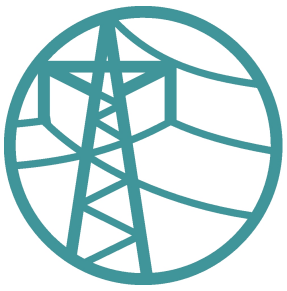


WYDZIAŁ
ELEKTROTECHNIKI
I INFORMATYKI
POLITECHNIKI RZESZOWSKIEJ

Jakub Maternia

Dla punktów płaszczyzny, o współrzędnych x i y zapisanych w tablicach, znaleźć ich najbliższych sąsiadów.

Projekt inżynierski

Opiekun pracy:
(dr inż. prof. PRz) Mariusz Borkowski

Rzeszów, 2025

Spis treści

1. Treść zadania

9. Dla punktów płaszczyzny, których współrzędne x i y są przechowywane w dwóch tablicach o długości n utwórz tablicę, która pod i-tym indeksem będzie przechowywać indeks najbliższego sąsiada i-tego punktu.

Przykład:

```
wejscie: 0, 1, -2, -1, 10
         0, 2, -3, -10, 9
wyjscie: 1, 0, 0, 2, 1

```

Listing 1: Przykład

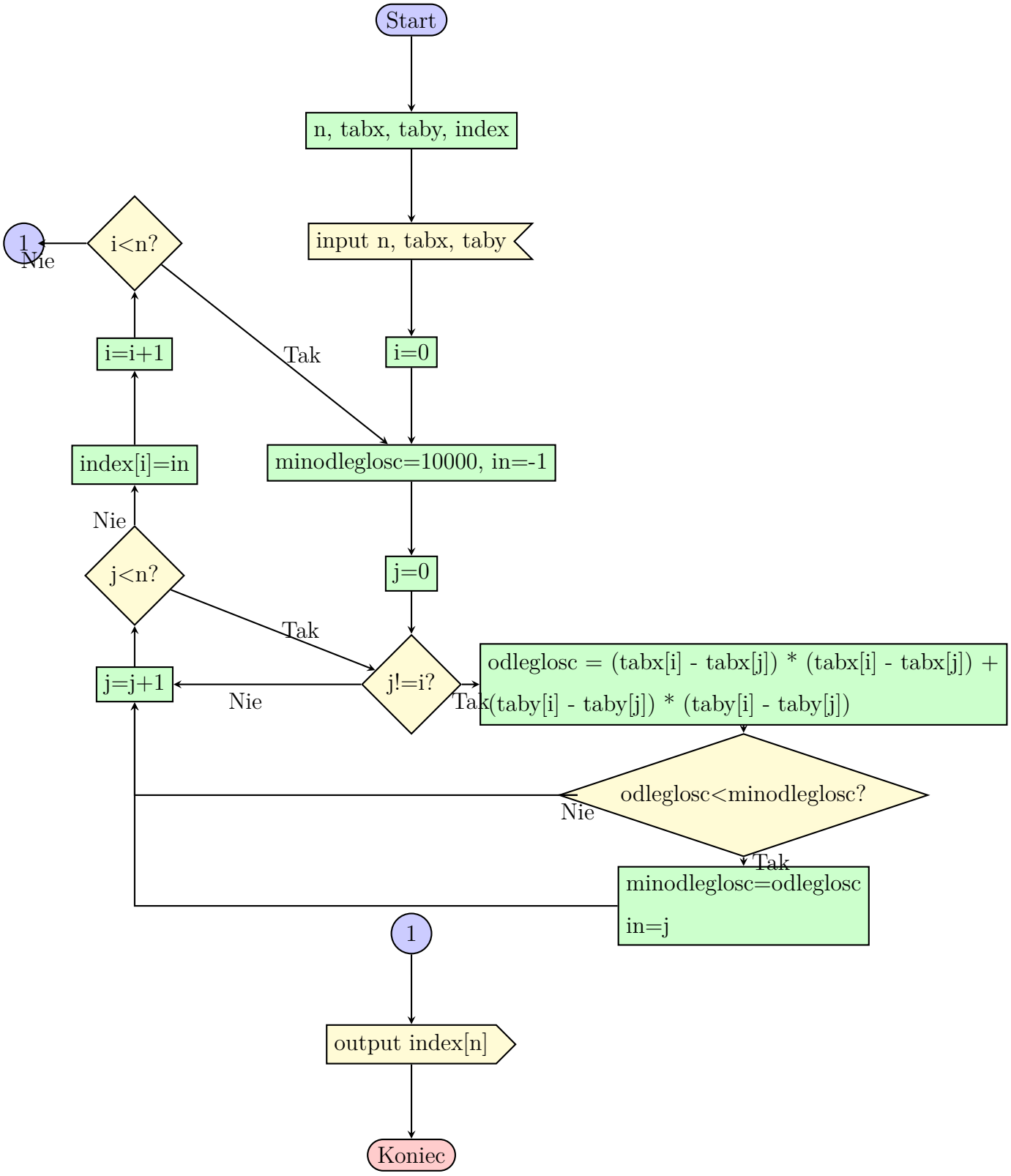
2. Podejście brute force

2.1. Analiza problemu i potencjalne rozwiązanie

Problem polega na obliczeniu odległości pomiędzy punktami oraz porównania jej z każdą inną wartością w poszukiwaniu najmniejszej dla danego punktu. Rozwiązanie polega na "wzięciu"jednego punktu oraz policzeniu jego odległości do każdego innego punku oraz znalezieniu najmniejszej i zapisaniu indeksu. Ta metoda rozwiąże ten problem niezależnie od sytuacji, sprawdzi wszystkie możliwe przypadki. W sytuacji kiedy wybrane zostaną dwa takie same punkty, program powinien wybrać inny od tego który został "wyciągnięty", stąd sprawdzanie czy indeksy są różne pomiędzy dwoma punktami. Zainicjowana tablica index i wypełniona wartościami -1, aby mieć pewność, że indeksy zapisują się odpowiednio zostanie wypełniona wartościami zmiennej in, w której będzie zapisywany indeks punktu o najmniejszej odległości. Na końcu zostanie zwrócona tablica i wypisana na ekran.

