

Algorytmy i Struktury Danych

Zadanie Projektowe

Norbert Tabisz
Inżynieria i Analiza Danych, grupa: 9
nr. indeks (184309)
Politechnika Rzeszowska

22 stycznia 2026

Spis treści

1 Problematyka	3
1.1 Treść zadania	3
1.2 Analiza problemu	3
2 Rozwiązywanie problemu metodą brute-force	3
2.1 Schemat blokowy	3
2.2 Pseudokod	5
2.3 "Ołówkowe" sprawdzenie	6
2.4 Złożoność obliczeniowa algorytmu brute-force	6
2.5 Implementacja metodą brute-force w C++	7
2.6 Wynik działania programu	10
3 Rozwiązywanie metodą sortowania	11
3.1 Schemat blokowy	12
3.2 Pseudokod programu zapisanego metodą z sortowaniem	14
3.3 Złożoność obliczeniowa drugiej metody	15
3.4 Kod programu metodą sortowania	15
3.5 Wynik działania drugiego programu	17
4 Podsumowanie	18
4.1 Porównanie obu złożoności	18
4.2 Testy wydajnościowe	19
4.3 Wnioski	20

1 Problematyka

1.1 Treść zadania

Dla zadanej tablicy liczb całkowitych znajdź te pary, których różnica jest równa zadanej liczbie k .

Wejście: tablica liczb całkowitych $A[]$, liczba całkowita k

Wyjście: wszystkie nie powtarzające się pary liczb (a, b) takie, że $|a - b| = k$

Przykład:

Wejście:

$A[] = [1,5,2,2,2,5,5,4]$

$k = 3$

Wyjście: $[2,5]$ oraz $[1,4]$

1.2 Analiza problemu

Danym wejściem jest tablica liczb całkowitych i liczba całkowita k . Celem jest znalezienie wszystkich par elementów tablicy, których różnica wynosi k .

Najprostszym podejściem do rozwiązywania problemu jest sprawdzenie wszystkich możliwych par elementów tablicy poszczególnie ze sobą i obliczenie różnicy dla każdej z nich.

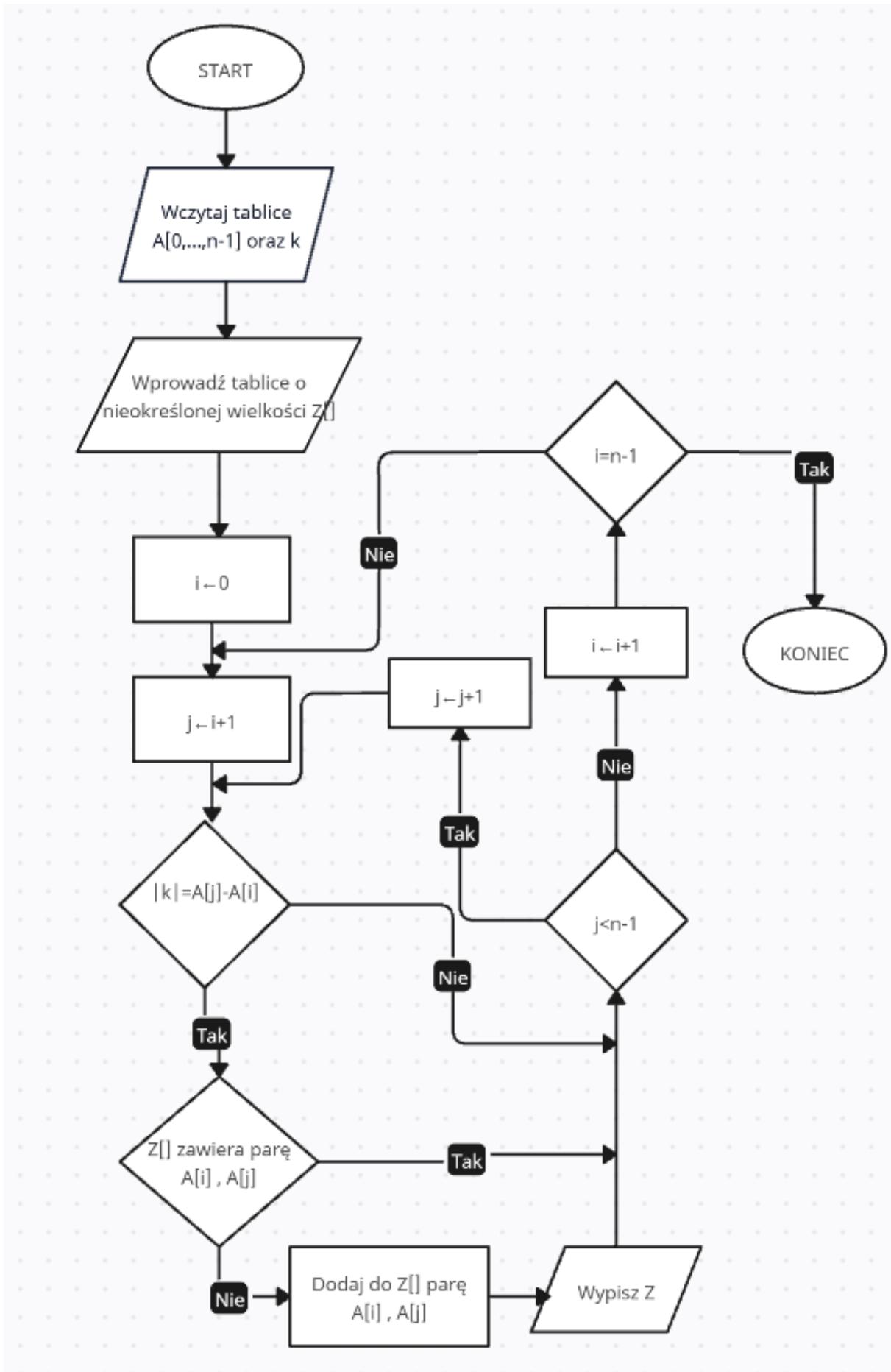
Analizując ten problem można zauważyc, że kolejność elementów w parze ma znaczenie, ponieważ $a - b \neq b - a$. Znaczy to tyle, że trzeba każdą parę rozpatrywać osobno. Mimo najprostszego działania metoda brute-force nie zawsze jest tą najoptymalniejszą, dlatego warto rozważyć rozwiązanie tego problemu inną metodą przedstawioną w dalszym etapie.

