WYDZIAŁ
MATEMATYKI
I FIZYKI STOSOWANEJ
POLITECHNIKI RZESZOWSKIEJ

Barbara Majkut

Wyszukiwanie najdłuższego malejącego podciągu

(dr inż.) Mariusz Borkowski (prof. PRz)

Rzeszów, 2025

Spis treści

1. Treść zadania

Dla ciągu (w postaci tablicy) zawierającego wartości całkowite, znajdź malejący podciąg o największej długości.

Przykład

Wejście: A[]=[-10,5,8,1,-4,-4,10,3,-1,1]

Wyjście: Najdłuższe malejące podciągi to 8,1,-4 oraz 10,3,-1

2. Etapy rozwiązywania problemu.

W trakcie rozwiązywania zagadnienia należy zachować **właściwą** kolejność kolejnych podzadań, która przedstawia się następująco:

1) Analiza problemu.

2) Utworzenie schematu blokowego.

3) Utworzenie schematu napisanego w pseudokodzie.

4) Analiza działania zapisanego algorytmu na kartce papieru (Rozwiązanie zadania przykładowego- ołókowe rozwiązanie).

5) Teoretyczne oszacowanie złożoności obliczeniowej.

6) Ponowne przemyślenie problemu i próba wymyślenia algorytmu wydajniejszego.

7) Powtórzenie punktów 2-5 dla algorytmu wydajnieszego.

8) Powtórzenie punktów 6-7 wybraną przez siebie liczbę razy.

9) Implementacja wymyślonych algortymów w wybranym środowisku i języku.

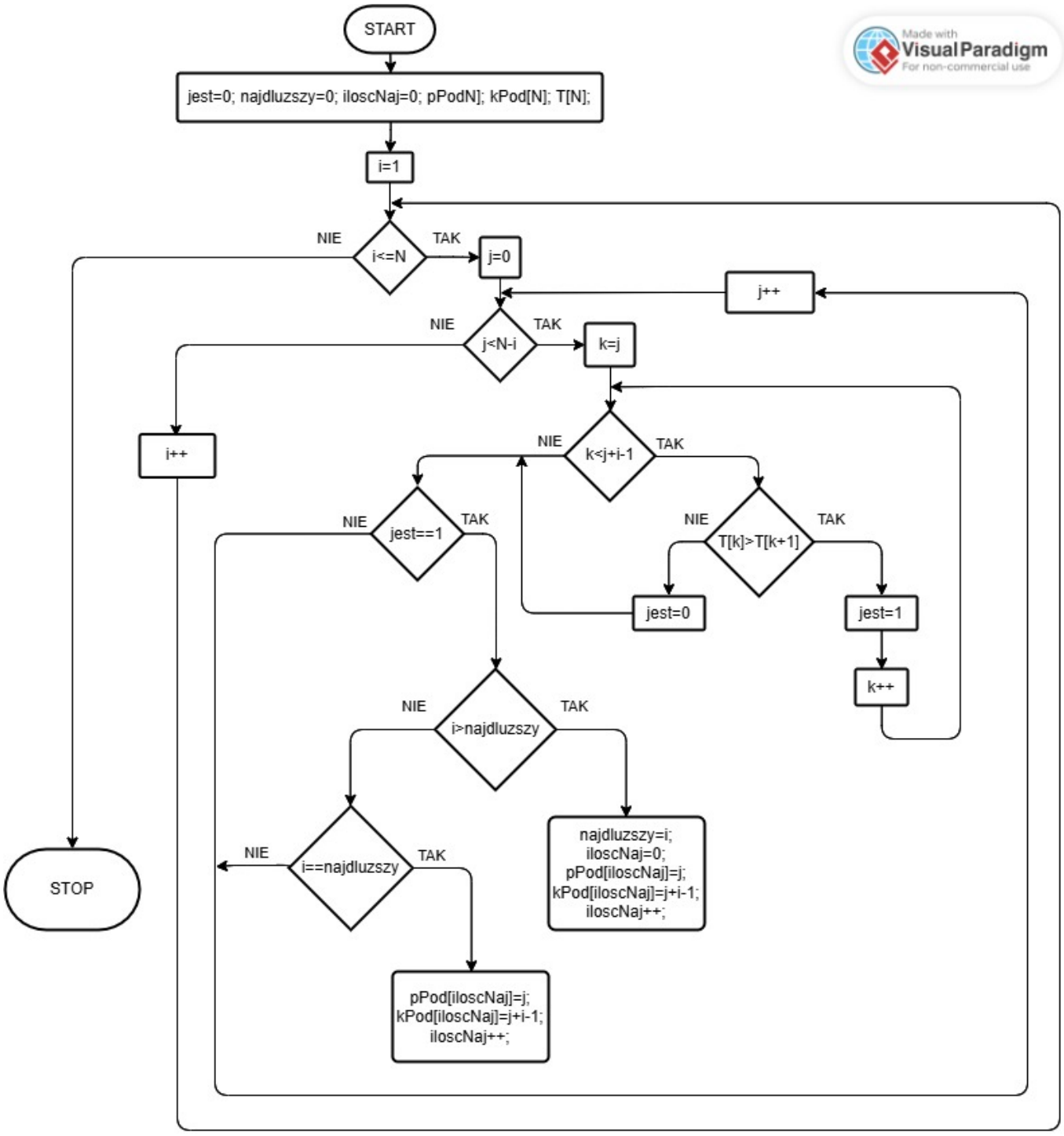
10) Testy numeryczne działania algorytmów i potwierdzenie doświadczalne wyników teoretycznych.