Danymi wejściowymi będą dwie tablice tabx i taby oraz długość tych tablic zapisana w zmiennej n, a danymi wyjściowymi będzie tablica index.

2.2. Schemat blokowy



2.3. Pseudokod

```
input: n, tabx, taby
output: index
index := -1
dla i od 0 do n-1 wykonaj
    minodleglosc := 10000
    in := -1
dla j od 0 do n-1 wykonaj
    jesli i != j wtedy
        odleglosc := (tabx[i] - tabx[j]) * (tabx[i] - tabx[j]) +
    (taby[i] - taby[j]) * (taby[i] - taby[j])
        jesli odleglosc < minodleglosc wtedy
            minodleglosc:= odleglosc
            in := j
        koniec jesli
    koniec jesli
koniec dla
index[i] := in
koniec dla
zwroc index
```

Listing 2: Pseudokod BruteForce

2.4. Przykładowe rozwiązanie

i	j	punkt[i]	punkt[j]	odleglosc	minodleglosc	in	index
0	1	0,0	1,2	2,23	2,23	1	1, -1, -1, -1
0	2	0,0	-2,-3	3,61	2,23	1	1, -1, -1, -1
0	3	0,0	-1,-10	10,04	2,23	1	1, -1, -1, -1
1	0	1,2	0,0	2,23	2,23	0	1, 0, -1, -1
1	2	1,2	-2,-3	5,83	2,23	0	1, 0, -1, -1
1	3	1,2	-1,-10	12,16	2,23	0	1, 0, -1, -1
2	0	-2,-3	0,0	3,06	3,06	0	1, 0, 0, -1
2	1	-2,-3	1,2	5,83	3,06	0	1, 0, 0, -1
2	3	-2,-3	-1,-10	7,07	3,06	0	1, 0, 0, -1
3	0	-1,-10	0,0	10,04	10,04	0	1, 0, 0, 0
3	1	-1,-10	1,2	12,16	10,04	0	1, 0, 0, 0
3	2	-1,-10	-2,-3	7,07	7,07	2	1, 0, 0, 2

Tabela 2.1: Przykładowe rozwiązanie BruteForce.