2 Rozwiązywanie problemu metodą brute-force

2.1 Schemat blokowy

W poniższym schemacie blokowym oraz pseudokodzie przyjęto założenie, że tablice indeksowane są od 0.

Algorytm zapisany w postaci schematu blokowego przedstawia się następująco:



Rysunek 1: Schemat blokowy algorytmu brute-force

Algorytm wykorzystuje trzy zmienne pomocnicze wymagające wstępnej inicjalizacji. Liczniki i oraz j służą jako indeksy do nawigacji po strukturze tablicy, natomiast tablica $Z[]$ przechowuje pasujące wartości o podanej różnicy k .

2.2 Pseudokod

Tworze pseudokod programu.

```
1. Array A, k           // A - tablica z wartościami, k-dana
   roznica
2. n                   // wielkość tablicy
3. Z                   // pusta tablica par
4.
5. for i <- 0 to n 1
6.   for j <- i+1 to n 1
7.     if |A[j] - A[i]| = |k|
8.       if A[j] - A[i] = k
9.         x<-A[i]
10.        y<-A[j]
11.      else
12.        x<-A[j]
13.        y<-A[i]
14.    end if
15.    if pair (x, y) is not in Z
16.      add (x, y) to Z
17.    end if
18.  end if
19. end for
20. end for
21.
22. if Z empty
23.   print "Brak par o roznicy k"
24. else
25.   print Z
26. end if
```

Listing 1: Pseudokod metodą brute-force

2.3 "Ołówkowe" sprawdzenie

Poniżej jest przedstawione "ołówkowe" sprawdzenie aby wychwycić potencjalne błędy lub niezgodności algorytmu. Dla przykładowych wartości w tablicy A oraz różnicy k między wartościami.

A=[1,5,2,4,3] oraz k=3

k	i	j	A[i]	A[j]	A[j]-A[i] = k
3	0	1	1	5	0
3	0	2	1	2	0
3	0	3	1	4	1
3	0	4	1	3	0
3	1	2	5	2	0
3	1	3	5	4	0
3	1	4	5	3	0
3	2	3	2	4	0
3	2	4	2	3	0
3	3	4	4	3	0

Jedynym prawidłowem wynikiem w tym przypadku jest [1, 4].

2.4 Złożoność obliczeniowa algorytmu brute-force

Aby obliczyć złożoność obliczeniową tego algorytmu, musimy przeanalizować liczbę operacji dominujących(tych wykonujących się wewnątrz pętli). Algorytm opiera się na dwóch zagnieżdzonych pętlach, które porównują pary elementów tablicy o rozmiarze n .Algorytm porównuje i-tą i j-tą, wartości tabeli. Liczba i-ta zaczyna się od $i=0$ do $i=n-1$, więc wykonuje się $n-1$ razy.Liczba j-ta natomiast zaczyna się od $j=i+1$ do $j=n-1$.Liczba iteracji pętli wewnętrznej zmniejsza razem ze wzrostem i .