2.1. Rozwiązanie - podejście pierwsze(brute force)

2.1.1. Analiza problemu

Problem który został przedstawiony polega na znalezieniu najdłuższego malejącego podciągu. Rozwiązanie za pomocą metody brute force pozwala na stworzenie każdego możliwego podciągu znajdującego się w tablicy A[i]. Jeżeli w utworzonym podciągu występuje warunek malejący, indeksy owego podciągu muszą być zapisane do osobnych tablic, przechowujących indeksy początkowe(poczatekPodciagow[N]), oraz końcowe(koniecPodciagow[N]). Następnie należy porównać długości tych podciągów i wyznaczyć najdłuższy spośród nich. W momencie wystąpienia kilku podciągów o takiej samej (najdłuższej) długości, ich indeksy powinny być zapisane pod kolejnym indeksem tablic przechowujących początki i końce danych podciągów.

2.1.2. Schemat blokowy algorytmu



Rysunek 2.1: Schemat blokowy algorytmu pierwszego

2.1.3. Algorytm zapisany w pseudokodzie

```
    input:
N   //liczba element w w tablicy
A[0....N-1] //tablica wejsciowa
jest        //zmienna do warunku sprawdzajacego czy dany
   podciag jest malejacy
najdluzszy
iloscNajdluzszych
    output:
poczatekPodciagow[iloscNajdluzszych]
koniecPodciagow[iloscNajdluzszych]    //tablice wynikowe
  // w ktorych przechowywane s  granice indeksow podciagow
   malejacych z tablicy A[N]
DLA i=1 DO N wykonaj
  DLA j=0 DO N-i-1 wykonaj
        DLA k=j DO j+i-2 wykonaj
          JEZELI A[k] > A[k+1]
                USTAW Jest na 1
          W PRZECIWNYM RAZIE
                USTAW jest na 0
          endJEZELI
        endDLA  //koniec petli wewnetrznej
      JEZELI jest==1 then
          JEZELI i>najdluzszy
          USTAW :
                najdluzszy na i
                iloscnajdluzszych na 0
                poczatekPodciagow[iloscNajdluzszych] na j
                koniecPodciagow[iloscNajdluzszych] na j+i-1
                iloscNajdluzszych zwieksz o 1
          endJEZELI
          W PRZECIWNYM RAZIE JEZELI i==najdluzszy
          USTAW :
                poczatekPodciagow[iloscNajdluzszych] na j
                koniecPodciagow[iloscNajdluzszych] na j+i-1
                iloscNajdluzszych zwieksz o 1
          endJEZELI
      endJEZELI
   endDLA  //koniec petli srodkowej
endDLA     //koniec petli zewnetrznej

```

Listing 1: Pseudokod BruteForce

2.1.4. Sprawdzenie poprawności algorytmu poprzez "ołówkowe"rozwiązanie problemu

Prezentacja poprawnego wykonania tej części zadania zostanie przedstawiona na przykładzie danych wejściowych. Pomijamy dlługość równą 1 (i=1).

