różnice między algorytmami

4. Wnioski

Algorytm 1 działa poprawnie, ale jego działanie na pewno mogłoby zostać poprawione. Sam w sobie posiada jednak dużo ciekawie zastosowanych narzędzi , które mogą być pomocne, nie tylko w tym zagadnieniu, ale także w wielu innych. Dla małej ilości danych algorytm spisuje się sprawnie , lecz dla ich większej ilości należałoby go udoskonalić lub zastosować inny.

Algorytm 2 jest o wiele szybszy, jednakże wyniki przez niego podawane są mniej dokładne i nie są tak szczegółowe jak w algorytmie 1. Przy dużej ilości danych algorytm 2 będzie dużo bardziej efektywny i z pewnością chętniej stosowany.

Oba algorytmy mają swoje dobre i złe strony i oba mogą okazać się przydatne różnych okolicznościach i do różnych zadań.

Załącznik

```
#include <iostream>
#include <fstream>
#include <set>
#include <vector>
#include <iterator>
#include <string>
#include <time.h>
#include <algorithm>
#include <functional>

using namespace std;
void dostuffA();
void dostuffB();
void dostuffC();

class Triplet                    //klasa, aby moc stworzyc ciag trojelemenowy jako objekt
{
public:
   int f, s, t;
};
int find_numbersA(int tab[], int n, int m)
{
   cout <<"Algorytm 1"<<endl;
   clock_t start, stop;
   double czas;

   fstream file_out;
   file_out.open("triplets_exit.txt", ios::out);

   vector<Triplet> triplets;          //w tym wektorze beda magazynowane ciagi
trojelementowe(z niego bede potem wypisywane)
   set<string> saveTriplets;        //zbior, gdzie beda przechowywane wartości ciagow, by
uniknac ich powtorzenia
   string tas  ;                    //tas=triplet as a string, zmienna string sluzaca
przekonwertowaniu ciagu liczb na tekst
   Triplet outputTriplet;              //objekt, gdzie zapisany bedzie ciag trojelemenowy z
wektora, jesli dany ciag sie nie powtarza

   start = clock();
   //pętla szukająca elementow ciagu
   for (int i=0; i<n-2; i++)              //wez pierwszy element
   {
      for (int j=i+1; j<n-1; j++)           //wez drugi element
      {
         for(int k=j+1; k<n; k++)            //wez trzeci element
         {
            if(tab[i]+tab[j]+tab[k]==m)     //zobacz czy dany ciąg jest równy sumie
            {
```

```
                tas=to_string(tab[i])+" : "+to_string(tab[j])+" : "+to_string(tab[k]);
//konwertowanie ciągu trojelementowego z typu int na string
                if(saveTriplets.find(tas)==saveTriplets.end())                    //sprawdzanie
czy dany ciąg znajduje się w zbiorze saveTriplets
                {
                   saveTriplets.insert(tas);                                      //dodanie elementow z
wektora tas do zbioru saveTriplets
                   outputTriplet.f = tab[i];                                      //przypisanie strukturze
outputTriplet wartości
                   outputTriplet.s = tab[j];
                   outputTriplet.t = tab[k];
                   triplets.push_back(outputTriplet);                             //dodanie
elemwntow z outputTriplet do wektora triplets
                }

             }
          }
       }
    }
    if (triplets.size() == 0)                //liczba elementów w wektorze triplets jest równa
liczbie kombinacji,
       //jeżeli elementów w wektorze jest 0 to nie wypisujemy kombinacji
    {
       cout<<"Liczba kombinacji: 0"<<endl;
       return 0;
    }
    cout <<"Mozliwe kombinacje: "<< endl;
    for(int i = 0; i< triplets.size(); i++)         //petla wypisujaca poszczgolne ciagi i liczbe
kombinacji
    {
       cout <<" [ " << triplets[i].f <<" "<< triplets[i].s <<" "<< triplets[i].t<<" ] ";
    }
    cout<<endl<<"Liczba kombinacji: "<<triplets.size();    //liczba elementów w wektorze
triplets jest równa liczbie kombinacji
    for(int i = 0; i< triplets.size(); i++)         //petla wypisujaca poszczgolne ciagi i liczbe
kombinacji do pliku
    {
       file_out << triplets[i].f;
       file_out << triplets[i].s;
       file_out << triplets[i].t<<endl;
    }

    file_out.close();

    stop = clock();
    czas = (double)(stop - start) / CLOCKS_PER_SEC;
    cout<<endl<<"Czas: "<<czas<<" s"<<endl;
    ;
}

int find_numbersB(int tab[], int n, int m)
{
```

```
   cout <<"Algorytm 2"<<endl;
   clock_t start, stop;
   double czas;
   sort (tab, tab+n);                    // KONIECZNIE posortuj tablice, inaczej zacznie
gubic niektore skrajne kombinacje
   fstream file_out;
   file_out.open("triplets_exit.txt", ios::out);

   vector<Triplet> triplets;           //w tym wektorze beda magazynowane ciagi
trojelementowe(z niego bede potem wypisywane)
   set<string> saveTriplets;         //zbior, gdzie beda przechowywane wartości ciagow, by
uniknac ich powtorzenia
   string tas  ;                      //tas=triplet as a string, zmienna string sluzaca
przekonwertowaniu ciagu liczb na tekst
   Triplet outputTriplet;              //objekt, gdzie zapisany bedzie ciag trojelemenowy z
wektora, jesli dany ciag sie nie powtarza

   start = clock();
   //pętla szukająca elementow ciagu
   for (int i=0; i<n-2; i++)              //wez pierwszy element
   {
      int j=i+1;                        // wez drugi element