- dla $i = 0, j$ idzie od 1 do $n - 1$ ($n - 1$ porównań).
- dla $i = 1, j$ idzie od 2 do $n - 1$ ($n - 2$ porównań).
- dla $i = 2, (n - 3$ porównań)
- ...

Łączną liczbę porównań ($k = A[j] - A[i]$) określa suma ciągu arytmetycznego:

$$(n - 1) + (n - 2) + (n - 3) + \dots + 1 = \frac{(n - 1) \cdot n}{2} = \frac{n^2 - n}{2}$$

Możemy go przekształcić w taki sposób aby dostać ilość porównań:

$$\frac{1}{2}n^2 - \frac{1}{2}n$$

dla n-elementowego ciągu, algorytm wykona się:

$$\frac{1}{2}n^2 - \frac{1}{2}n$$

Dominującym składnikiem jest n^2 . W notacji dużego O pomijamy stałe oraz składniki niższych rzędów co daje nam złożoność obliczeniową $O(n^2)$

2.5 Implementacja metodą brute-force w C++

Na podstawie powyższego schematu blokowego i pseudokodu napisałem kod programu w języku C++. Podany kod podzieliłem na funkcje dla łatwiejszego wyjaśnienia działania programu.

```
1 #include <iostream>
2 #include <vector>
3 #include <cmath>
4 #include <utility>
5 using namespace std;
6
7 void wyswietlTablice(int tab[], int rozmiar) {
8     cout << "Przykładowa tablica A: ";
9     for (int i = 0; i < rozmiar; i++) {
10         cout << tab[i];
11         if (i < rozmiar - 1) cout << ", ";
12     }
13     cout << "]" << endl;
14 }
```

Listing 2: Funkcja odpowiedzialna za wyświetlanie tablicy

Zaimplementowałem algorytm sprawdzający czy 2 pary zawierają te same elementy nie zależnie od kolejności.

```
1 bool czyTeSameElementy(const pair<int,int> para1, const pair<int,
2                         int> para2) {
3     return (para1.first == para2.first && para1.second == para2.
4             second) || (para1.first == para2.second && para1.second ==
5                         para2.first);
6 }
```

Listing 3: Funkcja logiczna czyTeSameElementy sprawdzającą czy para się powtarza

Poniższa funkcja sprawdza czy kolejne wartości z tablicy są równe podanej roznicy k. Ponadto dba o poprawne i zrozumiałe wyświetlenie danej pary w zależności od podanego znaku roznicy. Funkcja znajdzPary jest też odpowiedzialna za sprawdzanie czy dana para już wcześniej nie wystąpiła aby uniknąć powtórzeń. W tym fragmencie kodu użyto wbudowany typ danych, wektor który jest rodzajem tablicy dynamicznej.

```

1   vector<pair<int ,int>> znajdzPary( int A[], int n, int k) {
2     vector<pair<int ,int>> Z;
3
4     for (int i = 0; i < n; i++) {
5       for (int j = i + 1; j < n; j++) {
6
7         if (abs(A[j] - A[i]) == abs(k)) {
8           pair<int ,int> potencjalnaPara;
9
10          if (A[j] - A[i] == k) {
11            potencjalnaPara.first = A[i];
12            potencjalnaPara.second = A[j];
13          } else {
14            potencjalnaPara.first = A[j];
15            potencjalnaPara.second = A[i];
16          }
17
18          bool juz_jest = false;
19          for (int m = 0; m < Z.size(); m++) {
20            if (czyTeSameElementy(potencjalnaPara, Z[m])) {
21              juz_jest = true;
22              break;
23            }
24          }
25
26          if (!juz_jest) {
27            Z.push_back(potencjalnaPara);
28          }
29        }
30      }
31    }
32    return Z;
33 }
```

Listing 4: Funkcja znajdująca pary o roznicy k

W funkcji main pozostawiono Wczytywanie liczby k z klawiatury, sprawdzenie czy tablica odpowiedzialna za przechowywanie poprawnych par nie jest pusta oraz wyświetlanie wyniku.

```
1 int main() {
2     int A[] = {1, 5, 2, 2, 2, 5, 5, 4};
3     int n = sizeof(A) / sizeof(A[0]);
4     int k;
5
6     wyswietlTablice(A, n);
7     cout << "Podaj roznice k:" ;
8     cin >> k;
9
10    vector<pair<int,int>> Z = znajdzPary(A, n, k);
11
12    if (Z.empty()) {
13        cout << "Brak par o roznicy k." << endl;
14    } else {
15        cout << "Par o roznicy k:" ;
16        for (int i = 0; i < Z.size(); i++) {
17            cout << "[" << Z[i].first << "," << Z[i].second << "] ";
18        }
19        cout << endl;
20    }
21
22    return 0;
23 }
```