A[]=[-10,5,8,1,-4,-4,10,3,-1,1]

długość - i	indeks początkowy - j	granica indeksów - k	sprawdzane liczby	zmienna "jest"	zmienna "najdłuższy"	zmienna "ilość"	pPodciągów[ilość]	kPodciągów[ilość]
2	0	0-1	-10>5	0	-	-	-	-
2	1	1-2	5>8	0	-	-	-	-
2	2	2-3	8>1	1	2	0	pP[0]=2	kP[0]=3
2	3	3-4	1>-4	1	2	1	pP[1]=3	kP[1]=4
2	4	4-5	-4>-4	0	-	-	-	-
2	5	5-6	-4>10	0	-	-	-	-
2	6	6-7	10>3	1	2	2	pP[2]=6	kP[2]=7
2	7	7-8	3>-1	1	2	3	pP[3]=7	kP[3]=8
2	8	8-9	-1>1	0	-	-	-	-
3	0	0-1-2	-10>5>8	0	-	-	-	-
3	1	1-2-3	5>8>1	0	-	-	-	-
3	2	2-3-4	8>1>-4	1	3	0	pP[0]=2	kP[0]=4
3	3	3-4-5	1>-4>-4	0	-	-	-	-
3	4	4-5-6	-4>-4>10	0	-	-	-	-
3	5	5-6-7	-4>10>3	0	-	-	-	-
3	6	6-7-8	10>3>-1	1	3	1	pP[1]=6	kP[1]=8
3	7	7-8-9	3>-1>1	0	-	-	-	-
4	0	0-1-2-3	-10>5>8>1	0	-	-	-	-
4	1	1-2-3-4	5>8>1>-4	0	-	-	-	-
4	2	2-3-4-5	8>1>-4>-4	0	-	-	-	-
4	3	3-4-5-6	1>-4>-4>10	0	-	-	-	-
4	4	4-5-6-7	-4>-4>10>3	0	-	-	-	-
4	5	5-6-7-8	-4>10>3>-1	0	-	-	-	-
4	6	6-7-8-9	10>3>-1>1	0	-	-	-	-
itd...	*itd...*	*itd...*	*itd...*	*itd...*	*itd...*	*itd...*	*itd...*	*itd...*
9	0	0-1-2-3-4-5-6-7-8-9	10>5>8>1>-4>-4>10>3>-1>1	0	-	-	-	-

Rysunek 2.2: Ołówkowe rozwiązanie algorytm pierwszy

2.1.5. Teoretyczne oszacowanie złożoności obliczeniowej

Złożoność obliczeniowa programu można określić, analizując jego strukturę:

1) **Pętla zewnętrzna (długość podciągu):**

Pętla for, iteruje od długości 1 do rozmiar N. Oznacza to, że wykonuje O(n) iteracji, gdzie N to rozmiar tablicy.

2) **Pętla środkowa (początkowy indeks podciągu):**

Dla każdego i, pętla wewnętrzna iteruje od j=0 do j<N-i po wszystkich możliwych początkowych indeksach podciągu o danej długości i. W najgorszym przypadku liczba iteracji tej pętli wynosi O(n) dla każdej długości podciągu.

3) **Sprawdzanie warunku malejącości (pętla wewnętrzna:**

Wewnątrz drugiej pętli znajduje się pętla for, która sprawdza, czy dany podciąg jest malejący. Ta pętla iteruje przez każdy element danego podciągu, czyli wykonuje O(i) iteracji.

4)**Złożoność całkowita:**

Dla każdego i (od 1 do n) sprawdzamy wszystkie możliwe j, a dla każdego takiego początkowego indeksu przechodzimy przez cały podciąg. Całkowita liczba operacji to suma O(i)×O(n) dla wszystkich możliwych długości i.

