Algorytmy i struktury danych

Zadanie programistyczne

Autor:

Szymon Jucha

Data:

27 stycznia 2026

Spis treści

1 Treść zadania

Wypisz wszystkie elementy tablicy dwuwymiarowej MxN w kolejności wyznaczanej przez kierunek antyprzekątnej.

Przykład wejściowy:

$$\begin{bmatrix} 1 & 5 & 8 & 5 \\ 8 & 2 & 9 & 2 \\ 2 & 1 & 6 & 0 \end{bmatrix}$$

Wyjście:

```
1
8 5
2 2 8
1 9 5
6 2
0
```

2 Etapy rozwiązywania problemu

2.1 Rozwiązanie pierwsze

2.1.1 Analiza zadania

Zadanie polega na wypisywaniu każdej liczby z tablicy o wymiarach MxN jeden raz. To powoduje, że optymalny program powinien mieć złożoność liniową $O(M \cdot N)$. Do wykonania tego najlepiej jest działać na indeksach tablicy, czyli trzeba zauważyć taką zależność, że suma indeksu i (wierszy) oraz j (kolumn) daje indeks przekątnej p, licząc przekątne od zera i zaczynając od lewego górnego rogu tak jak w treści zadania. Na przykład liczba na pozycji (0 0) należy do przekątnej o indeksie 0, a (1 1) do przekątnej 2. Dzięki temu możemy iterować po każdej przekątnej p po kolei, co daje nam pewność że nie będziemy dwukrotnie przechodzić przez jedną liczbę. Każda przekątna rozpoczyna się albo w punkcie $i = p$, lub $i = M - 1$, gdy numer przekątnej przekroczy rozmiar tablicy. Do tego wystarczy dać warunek $min(p, (M - 1))$. Za to indeks i na końcu przekątnej jest równy 0 lub $i = p - (N - 1)$, gdy $p > (N - 1)$, więc do znalezienia najmniejszego i wystarczy warunek $max(0, p - (N - 1))$. Skoro $i + j = p$, to żeby wyznaczyć j dla każdego punktu wystarczy warunek $j = p - i$. Teraz żeby przechodzić po całej przekątnej wystarczy iterować po każdym i od $min(p, M-1)$ do $max(0, p-(N-1))$ i wyświetlać każdą liczbę po kolei.

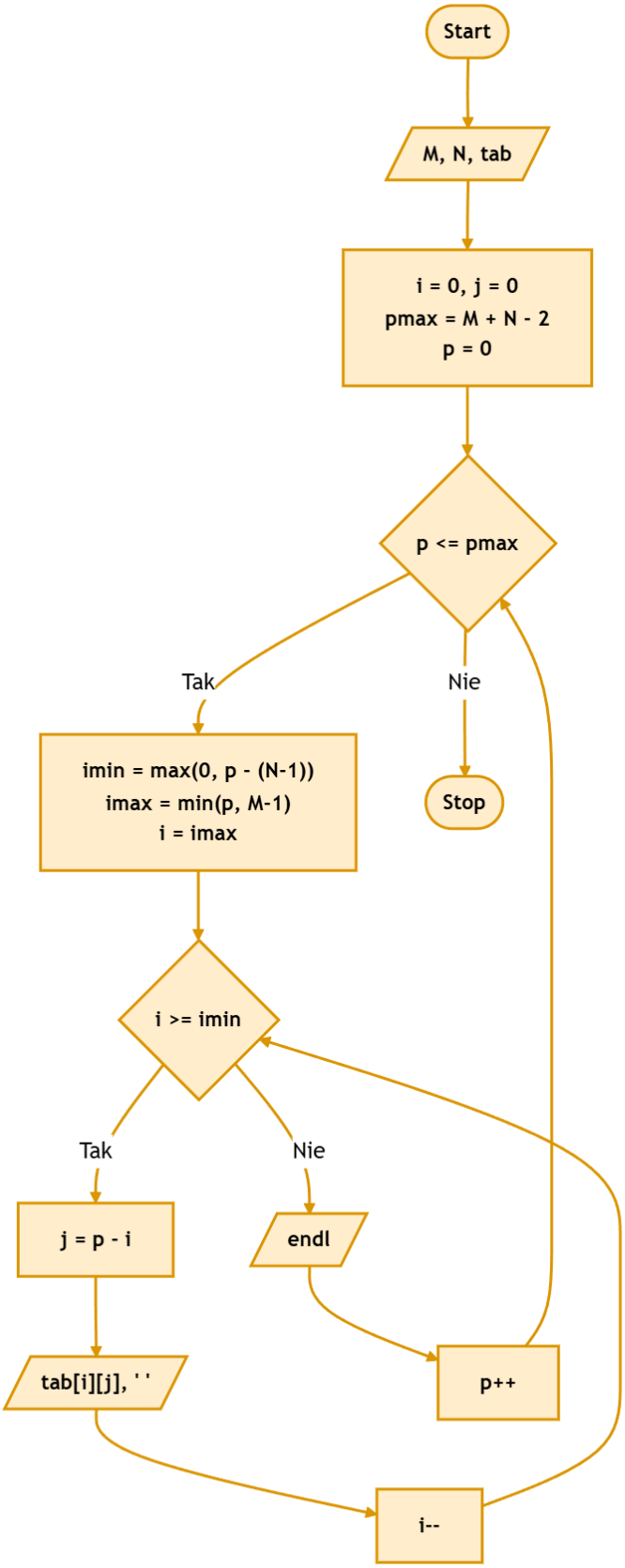
Dane wejściowe:

- Wielkość tablicy (zmienne M, N),
- Struktura danych typu tablica (zmienna `tab`), przechowująca wartości.

Dane wyjściowe:

- Po kolei wypisane liczby względem antyprzekątnej.

2.1.2 Schemat blokowy algorytmu



Rysunek 1: Schemat blokowy pierwszego algorytmu.

2.1.3 Algorytm zapisany w pseudokodzie

```
Input M, N // rozmiary tablicy
w // tablica dwuwymiarowa
imin := 0
imax := 0
i := 0
j := 0
for p := 0 to M + N - 2
    imin := max(0, p - (N - 1))
    imax := min(p, M - 1)
    i := imax
    while i >= imin
        j := p - i
        wypisz w[i][j]
        wypisz " "
        i := i - 1
    endwhile
    wypisz nowa_linia
endfor
```

2.1.4 Ołówkowe sprawdzenie poprawności algorytmu

p	i	j	tab[i][j]
0	0	0	1
1	1	0	8
	0	1	5
2	2	0	2
	1	1	2
	0	2	8
3	2	1	1
	1	2	9
	0	3	5
4	2	2	6
	1	3	2
5	2	3	0

2.1.5 Wykazanie poprawności algortymu

1. Dowód poprawności częściowej (Zastosowanie niezmiennika)

   Problem rozważany w algorytmie można opisać jako znajdowanie odpowiednich par indeksów $(i,\ j)$ spełniających warunek $i + j = p$ w odpowiedniej kolejności.
   Można stwierdzić, że jako niezmiennik algortymu można przyjąć stwierdzenie: "Odpowiednia kolejność wyświetlania jest wtedy, gdy zaczniemy wypisywać dane od największego możliwego indeksu i zachowując rozmiary macierzy i warunek $i + j = p$, czyli $j = p - i$.

   Stwierdzamy, że ten warunek:

   - jest prawdziwy przed pierwszą iteracją,
   - każda kolejna iteracja zachowuje kolejność wyświetlania poprzez zaczynanie indeksowania i od $min(p, (M-1))$ i kończąc na $max(0, p-(N-1))$, spełniając warunek $i + j = p$

   Po zakończeniu pętli wszystkie dane z macierzy zostaną rozpatrzone i wypisane w odpowiedniej kolejności, zatem niezmiennik implikuje warunek końcowy.