Listing 5: Funckja main

2.6 Wynik działania programu

```
C:\Users\Admin\Documents\c++\projekt\Wyszukiwanie par o różnicy k.exe
Przykładowa tablica A: [1, 5, 2, 2, 2, 5, 5, 4]
Podaj roznice k: 3
Pary o różnicy k: [1,4] [2,5]

-----
Process exited after 10.67 seconds with return value 0
Press any key to continue . . .
```

Rysunek 2: Przykładowe działanie dla dodatniej różnicy k

```
C:\Users\Admin\Documents\c++\projekt\Wyszukiwanie par o różnicy k.exe
Przykładowa tablica A: [1, 5, 2, 2, 2, 5, 5, 4]
Podaj roznice k: -1
Pary o różnicy k: [2,1] [5,4]

-----
Process exited after 5.877 seconds with return value 0
Press any key to continue . . .
```

Rysunek 3: Przykładowe działanie dla ujemnej różnicy k

```
C:\Users\Admin\Documents\c++\projekt\Wyszukiwanie par o roznicy k.exe
Przykładowa tablica A: [1, 5, 2, 2, 2, 5, 5, 4]
Podaj roznice k: 9
Brak par o roznicy k.

-----
Process exited after 4.439 seconds with return value 0
Press any key to continue . . .
```

Rysunek 4: Przykładowe działanie dla braku par o roznicy k

Ważnym elementem jest to że różnice k można przedstawić jako wartość bezwzględna lecz później dla trafniejszego wyświetlania wprowadzono warunek sprawdzający znak różnicy liczb $A[i]$ oraz $A[j]$.

3 Rozwiążanie metodą sortowania

Po głębszym zastanowieniu się nad przedstawionym problemem można znaleźć inne, efektywniejsze oraz mniej obciążające metody rozwiązania go. Jedną z takich metod jest metoda z sortowaniem tablicy która różni się od wyżej przedstawionej metody brute-force oraz znacząco zmniejsza złożoność obliczeniową programu.