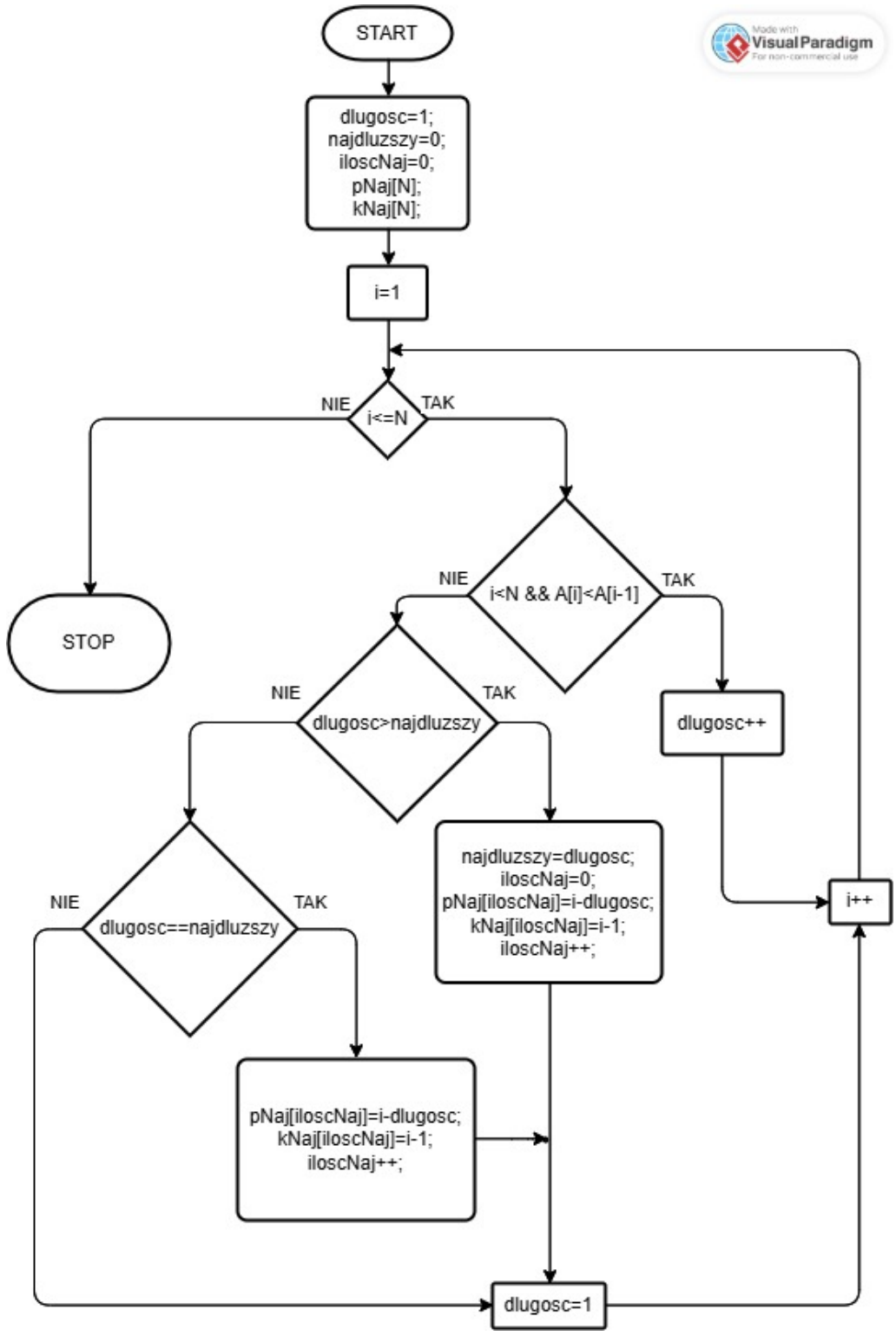
Złożoność:

$$O(n) \times O(n) \times O(n) = O(n^3)$$

2.2. Rozwiązanie - próba druga

Problem polega na znalezieniu najdłuższego malejącego podciągu za pomocą prostszej implementacji i uzyskaniu optymalnie mniejszej złożoności obliczeniowej oraz czasowej. Na wejściu mamy tablicę liczb całkowitych o długości N.
Do rozwiązania problemu należy użyć instrukcji warunkowej if, w której nastąpi porównanie elementów z tablicy A[]. Jeżeli warunek zostanie spełniony, zmienna określająca długość zostanie zwiększona o 1. Podczas kolejnych iteracji pętli for, w momencie gdy dany warunek nie zostanie spełniony, dotychczasowa długość oraz jej indeks początkowy oraz końcowy, zostanie zapisany w tablicach dynamicznych.

2.2.1. Schemat blokowy algorytmu wydajniejszego



Rysunek 2.3: Schemat blokowy algorytmu drugiego

2.2.2. Algorytm wydajniejszy zapisany w pseudokodzie

```
    input:
N   //liczba element w w tablicy
A[0....N-1] //tablica wejsciowa
dlugosc ustaw na 1            //zmienna do zliczania dlugosci
   podciagow
najdluzszy
iloscNajdluzszych
    output:
pNajdluzszego[iloscNajdluzszych]
kNajdluzszego[iloscNajdluzszych]    //tablice wynikowe

DLA i=1 DO i = N wykonaj
    JEZELI i<N oraz A[i]<A[i-1]
        zwieksz dlugosc o 1
    W PRZECIWNYM RAZIE
      JEZELI dlugosc jest wieksza od najdluzszy
            USTAW:
            najdluzszy na dlugosc
            iloscNajdluzszych na 0
            pNajdluzszego[iloscNajdluzszych] na i-dlugosc
            kNajdluzszego[iloscNajdluzszych] na i-1
            iloscNajdluzszych zwieksz o 1
      W PRZECIWNYM RAZIE JEZELI dlugosc jest rowna najdluzszy
            USTAW:
            pNajdluzszego[iloscNajdluzszych] na i-dlugosc
            kNajdluzszego[iloscNajdluzszych] na i-1
            iloscNajdluzszych zwieksz o 1
      endJEZELI
      ustaw dlugosc na 1
    endJEZELI
endDLA
```