2. Wykazanie własności stopu (Zastosowanie półniezmiennika)

   a) Pętla wewnętrzna Rozważmy pętlę: while $i \geq imin$
   Jako półniezmiennik przyjmujemy funkcję: $V_1(i) = i - imin$
   Zauważmy, że:

      - $V_1(i) \in N$
      - w każdej iteracji pętli zmienna i jest dekrementowana
      - $V_1(i) \geq 0$, a dla $i = imin - 1$ warunek pętli przestaje być spełniony.

      Stąd wynika, że pętla wewnętrzna kończy się po skończonej liczbie iteracji.

   b) Pętla zewnętrzna Rozważmy pętlę: for $p = 0; p \leq (M + N - 2); p++$
   Jako półniezmiennik przyjmujemy funkcję: $V_2(p) = (M + N - 2) - p$
   Zauważmy, że:

      - $V_2(p) \in N$
      - w każdej iteracji pętli zmienna p jest inkrementowana
      - $V_2(p) \geq 0$, a dla $p = (M + N - 1)$ warunek pętli przestaje być spełniony.

      Stąd wynika, że pętla zewnętrzna kończy się po skończonej liczbie iteracji.

   c) Wniosek:
   Ponieważ zarówno pętla wewnętrzna, jak i pętla zewnętrzna kończą się po skończonej liczbie kroków, cały algorytm posiada własność stopu.

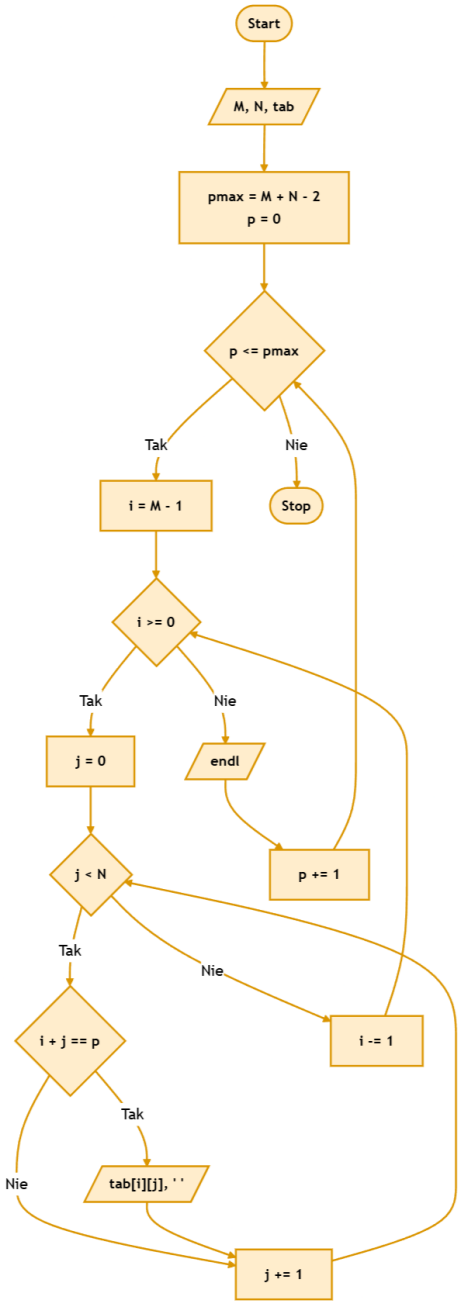
3. Podsumowanie:
   Poprzez wykazanie warunku stopu i poprawności częściowej udowodniono poprawność przedstawionego algorytmu.

2.2 Rozwiązanie drugie

2.2.1 Analiza zadania

Nadal pozostajemy przy zależności, że $i + j = p$ bo bez tego ciężko wykonać dobrze to zadanie. Teraz jednak zamiast perfekcyjnego wybierania współrzędnych tablicy przechodzimy po całej tablicy tyle razy ile jest przekątnych, czyli $(M + N)$-razy, sprawdzając tylko funkcją warunkową czy $i+j = p$. Powoduje to, że złożoność obliczeniowa tym razem wynosi $O((M + N - 1) \cdot M \cdot N)$.

2.2.2 Schemat blokowy



Rysunek 2: Schemat blokowy drugiego algorytmu.