2.5. Złożoność obliczeniowa

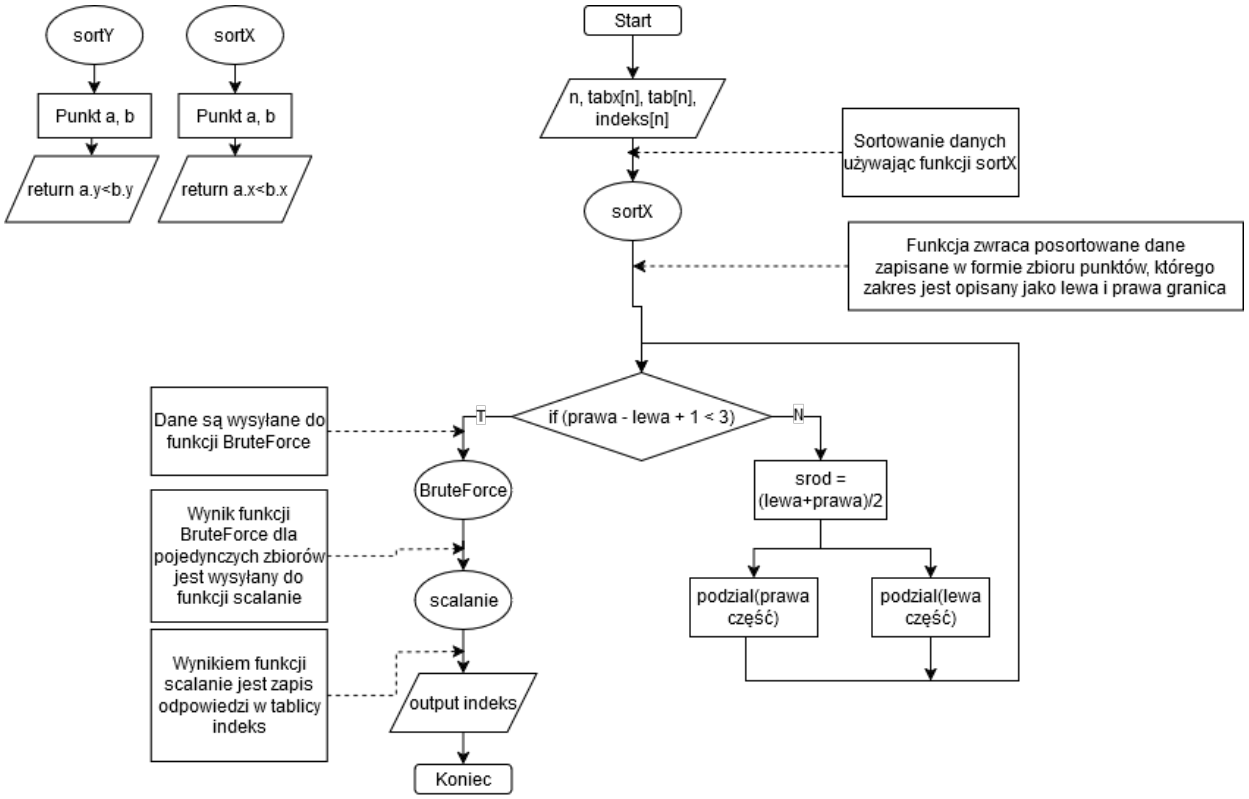
Złożoność obliczeniowa tego algorytmu wynosi $O(n^2)$, wynika to z faktu sprawdzania każdego punktu (oprócz tego samego). Program wykonuje porównanie i-tego elementu z j-tym elementem, gdzie jest ich n-1. Dokonujemy więc $n \cdot (n-1)$ porównań co ostatecznie daje nam złożonośc na poziomie: $n^2 - n - 1$.
Zatem złożoność czasowa algorytmu jest skategoryzowana jako $O(n^2)$.