Listing 2: Pseudokod

2.2.3. "Ołówkowe"sprawdzenie poprawności algorytmu nr 2

Prezentacja poprawanego wykonania tej części zadania zostanie przedstawiona na przykladzie danych wejściowych

$$A[]=[-10,5,8,1,-4,-4,10,3,-1,1]$$

i	i<N && A[i-1]>A[i]	zmienna "długość"	zmienna "najdłuższy"	zmienna "ilość"	pPodciagow[ilość]	kPodciagow[ilość]	zmienna "dlugosc" po warunku
1	-10>5	1	1	0	pP[0]=0	kP[0]=0	1
2	5>8	1	1	1	pP[1]=1	kP[1]=1	1
3	8>1	2	-	-	-	-	-
4	1>-4	3	-	-	-	-	-
5	-4>-4	3	3	0	pP[0]=2	kP[0]=4	1
6	-4>10	1	-	-	-	-	-
7	10>3	2	-	-	-	-	-
8	3>-1	3	-	-	-	-	-
9	-1>1	3	3	1	pP[1]=6	kP[1]=8	1
10	-	1	-	-	-	-	-

Rysunek 2.4: Ołówkowe rozwiązanie - algorytm drugi

2.2.4. Teoretyczne oszacowanie złożoności obliczeniowej dla algorytmu 2

Kod działa w oparciu o jednokrotną iterację tablicy wejściowej A w celu znalezienia najdłuższego malejącego podciągu oraz zapisania indeksów jego początku i końca.

1) **Pętla for w funkcji:**

Wspomniana pętla przechodzi przez wszystkie elementy tablicy A, wykonując dokładnie N iteracji. Każda iteracja wykonuje stałą liczbę operacji (porównania i przypisania), co oznacza złożoność O(n).

2) **Złożoność czasowa**

Wszystkie operacje (iteracja, alokacja pamięci) mają złożoność O(N). W rezultacie złożoność czasowa całego algorytmu to O(n)

3) **Złożoność całkowita**

Wykorzystanie tylko jednej pętli sprowadza się do iteracji tylko N razy w całym algorytmie, dlatego ostateczna złożonośc wynosi:

$$O(n)$$

2.3. Implementacja wymyślonych algorytmów w wybranym środowisku i języku oraz eksperymentalne potwierdzenie wydajności(złożoności obliczeniowej) algorytmów

2.3.1. Implementacja pierwszego algorytmu (Brute Force)

```
void ZnajdzNajwiekszyPodciagMalejacy(int A[], int N)  //BRUTE FORCE
{
    int jest=0;
    int najdluzszy=0;
    int iloscNajdluzszych=0;
    int poczatekPodciagow[N];
    int koniecPodciagow[N];

    for(int i=1; i<=N; i++)  //ilosc wyrazow -> dlugosc podciagu
    {
        for(int j=0; j<N-i; j++)  //indeks od ktorego zaczyna sie podciag
        {
            for(int k=j; k<j+i-1; k++)   //granica indeksow w ktorych jest sprawdzany aktualny podciag
            {
                if(A[k] > A[k+1])   //warunek porownania liczb w danym podciagu
                {
                    jest=1;
                }
                else {jest=0; break;}
            }

            if(jest==1)
            {
                if(i>najdluzszy)
                {
                    najdluzszy=i;
                    iloscNajdluzszych=0;
                    poczatekPodciagow[iloscNajdluzszych]=j;   //zapisywanie indeksu poczatkowego podciagu
                    koniecPodciagow[iloscNajdluzszych]=j+i-1; //zapisywanie indeksu koncowego podciagu
                    iloscNajdluzszych++;
                }

                else if(i==najdluzszy)  //wykonuje sie w momencie wystapienia wiecej niz jednego podciagu tej samej dlugosci
                {
                    poczatekPodciagow[iloscNajdluzszych]=j;
                    koniecPodciagow[iloscNajdluzszych]=j+i-1;
                    iloscNajdluzszych++;
                }
            }
        }
    }
```