2.2.3 Algorytm zapisany w pseudokodzie

```
input: tab, M, N
pmax := (M - 1) + (N - 1)
for p := 0 to pmax
    for i := M - 1 downto 0
        for j := 0 to N - 1
            if i + j = p
                wypisz tab[i][j]
            endif
        endfor
    endfor
    wypisz nowa_linia
endfor
```

2.2.4 Ołówkowe sprawdzenie poprawności algorytmu

p	i	j	i+j=p	tab[i][j]
0	2	0	nie	–
0	2	1	nie	–
0	2	2	nie	–
0	2	3	nie	–
0	1	0	nie	–
0	1	1	nie	–
0	1	2	nie	–
0	1	3	nie	–
0	0	0	TAK	1
0	0	1	nie	–
0	0	2	nie	–
0	0	3	nie	–
1	2	0	nie	–
1	2	1	nie	–
1	2	2	nie	–
1	2	3	nie	–
1	1	0	TAK	8
1	1	1	nie	–
1	1	2	nie	–
1	1	3	nie	–
1	0	0	nie	–
1	0	1	TAK	5
1	0	2	nie	–
1	0	3	nie	–
2	2	0	TAK	2
2	2	1	nie	–
2	2	2	nie	–
2	2	3	nie	–
2	1	0	nie	–
2	1	1	TAK	2
2	1	2	nie	–
2	1	3	nie	–
2	0	0	nie	–
2	0	1	nie	–
2	0	2	TAK	8
2	0	3	nie	–

p	**i**	**j**	**i+j=p**	**tab[i][j]**
3	2	0	nie	–
3	2	1	TAK	1
3	2	2	nie	–
3	2	3	nie	–
3	1	0	nie	–
3	1	1	nie	–
3	1	2	TAK	9
3	1	3	nie	–
3	0	0	nie	–
3	0	1	nie	–
3	0	2	nie	–
3	0	3	TAK	5
4	2	0	nie	–
4	2	1	nie	–
4	2	2	TAK	6
4	2	3	nie	–
4	1	0	nie	–
4	1	1	nie	–
4	1	2	nie	–
4	1	3	TAK	2
4	0	0	nie	–
4	0	1	nie	–
4	0	2	nie	–
4	0	3	nie	–
5	2	0	nie	–
5	2	1	nie	–
5	2	2	nie	–
5	2	3	TAK	0
5	1	0	nie	–
5	1	1	nie	–
5	1	2	nie	–
5	1	3	nie	–
5	0	0	nie	–
5	0	1	nie	–
5	0	2	nie	–
5	0	3	nie	–

2.2.5 Wykazanie poprawności algortymu

1. Dowód poprawności częściowej (Zastosowanie niezmiennika)

   Problem rozważany w algorytmie można opisać jako znajdowanie odpowiednich par indeksów $(i,\ j)$ spełniających warunek $i + j = p$ w odpowiedniej kolejności.
   Można stwierdzić, że jako niezmiennik algortymu można przyjąć stwierdzenie: "Odpowiednia kolejność wyświetlania jest wtedy, gdy zaczniemy wypisywać dane od największego możliwego indeksu i i możliwie najmniejszego indeksu j zachowując rozmiary macierzy i warunek $i + j = p$.

   Stwierdzamy, że ten warunek:

   - jest prawdziwy przed pierwszą iteracją,
   - każda kolejna iteracja obu pętli zachowuje kolejność wyświetlania poprzez zaczynanie indeksowania i od $i = M - 1$ oraz j od $j = 0$ spełniających warunek $i + j = p$

   Po zakończeniu pętli wszystkie dane z macierzy zostaną rozpatrzone i wypisane w odpowiedniej kolejności, zatem niezmiennik implikuje warunek końcowy.