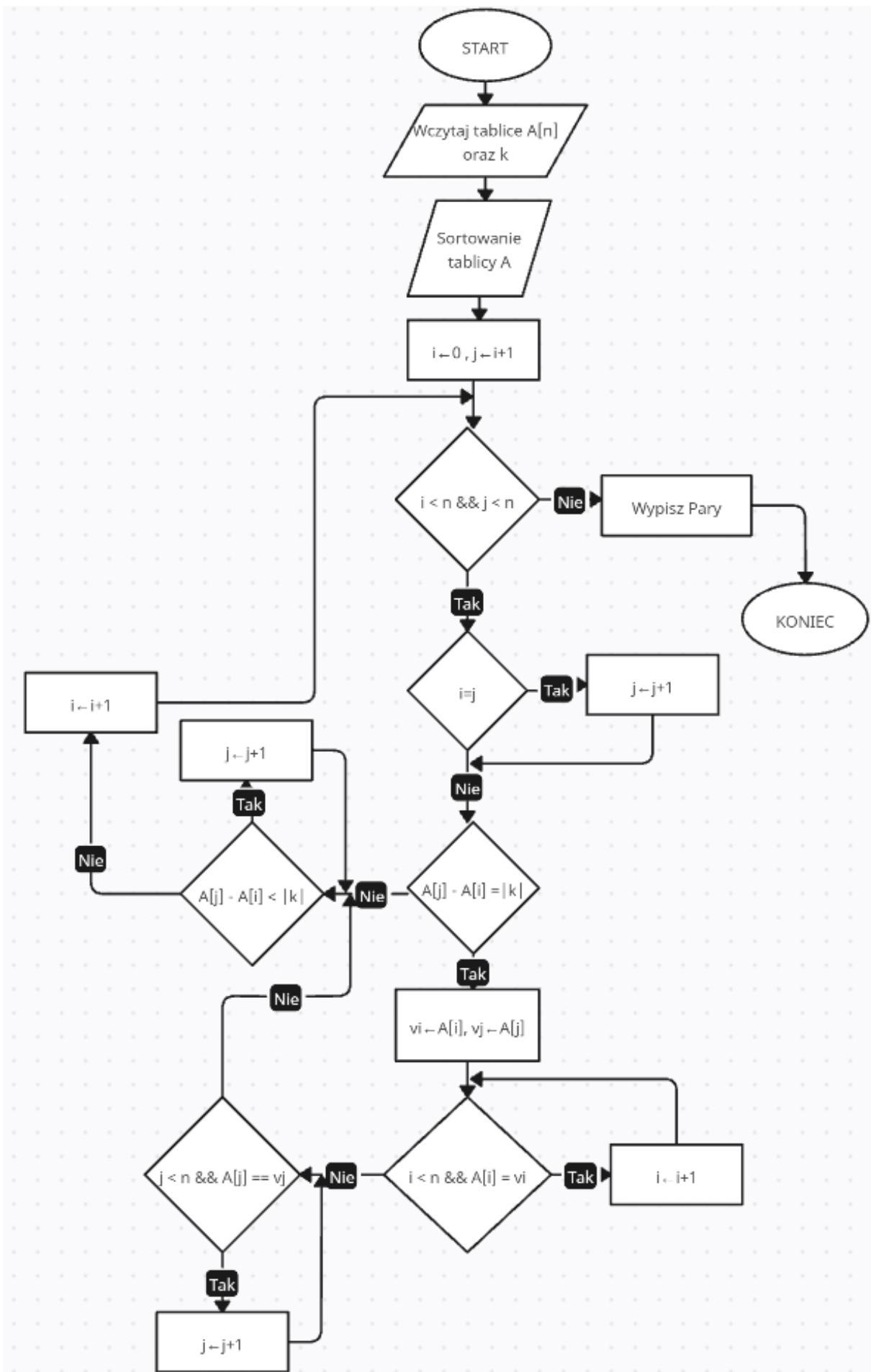
Ideą tej metody jest to że znając różnicę k i pierwszą liczbę $x=A[i]$, możemy wyznaczyć drugą liczbę $y=A[j]$. Tablice sortujemy rosnąco za pomocą gotowej funkcji. Dzięki temu nie jest konieczne sprawdzanie wielu pozycji dwukrotnie. Zarówno wartości i sprawdzamy n razy a także j, n razy co zamiast n^2 daje nam n.

$$y - x = k \rightarrow y = k + x \quad (1)$$

Ważnym elementem tej metody jest to że, sortujemy podana tablice i z powtorzonych wartości bierzemy tylko tę pierwszą. Sortowanie jest kluczowe, bo umożliwia użycie dwóch wskaźników. Pętla działa, dopóki oba wskaźniki i oraz j są w granicach tablicy. Ponieważ tablica zostaje posortowana to zawsze różnica j-i będzie większa od 0. Kod w swojej usprawnionej wersji zarówno jak i poprzedni pomija wypisywanie duplikatów par.

3.1 Schemat blokowy

Na poniższym schemacie blokowym przedstawiono działanie algorytmu dla metody z sortowaniem:



3.2 Pseudokod programu zapisanego metodą z sortowaniem

```
1 A []                                // tablica A z podanymi liczbami
2 k                                     // roznica miedzy dwoma wartosciami
3 n                                     // wielkosc tablicy A
4 print "Tablica przed sortowaniem:"
5 Print A
6
7 sort array A
8
9 print Sorted A
10
11 i <- 0
12 j <- i + 1
13
14 while i < n and j < n
15     if i = j
16         j <- j + 1
17     end if
18
19     if A[j] - A[i] = |k|
20         if k > 0
21             print (A[i], A[j])
22         else
23             print (A[j], A[i])
24         end if
25
26         vi <- A[i]
27         vj <- A[j]
28
29         while i < n and A[i] = vi
30             i <- i + 1
31         end while
32         while j < n and A[j] = vj
33             j <- j + 1
34         end while
35
36         else if A[j] - A[i] < |k|
37             j <- j + 1
38         else
39             i <- i + 1
40         end if
41     end while
```

Listing 6: Pseudokod drugiej metody

3.3 Złożoność obliczeniowa drugiej metody

Rozważmy sortowanie tablicy zawierającej n elementów.

Złożoność obliczeniową programu wyznaczono poprzez analizę liczby operacji wykonywanych w zależności od rozmiaru danych wejściowych n , gdzie n oznacza liczbę elementów tablicy. Dla notacji O uwzględniane są jedynie operacje dominujące, bo decydują o czasie działania algorytmu dla dużych wartości n .