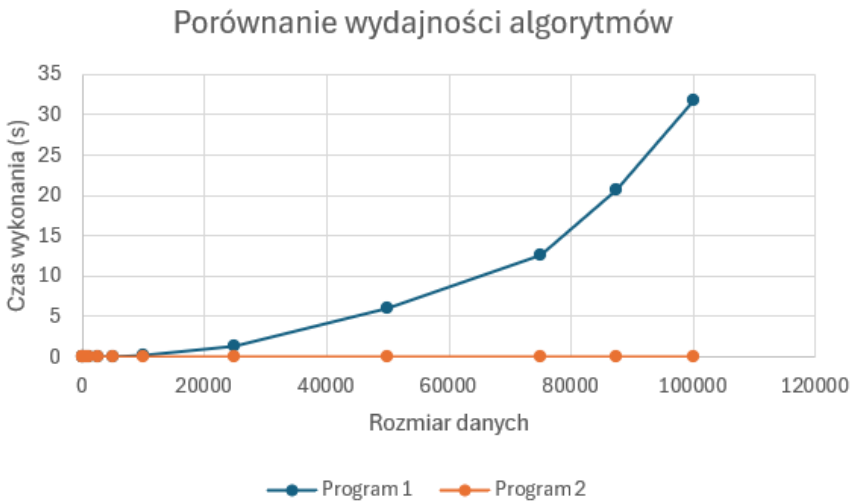
```
TEST NR: 1
Rozmiar tablicy wynosi: 15
Wejscie:
{ -8 -98 -69 79 -44 -51 -28 -54 51 10 90 6 -32 56 -79 }
Najdluzszy podciag zawiera 3 wyrazow:
Wyjscie:
{ 79 -44 -51 }
{ 90 6 -32 }
Ilosc znalezionych podciagow: 2


TEST NR: 2
Rozmiar tablicy wynosi: 20
Wejscie:
{ 19 98 -70 72 30 39 44 7 -25 49 66 5 32 -16 -2 -10 -37 -19 -50 -23 }
Najdluzszy podciag zawiera 3 wyrazow:
Wyjscie:
{ 44 7 -25 }
{ -2 -10 -37 }
Ilosc znalezionych podciagow: 2


TEST NR: 3
Rozmiar tablicy wynosi: 17
Wejscie:
{ -13 7 23 -74 80 72 82 22 61 -24 84 75 10 -87 85 89 -74 }
Najdluzszy podciag zawiera 4 wyrazow:
Wyjscie:
{ 84 75 10 -87 }
Ilosc znalezionych podciagow: 1
```

2.3.2. Wydajniejsza implementacja

```
void ZnajdzNajdluzszyPodciagMalejacy2(int A[], int N)
{
    int dlugosc=1;
    int najdluzszy=0;
    int IloscNajdluzszych=0;

    int *PoczatekNajdluzszego=new int[N];
    int *KoniecNajdluzszego=new int[N];

    //algorytm
    for(int i=1; i<=N; i++)
    {
        if(i<N && A[i]<A[i-1]) //porównanie elementow w tablicy A
        {
            dlugosc++;
        }
        else
        {
            if(dlugosc>najdluzszy)
            {
                najdluzszy=dlugosc;
                IloscNajdluzszych=0;
                PoczatekNajdluzszego[IloscNajdluzszych]=i-dlugosc;
                KoniecNajdluzszego[IloscNajdluzszych]=i-1;
                IloscNajdluzszych++;
            }
            else if(dlugosc==najdluzszy)
            {
                PoczatekNajdluzszego[IloscNajdluzszych]=i-dlugosc;
                KoniecNajdluzszego[IloscNajdluzszych]=i-1;
                IloscNajdluzszych++;
            }
            dlugosc=1;
        }
    }
```

```
TEST NR 1:
Rozmiar tablicy wynosi: 21
Wejscie:
{ 42  84 -71  -8  86  67 -89  37 -93  82  81 -18  41 -73  94  24 -72  16 -26 -31  31   }
Najdluzszy podciag zawiera 3 wyrazow:
Wyjscie:
{ 86 67 -89 }
{ 82 81 -18 }
{ 94 24 -72 }
{ 16 -26 -31 }
Ilosc znalezionych podciagow: 4

TEST NR 2:
Rozmiar tablicy wynosi: 18
Wejscie:
{ 49 -11  -6 -60 -10  65  92  34  38  78  93  72 -14  24 -74  67  66  -8   }
Najdluzszy podciag zawiera 3 wyrazow:
Wyjscie:
{ 93 72 -14 }
{ 67 66 -8 }
Ilosc znalezionych podciagow: 2

TEST NR 3:
Rozmiar tablicy wynosi: 21
Wejscie:
{ -15 -65 -66  55  74  11  11  62  36 -16 -26 -57 -23 -81 -34  57 -36 -46  55   1  49   }
Najdluzszy podciag zawiera 5 wyrazow:
Wyjscie:
{ 62 36 -16 -26 -57 }
Ilosc znalezionych podciagow: 1
```