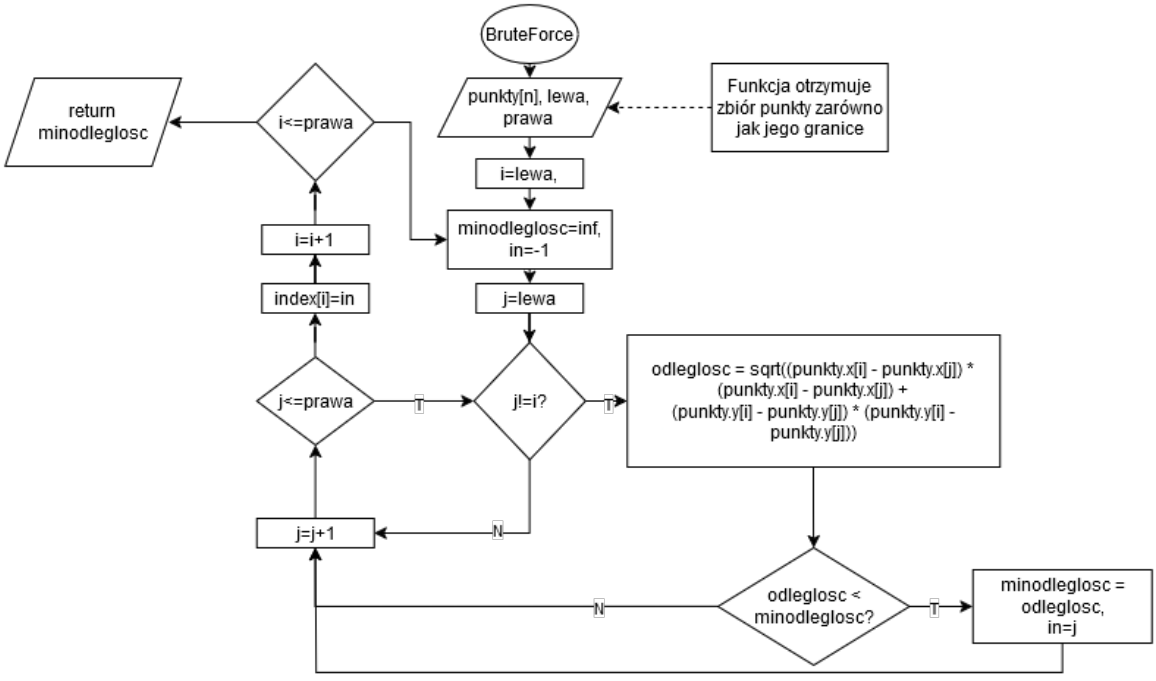
3. Druga metoda z wykorzystaniem metody dziel i zwyciężaj