2. Wykazanie własności stopu (Zastosowanie półniezmiennika)

   a) Pętle wewnętrzne Rozważmy pętlę: for $i = M - 1; i \geq 0; i - -$ oraz for $j = 0; j \leq N - 1; j + +$

      Jako półniezmiennik przyjmujemy funkcję: $V_1(i) = i$ i $V_2(j) = (N - 1) - j$
      Zauważmy, że:

      - $V_1(i), V_2(j) \in N$
      - w każdej iteracji pętli zmienna i jest dekrementowana, a j inkrementowana
      - $V_1(i) \geq 0$, a dla $i = -1$ warunek pętli przestaje być spełniony.
      - $V_2(j) \geq 0$, a dla $j = N$ warunek pętli przestaje być spełniony.

      Stąd wynika, że pętle wewnętrzne kończą się po skończonej liczbie iteracji.

   b) Pętla zewnętrzna Rozważmy pętlę: for $p = 0; p \leq (M + N - 2); p + +$
      Jako półniezmiennik przyjmujemy funkcję: $V_3(p) = (M + N - 2) - p$
      Zauważmy, że:

      - $V_3(p) \in N$
      - w każdej iteracji pętli zmienna p jest inkrementowana
      - $V_3(p) \geq 0$, a dla $p = (M + N - 1)$ warunek pętli przestaje być spełniony.

      Stąd wynika, że pętla zewnętrzna kończy się po skończonej liczbie iteracji.

   c) Wniosek:
      Ponieważ zarówno pętla wewnętrzna, jak i pętla zewnętrzna kończą się po skończonej liczbie kroków, cały algorytm posiada własność stopu.

3. Podsumowanie:
   Poprzez wykazanie warunku stopu i poprawności częściowej udowodniono poprawność przedstawionego algorytmu.

2.3 Implementacja algorytmów w środowisku Code::Blocks IDE w języku C/C++ oraz eksperymentalne potwierdzenie wydajności algorytmów.

2.3.1 Implementacja algorytmów

Rozważmy podejście, w którym cały program umieszczamy w jednym pliku. Aby zachować modularność oba algorytmu zapiszemy w postaci oddzielonych funkcji.
Przykład tego rodzaju programu przedstawia się następująco:

```
#include <iostream>
#include <vector>
#include <ctime>
#include <fstream>
#include <chrono>

using namespace std;

void AntyPrzekatna1(int M, int N, vector<vector<int>> tab)
{
    int i = 0, j = 0;
    int imin = 0, imax = 0;

    for(int p = 0; p <= M + N - 2; p++){
        imin = max(0, p - (N - 1));
        imax = min(p, M - 1);
        i = imax;

        while(i >= imin){
            j = p - i;
            cout << tab[i][j] << " ";
            i--;
        }
        cout << endl;
    }
}

void AntyPrzekatna2(int M, int N, vector<vector<int>> tab)
{
    int pmax = (M - 1) + (N - 1);

    for(int p = 0; p <= pmax; p++){
        for(int i = M - 1; i >= 0; i--){
            for(int j = 0; j < N; j++){
                if(i + j == p){
                    cout << tab[i][j] << " ";
                }
            }
        }
        cout << endl;
    }
}
```

```
43
44 int main()
45 {
46     ios_base::sync_with_stdio(false);
47     cin.tie(NULL);
48
49     int M, N;
50     cin >> M >> N;
51
52     vector<vector<int>> tab(M, vector<int>(N));
53     srand(time(NULL));
54
55     for(int i = 0; i < M; i++){
56         for(int j = 0; j < N; j++){
57             tab[i][j] = rand() % 10;
58         }
59     }
60
61     AntyPrzekatna1(M, N, tab);
62     AntyPrzekatna2(M, N, tab);
63
64
65     return 0;
66 }
```

2.3.2 Test wydajności algorytmów

Użyjmy teraz poniższego programu do przetestowania czasu działania poszczególnych algorytmów na większych przykładach by dokładnie zobaczyć jak bardzo różni się czas działania obu algorytmów.

```
1      #include <iostream>
2  #include <vector>
3  #include <ctime>
4  #include <fstream>
5  #include <chrono>
6
7  using namespace std;
8
9  void AntyPrzekatna1(int M, int N, vector<vector<int>> tab)//
       Najbardziej optymalny algorytm
10 {
11     int i=0, j=0;
12     int imin=0, imax=0;
13     for(int p=0;p<=M+N-2;p++){
14         imin=max(0,p-(N-1));
15         imax=min(p, (M-1));
16         i=imax;
17         while(i>=imin){
18             j=p-i;
19             cout<<tab[i][j]<<" ";
```