Wyświetlanie tablicy i wypisywanie znalezionych par ma złożoność $O(n)$. Dominującym elementem jest sortowanie tablicy przy użyciu funkcji `sort()` posiadającej złożoność $O(n \log n)$. (Wytłumaczenie tej funkcji zawarte jest pod fragmentem kodu "Funkcja main"). Po posortowaniu danych wykonywany jest algorytm wyszukiwania par liczb o zadanej różnicy, oparty na technice dwóch wskaźników. Każdy wskaźnik przesuwa się wyłącznie w jednym kierunku, dzięki czemu każdy element tablicy jest odwiedzany co najwyżej jeden raz co daje złożoność $O(n)$.

Ostatecznie otrzymujemy złożoność obliczeniową $O(n \log n)$.

3.4 Kod programu metodą sortowania

```
1 #include <iostream>
2 #include <vector>
3 #include <algorithm>
4 #include <utility>
5 using namespace std;
6
7 void wyswietlTablice(const vector<int>& A) {
8     for (int x : A) {
9         cout << x << " ";
10    }
11 }
```

Listing 7: Funkcja `wyswietlTablice`

Funkcja `wyswietlTablice` jest odpowiedzialna za wyświetlanie tablicy A .

```
1     vector<pair<int,int>> znajdzPary(const vector<int>& A, int k) {
2         int n = A.size();
3         int i = 0, j = i + 1;
4         vector<pair<int,int>> pLiczby;
5         while (i < n && j < n) {
6             if (i == j) {
7                 j++;
8             }
9
10            if (A[j] - A[i] == abs(k)) {
11
12                if (k > 0) {
13                    pair<int,int> para(A[i],A[j]);
14                    pLiczby.push_back(para);
15                } else {
16                    pair<int,int> para(A[j],A[i]);
17                    pLiczby.push_back(para);
18                }
19        }
20    }
21 }
```

```

19         int vi = A[i], vj = A[j];
20         while (i < n && A[i] == vi) i++;
21         while (j < n && A[j] == vj) j++;
22
23     }
24     else if (A[j] - A[i] < abs(k)) {
25         j++;
26     }
27     else {
28         i++;
29     }
30 }
31 return pLicz;
32 }
```

Listing 8: Funkcja znajdzPary

Funkcja znajdzPary przyjmuje jako argument vector liczb spośród których ma znaleźć pary o różnicy k.

```

1 int main() {
2     vector<int> A = {1, 5, 2, 2, 2, 5, 5, 4};
3     int k;
4     cout << "Tablica przed sortowaniem:\n";
5     cout << "[ ";
6     wyswietlTablice(A);
7     cout << " ] ";
8
9     sort(A.begin(), A.end());
10
11    cout << "\nPodaj k: ";
12    cin >> k;
13    cout << endl;
14
15    cout << "Tablica po posortowaniu:\n";
16    cout << "[ ";
17    wyswietlTablice(A);
18    cout << " ] " << endl << endl;
19
20
21    vector<pair<int, int>> pary=znajdzPary(A, k);
22
23    if(pary.size()==0){
24        cout << "Brak par o podanej roznicy k" << endl;
25        return 0;
26    }else{
27        cout << "Znaleziono pary:" << endl;
28        for(int i=0;i<pary.size();i++){
29            pair<int, int> para=pary.at(i);
30            cout << "[ " << para.first << ", " << para.second << " ] " ;
31        }
32    }
33 }
```