2.4. Testy wydajności algorytmów



Rozmiar danych	Program 1 (s)	Program 2 (s)
10	0,001	0,001
50	0,001	0,001
100	0,002	0,002
500	0,002	0,002
1000	0,003	0,003
2500	0,015	0,007
5000	0,035	0,006
10000	0,18	0,004
25000	1,379	0,003
50000	5,947	0,004
75000	12,603	0,004
87500	20,681	0,003
100000	31,861	0,004
120000	44,522	0,004

2.5. Appendix

```
#include <iostream>
#include <ctime>
#include <time.h>

using namespace std;

//funkcja szukajaca najdluzszego malejacego podciagu za pomoca metody brute force
void ZnajdzNajwiekszyPodciagMalejacy(int A[], int N)  //BRUTE FORCE
{
    int jest=0;
    int najdluzszy=0;
    int iloscNajdluzszych=0;
    int *poczatekPodciagow=new int [N]; //dynamiczne alokowanie tablicy przechowujacej indeks pierwszej liczby podciagu
    int *koniecPodciagow=new int [N];   //dynamiczne alokowanie tablicy przechowujacej indeks ostatniej liczby podciagu

    for(int i=1; i<=N; i++)  //ilosc wyrazow -> dlugosc podciagu
    {
        for(int j=0; j<N-i; j++)  //indeks od ktorego zaczyna sie aktualnie sprawdzany podciag
        {
            for(int k=j; k<j+i-1; k++)   //granica indeksow w ktorych jest sprawdzany aktualny podciag
            {
                if(A[k] > A[k+1])   //warunek porownania liczb w danym podciagu
                {
                    jest=1;
                }
                else {jest=0; break;}
            }

            if(jest==1)
            {
                if(i>najdluzszy)        //sprawdzanie czy aktualnie sprawdzana dlugosc jest wieksza od dotychczasowej najdluzszej
```

```
32                  {
33                      najdluzszy=i;
34                      iloscNajdluzszych=0;
35                      poczatekPodciagow[iloscNajdluzszych]=j;
    //zapisywanie indeksu poczatkowego podciagu
36                      koniecPodciagow[iloscNajdluzszych]=j+i-1;
    //zapisywanie indeksu koncowego podciagu
37                      iloscNajdluzszych++;     //zwiekszamy ilosc
    najdluzszych podciagow w razie wystapienia wiecej niz jednego
    podciagu takiej samej dlugosci
38                  }
39
40                  else if(i==najdluzszy)  //warunek dla wystapienia
    wiecej niz jednego podciagu tej samej dlugosci
41                  {
42                      poczatekPodciagow[iloscNajdluzszych]=j;
    //zapisywanie indeksu poczatkowego podciagu
43
    koniecPodciagow[iloscNajdluzszych]=j+i-1;//zapisywanie indeksu
    koncowego podciagu
44                      iloscNajdluzszych++;
45                  }
46              }
47          }
48      }
49      if(najdluzszy>1)
50      {
51          cout<<"Najdluzszy podciag zawiera "<<najdluzszy<<"
    wyrazow:"<<endl;
52          cout<<"Wyjscie: "<<endl;
53          for(int i=0; i<iloscNajdluzszych; i++)       //petla
    zewnetrzna wypisywania wartosci podciagow
54          {
55              cout<<"{ ";
56              for(int j=poczatekPodciagow[i]; j<=koniecPodciagow[i];
    j++)  //petla wewnetrzna wypisywania wartosci podciagow
57              {
58                  cout<<A[j]<<" ";
59              }
60              cout<<"}"<<endl;
```

```
61          }
62          cout<<"Ilosc znalezionych podciagow:
    "<<iloscNajdluzszych<<endl<<endl;
63      }
64      else {cout<<"Nie znaleziono najwiekszego podciagu.";}
65
66       // Zwolnienie pamieci
67      delete [] poczatekPodciagow;
68      delete [] koniecPodciagow;
69  }
70
71  //funkcja szukajaca najdluzszego malejaceego podciagu - wersja
    wydajniejsza
72  void ZnajdzNajdluzszyPodciagMalejacy2(int A[], int N)
    //wydajniejszy
73  {
74      int dlugosc=1;
75      int najdluzszy=0;
76      int IloscNajdluzszych=0;
77
78      int *PoczatekNajdluzszego=new int[N];  //dynamiczne alokowanie
    tablicy przechowujacej indeks pierwszej liczby podciagu
79      int *KoniecNajdluzszego=new int[N];   //dynamiczne alokowanie
    tablicy przechowujacej indeks ostatniej liczby podciagu
80
81      //algorytm