      int k=n-1;                         // weź ostatni element
      while (j<k)                        // jeśli j<k(tablica posortowana, więc jest to prawda):
      {
         if(tab[i]+tab[j]+tab[k]==m)    //jak i+j+k=m to zapisz ciąg
         {

            tas=to_string(tab[i])+" : "+to_string(tab[j])+" : "+to_string(tab[k]);
//konwertowanie ciągu trojelementowego z int na string
            if(saveTriplets.find(tas)==saveTriplets.end())                    //sprawdzanie czy
dany ciąg znajduje się w zbiorze saveTriplets
            {
               saveTriplets.insert(tas);                                  //dodanie elemwntow z
wektora tas do zbioru saveTriplets
               outputTriplet.f = tab[i];                                  //przypisanie strukturze
outputTriplet wartości
               outputTriplet.s = tab[j];
               outputTriplet.t = tab[k];
               triplets.push_back(outputTriplet);                          //dodanie elemwntow
z outputTriplet do wektora triplets
            }
            j++;
            k--;

         }
         else if(tab[i]+tab[j]+tab[k]<m)     // jeśli i+j+k<m powiększ j, bo j mniejsze
         {
            j++;
         }
         else if(tab[i]+tab[j]+tab[k]>m)     // jeśli i+j+k<m pomniejsz k, bo k wieksze
         {
```

```
            k--;
          }
        }

    }

    if (triplets.size() == 0)                   //liczba elementów w wektorze triplets jest równa
liczbie kombinacji, jeżeli elementów w wektorze jest 0 to nie wypisujemy kombinacji
    {
        cout<<"Liczba kombinacji: 0"<<endl;
        return 0;
    }
    cout <<"Mozliwe kombinacje: "<< endl;
    for(int i = 0; i< triplets.size(); i++)         //petla wypisujaca poszczgolne ciagi i liczbe
kombinacji
    {
        cout <<" [ " << triplets[i].f <<" "<< triplets[i].s <<" "<< triplets[i].t<<" ] ";
    }
    cout<<endl<<"Liczba kombinacji: "<<triplets.size() ;    //liczba elementów w wektorze
triplets jest równa liczbie kombinacji
    for(int i = 0; i< triplets.size(); i++)         //petla wypisujaca poszczgolne ciagi i liczbe
kombinacji
    {
        file_out << triplets[i].f;
        file_out << triplets[i].s;
        file_out << triplets[i].t<<endl;
    }

    file_out.close();

    stop = clock();
    czas = (double)(stop - start) / CLOCKS_PER_SEC;
    cout<<endl<<"Czas: "<<czas<<" s"<<endl;

}

int main()
{
    char choice;
    cout<<"1.Sam/a chce wpisac tablice"<<endl;
    cout<<"2.Pobierz dane z gotowego pliku"<<endl;
    cout<<"3.Pobierz dane z pliku, gdzie dane zostaly wygenerowane losowo"<<endl;
    cout<<"Wpisz 1 lub 2 lub 3: "<<endl;
    cin >> choice;
    switch(choice)
    {
    case '1':
        dostuffA();
        break;
    case '2':
        dostuffB();
        break;
```

```
    case '3':
        dostuffC();
        break;
    default:
        cout<<"Nie ma takiej opcji w menu!";
    }

    return 0;
}

void dostuffA()
{
    int n, M;
    cout << "Ile elementow w tablicy:"<< endl;
    cin >> n;
    int arr[n];
    cout << "Wpisz liczby do tablicy: "<<endl;
    for (int i=0; i<n; i++)
    {
        cin>> arr[i];
    }
    cout << "Ciag 3-ch znakow ma byc rowny liczbie M = "<<endl;
    cin >> M;
    find_numbersA(arr, n, M);
    find_numbersB(arr, n, M);
}

void dostuffB()
{
    fstream file;
    int n=0, M, nr_lines=0 ;
    int i=0;
    int *arr;
    arr=new int [nr_lines];

    string line;
    file.open("random_from_0_to_9.txt", ios::in);
    if(file.good()==false)
    {
        cout<<"Nie mozna otworzyc pliku!";
    }
    else
    {
        while( !file.eof() )
        {
            getline(file, line);
            arr[nr_lines]=atof(line.c_str());
            nr_lines++;
        }

        cout << "Ciag 3-ch znakow ma byc rowny liczbie M = "<<endl;
        cin >> M;
```

```
        for (int i=0; i<nr_lines; i++)
        {
           file>>arr[i];
        }
        cout <<"Tablica z ktorej pobrano dane: "<< endl;
        for (int i=0; i<nr_lines; i++)
        {
           cout << arr[i];
        }
        cout << endl;
        find_numbersA(arr, nr_lines, M);
        find_numbersB(arr, nr_lines, M);
    }
    file.close();
}

void dostuffC()
{
    srand ( ( unsigned )time ( NULL ) );
    fstream file_random;
    int n, M;
    cout << "Ile elementow w tablicy:"<< endl;
    cin >> n;
    cout << "Ciag 3-ch znakow ma byc rowny liczbie M = "<<endl;
    cin >> M;
    file_random.open("new_random.txt", ios::out);
    file_random.open("new_random.txt", ios::in);
    int *arr = new int[n];
    for(int i=0; i<n; i++)  //zapis liczb losowych do tablicy
    {
       arr[i] = rand() % 10;
    }
    for(int i=0; i<n; i++) // zapis tablicy do pliku
    {
       file_random << arr[i];
    }

    file_random.open("new_random.txt", ios::in);

    for(int i=0; i<n; i++)
    {
       file_random >> arr[i];
    }
    cout <<"Tablica z ktorej pobrano dane: "<< endl;
    for(int i=0; i<n; i++)
    {
       cout << arr[i];
    }
    cout << endl;
    find_numbersA(arr, n, M);
    find_numbersB(arr, n, M);
    file_random.close();}
```