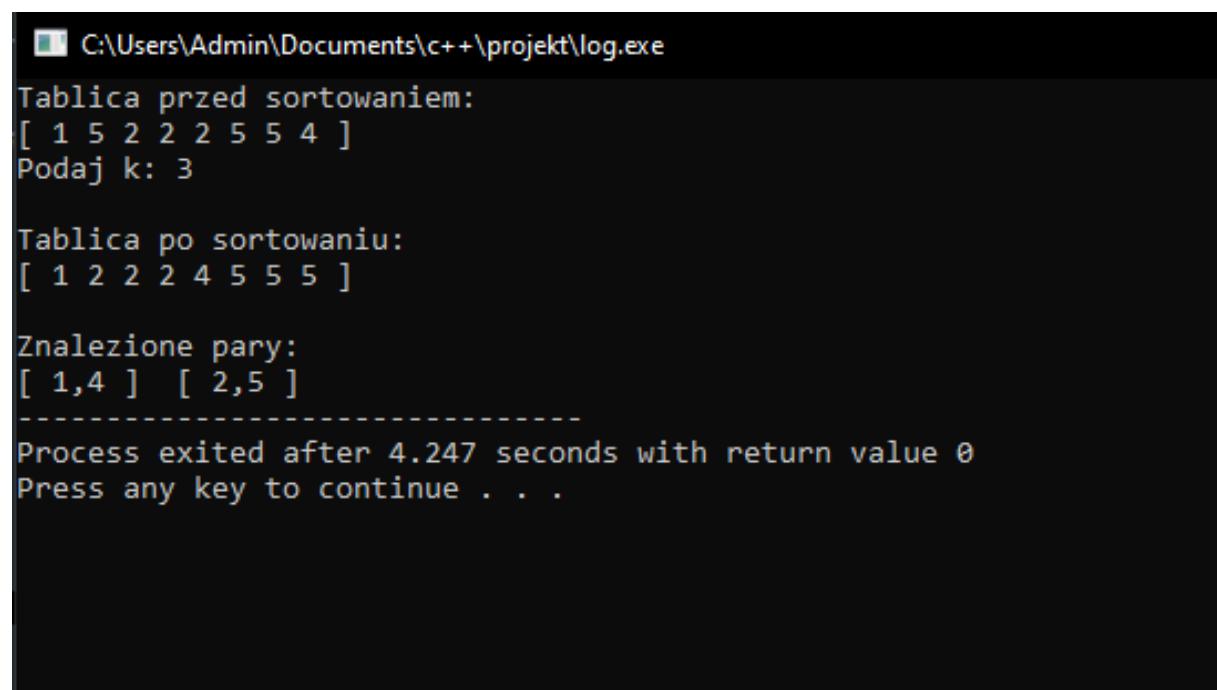
```
34     return 0;  
35 }
```

Listing 9: Funkcja main

Zainportowanie biblioteki `<algorithm>` umożliwia użycie funkcji sortującej `sort()`. Funkcja ta wykorzystuje algorytm typu Introsort, będący połączeniem sortowania szybkiego, sortowania przez kopcowanie oraz sortowania przez wstawianie, co zapewnia złożoność $O(n \log n)$ dla małych jak i dużych liczb.

Introsort to algorytm hybrydowy który w zależności od ilości sortowanych elementów przełącza się między algorytmami QuickSort, HeapSort i InsertionSort. QuickSort jest najszybszy lecz jego złożoność może spaść do $O(n^2)$, gdy spadek złożoności jest to możliwy, typ sortowania zostaje zmieniony na HeapSort który zawsze posiada złożoność $O(n \log n)$.

3.5 Wynik działania drugiego programu



```
C:\Users\Admin\Documents\c++\projekt\log.exe  
Tablica przed sortowaniem:  
[ 1 5 2 2 2 5 5 4 ]  
Podaj k: 3  
  
Tablica po sortowaniu:  
[ 1 2 2 2 4 5 5 5 ]  
  
Znalezione pary:  
[ 1,4 ] [ 2,5 ]  
-----  
Process exited after 4.247 seconds with return value 0  
Press any key to continue . . .
```

Rysunek 6: Przykładowe działanie dla dodatniej różnicy k

```
C:\Users\Admin\Documents\c++\projekt\log.exe
Tablica przed sortowaniem:
[ 1 5 2 2 2 5 5 4 ]
Podaj k: -2

Tablica po sortowaniu:
[ 1 2 2 2 4 5 5 5 ]

Znalezione pary:
[ 4,2 ]
-----
Process exited after 1.94 seconds with return value 0
Press any key to continue . . .
```

Rysunek 7: Przykładowe działanie dla ujemnej różnicy k

```
C:\Users\Admin\Documents\c++\projekt\log.exe
Tablica przed sortowaniem:
[ 1 5 2 2 2 5 5 4 ]
Podaj k: 8

Tablica po sortowaniu:
[ 1 2 2 2 4 5 5 5 ]

Brak par o podanej roznicy k
-----
Process exited after 3.468 seconds with return value 0
Press any key to continue . . .
```