82      for(int i=1; i<=N; i++)
83      {
84          if(i<N && A[i]<A[i-1]) //por wnanie elementow w tablicy A
85          {
86              dlugosc++;
87          }
88          else
89          {
90              if(dlugosc>najdluzszy)
91              {
92                  najdluzszy=dlugosc;
93                  IloscNajdluzszych=0;
94                  PoczatekNajdluzszego[IloscNajdluzszych]=i-dlugosc;
    //tablica zapisujaca indeks poczatkowy znalezionego podciagu
```

```
95                  KoniecNajdluzszego[IloscNajdluzszych]=i-1;
    //tablica zapisujaca indeks koncowy znalezionego podciagu
96                  IloscNajdluzszych++;  //zwiekszamy ilosc
    najdluzszych podciagow w razie wystapienia wiecej niz jednego
    podciagu takiej samej dlugosci
97              }
98              else if(dlugosc==najdluzszy)
99              {
100                 PoczatekNajdluzszego[IloscNajdluzszych]=i-dlugosc;
     //tablica zapisujaca indeks poczatkowy znalezionego podciagu
101                 KoniecNajdluzszego[IloscNajdluzszych]=i-1;
    //tablica zapisujaca indeks koncowy znalezionego podciagu
102                 IloscNajdluzszych++;
103             }
104             dlugosc=1;
105         }
106     }
107
108     if(najdluzszy>1)
109     {
110         cout<<"Najdluzszy podciag zawiera "<<najdluzszy<<"
    wyrazow:"<<endl;
111         cout<<"Wyjscie: "<<endl;
112         for(int i=0; i<IloscNajdluzszych; i++)    //petla
    zewnetrzna wypisywania wartosci podciagow
113         {
114             cout<<"{ ";
115             for(int j=PoczatekNajdluzszego[i];
    j<=KoniecNajdluzszego[i]; j++)  //petla wewnetrzna wypisywania
    wartosci podciagow
116             {
117                 cout<<A[j]<<" ";
118             }
119             cout<<"}"<<endl;
120         }
121         cout<<"Ilosc znalezionych podciagow:
    "<<IloscNajdluzszych<<endl;
122     }
123     else {cout<<"Nie znaleziono najwiekszego podciagu.";}
124
```

```
125       // Zwolnienie pamieci
126     delete[] PoczatekNajdluzszego;
127     delete[] KoniecNajdluzszego;
128 }
129
130 int main()
131 {
132     srand(time(NULL));
133     for(int test=1; test<4; test++)
134     {
135         cout<<"TEST NR: "<<test<<endl;
136         int N=rand()% 30 + 10;           //losowanie zakresu danych
137         cout<<"Rozmiar tablicy wynosi: "<<N<<endl;
138         cout<<"Wejscie: "<<endl;
139         int *A=new int[N];              //tworzenie tablicy
    dynamicznej A[] przechowujacej dane
140         cout<<"{ ";
141         for(int i=0; i<N; i++)
142         {
143             A[i]=rand()%201-100;     //generowanie losowych liczb z
    zakresu od -100 do 100
144             cout<<A[i]<<" ";
145         }
146         cout<<"}"<<endl;
147         //obliczanie wyniku dwoma metodami
148         cout<<endl<<"Program 1: "<<endl;
149         ZnajdzNajwiekszyPodciagMalejacy(A, N);
150         cout<<"Program 2: "<<endl;
151         ZnajdzNajdluzszyPodciagMalejacy2(A, N);
152         cout<<endl;
153
154         // Zwolnienie pamieci
155         delete[] A;
156         cout<<endl<<endl;
157     }
158
159
160     return 0;
161 }
```

POLITECHNIKA RZESZOWSKA im. I. Łukasiewicza Rzeszów, 2025
Wydział Elektrotechniki i Informatyki

STRESZCZENIE PRACY DYPLOMOWEJ WPISZ-RODZAJ-PRACY

WYSZUKIWANIE NAJDŁUŻSZEGO MALEJĄCEGO PODCIĄGU

Autor: Barbara Majkut, nr albumu: 179954
Opiekun: (dr inż.) Mariusz Borkowski (prof. PRz)
Słowa kluczowe: (max. 5 słów kluczowych w 2 wierszach, oddzielanych przecinkami)

Treść streszczenia po polsku

RZESZOW UNIVERSITY OF TECHNOLOGY Rzeszow, 2025
Faculty of Electrical and Computer Engineering

WPISZ-RODZAJ-PRACY THESIS ABSTRACT

SEARCHING FOR THE LONGEST DESCENDING SUBSTRING

Author: Barbara Majkut, nr albumu: 179954
Supervisor: (academic degree) Mariusz Borkowski
Key words: (max. 5 słów kluczowych w 2 wierszach, oddzielanych przecinkami)

Treść streszczenia po angielsku