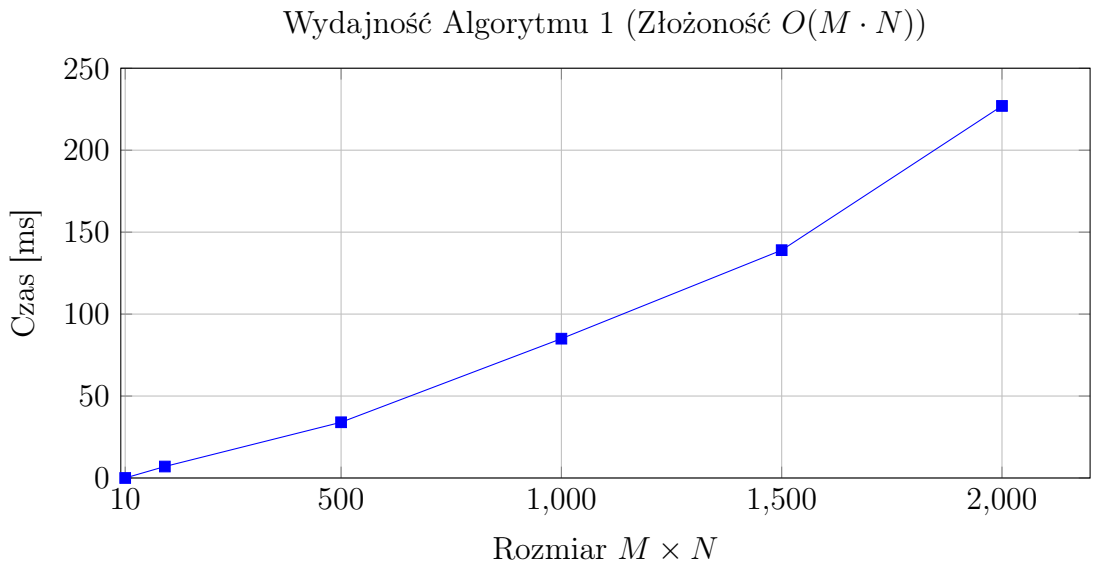
```
20              i--;
21          }
22          cout<<endl;
23      }
24  }
25
26  void AntyPrzekatna2(int M, int N, vector<vector<int>> tab)//Mniej
        optymalny algorytm
27  {
28      int pmax = (M-1)+(N-1);
29      for(int p=0;p<=pmax;p++){
30          for(int i=M-1;i>=0;i--){
31              for(int j=0;j<N;j++){
32                  if(i+j==p){
33                      cout<<tab[i][j]<<" ";
34                  }
35              }
36
37          }
38          cout<<endl;
39      }
40  }
41
42  int main()
43  {
44      ios_base::sync_with_stdio(false);//Przyspieszenie operacji
            wejscia wyjscia
45      cin.tie(NULL);
46      fstream plik;
47      plik.open("Proba.txt", ios::in);//Uzyskanie dostepu do pliku
48
49      int M,N;
50
51      cin>>M>>N;
52      //plik>>M>>N;
53      vector<vector<int>> tab (M, vector<int>(N)); //Tworzenie
            vectora dwuwymiarowego o rozmiarach MxN
54      srand(time(NULL)); //Generowanie pseudo losowych
55      for(int i=0;i<M;i++){
56          for(int j=0;j<N;j++){
57               tab[i][j]=rand()%10;
58          }
59      }
60
61      /*for(int i=0;i<M;i++){//Wczytywanie danych z pliku
62          for(int j=0;j<N;j++){
63              plik>>tab[i][j];
64          }
65      }*/
66
67
```

```
68      /*for(int i=0;i<M;i++){//Wyswietlenie tablicy przed wykonaniem
            algorytmu
69          for(int j=0;j<N;j++){
70                cout<<tab[i][j]<<" ";
71          }
72          cout<<endl;
73      }*/
74      //Porownywanie czasu algorytmow
75      auto start = std::chrono::high_resolution_clock::now();
76      AntyPrzekatna1(M,N,tab);
77      auto stop = std::chrono::high_resolution_clock::now();
78      auto duration = std::chrono::duration_cast<std::chrono::
            milliseconds>(stop - start);
79
80      std::cout << "Czas wykonywania: " << duration.count() << " ms"
            << std::endl;
81
82      start = std::chrono::high_resolution_clock::now();
83      AntyPrzekatna2(M,N,tab);
84      stop = std::chrono::high_resolution_clock::now();
85      duration = std::chrono::duration_cast<std::chrono::milliseconds
            >(stop - start);
86
87      std::cout << "Czas wykonywania: " << duration.count() << " ms"
            << std::endl;
88
89
90      plik.close();
91
92      return 0;
93  }
```