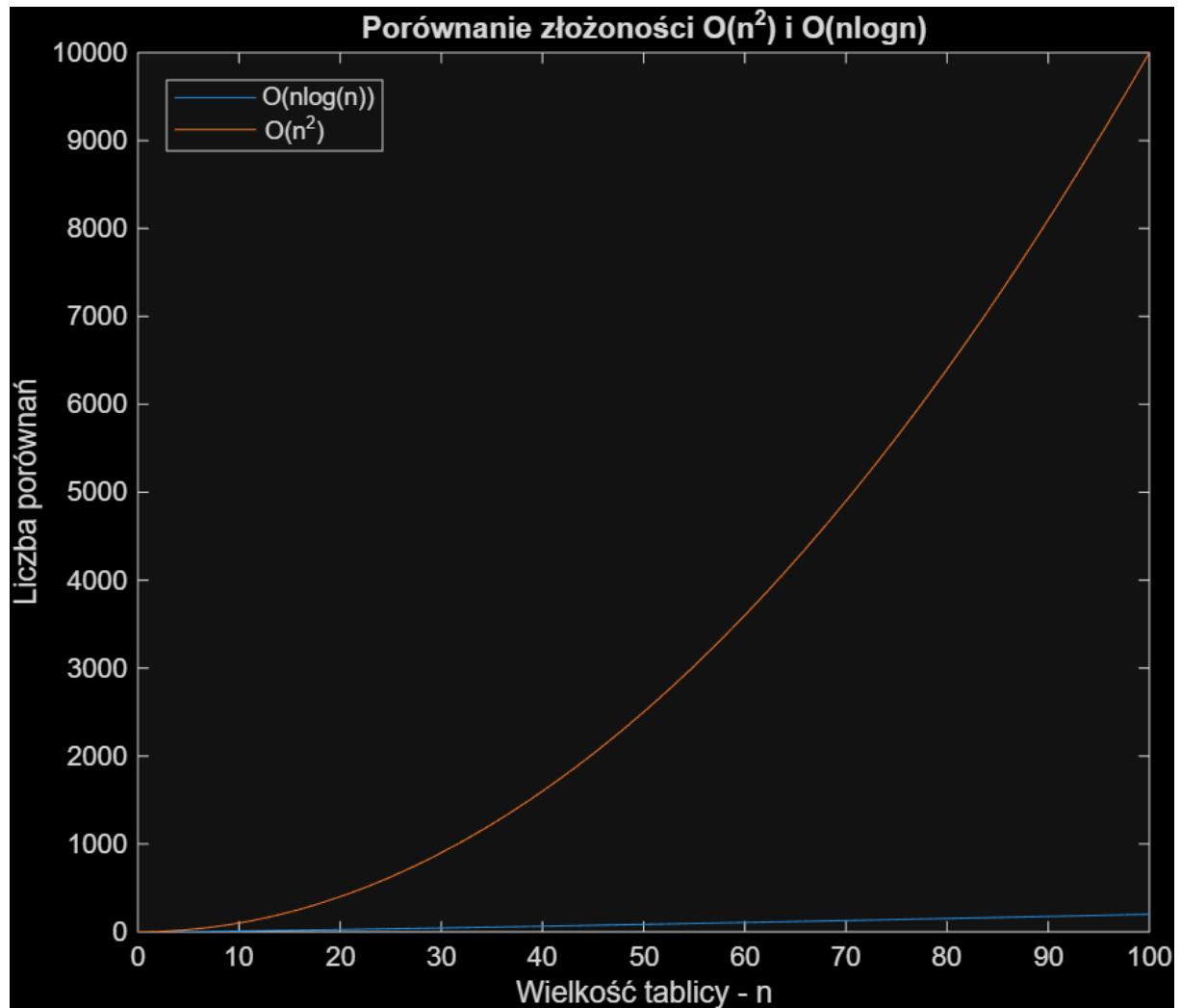
Rysunek 8: Przykładowe działanie dla braku par o roznicy k

4 Podsumowanie

4.1 Porównanie obu złożoności

Na poniższym wykresie można zauważyc jak wielką różnicę daje nam zoptymalizowanie początkowego pomysłu o złożoności $O(n^2)$ do $O(n \log n)$. Wraz z zwiększeniem wielkości

tablicy liczba porównań diametralnie wzrasta dla sposobu pierwszego który był początkowym zamysłem programu. Natomiast liczba porównań dla drugiego sposobu wzrasta w znacznie powolniejszym tempie.



Rysunek 9: Porównanie ilosci operacji

4.2 Testy wydajnościowe

Do policzenia czasu wykonywania obu programów użyłem funkcji wbudowanej w C++ biblioteki `<chrono>` zawierającej funkcje do operacji na czasie systemowym.

Ilość elementów	$O(n^2)$	$O(n \log n)$
10000	277.253 ms	2.001 ms
50000	11068.8 ms	10.011 ms
100000	44491.5 ms	20.019 ms

Tabela 1: Porównanie czasu wykonywania dla poszczególnych wielkości tablicy

4.3 Wnioski

Ostatecznie udało się zoptymalizować program do znacznie wydajniejszej wersji. Dokładna analiza problemu pozwoliła stworzyć program o mniejszym czasie wykonywania.