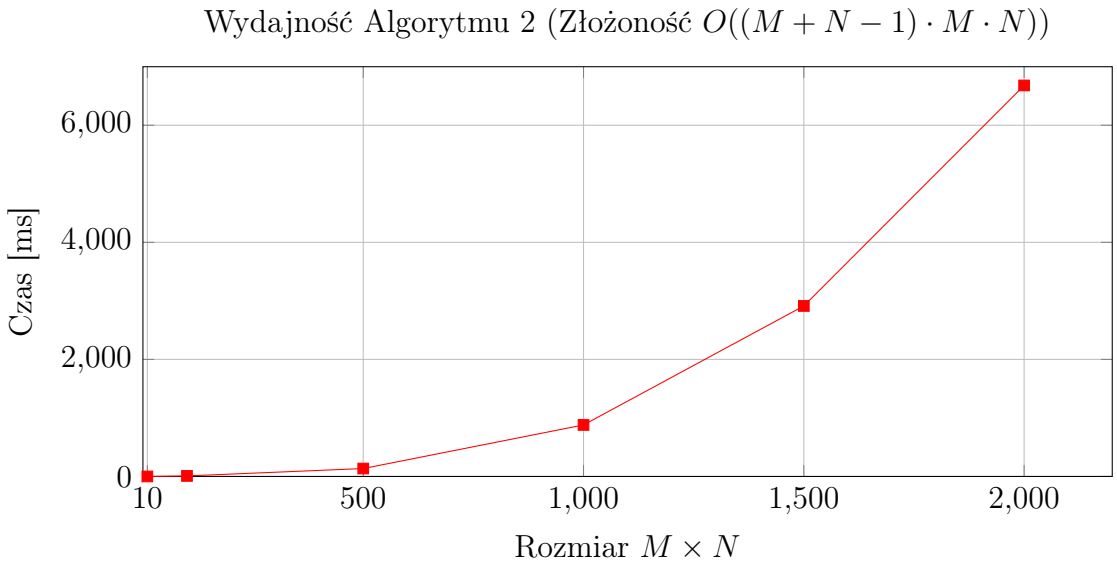
Poniższa tabela przedstawia porównanie czasu działania obu algorytmów w zależności od rozmiaru macierzy wejściowej $M \times N$.

M	**N**	**t1 [ms]**	**t2 [ms]**
10	10	0	0
100	100	7	9
500	500	34	135
1000	1000	85	880
1500	1500	139	2913
2000	2000	227	6680

Jak widać na tabeli, obliczenia wykonywane przy pomocy algortymu w wersji pierwszej trwają o wiele krócej niż w przypadku drugiego algorytmu który jest przykładem mniej przemyślanego algorytmu. Analizując wykresy poniżej można dostrzec, że czas drugiego algorytmu dużo szybciej odbija do góry, podczas gdy pierwszy wykres pozostaje względnie liniowy tak jak oczekiwaliśmy na początku.



Rysunek 3: Zależność czasu t_1 od rozmiaru danych.



Rysunek 4: Zależność czasu t_2 od rozmiaru danych.