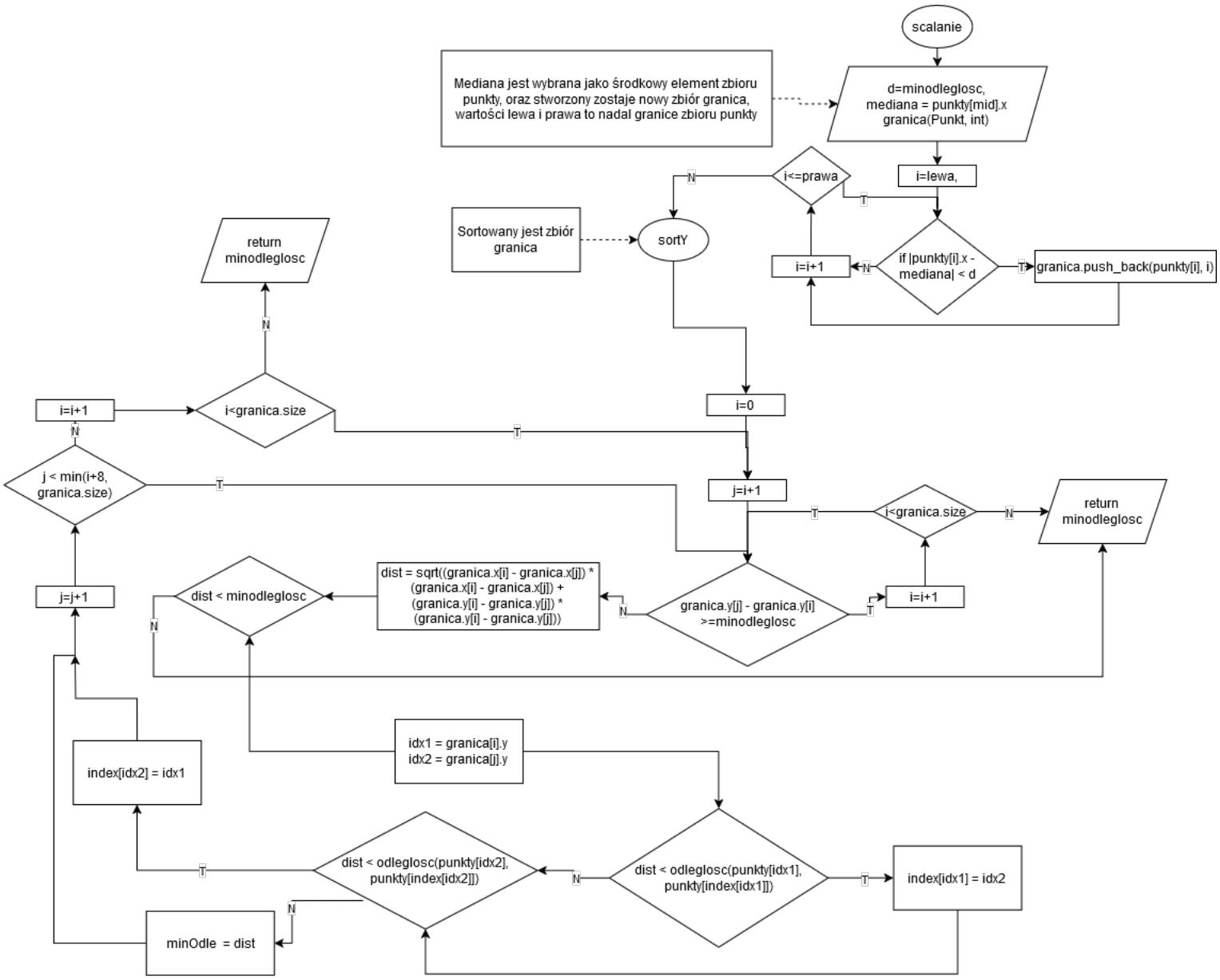
3.1. Metoda działania

Jeśli chcemy przyspieszyć działanie programu należy zająć się ilością dokonywanych operacji obliczania odległości. Dzieląc wejście do sytuacji kiedy mamy 2 lub 3 punkty możemy ograniczyć liczbę liczenia odległości i niskim kosztem stwierdzić indeks którego punktu jest najbliżej. Jednak zanim to będzie możliwe trzeba ułożyć dane w taki sposób żeby choć w małej części punkty leżały blisko siebie np. sortując po wartości współrzędnej x. Tym sposobem otrzymamy punkty, których indeksy najbliższych sąsiadów są obok siebie. Jeśli to nie nastąpi, podczas procesu scalania należy sprawdzić pas graniczny wokół granicy podziału o długości $2 \cdot d$. Wtedy porównamy graniczne punkty z obu stron granicy podziału tak aby upewnić się czy przypadkiem po drugiej stronie granicy nie leży punkt, który jest bliższy niż ten, który wcześniej został oznaczony jako najbliższy wewnątrz podzielonego zbioru. Dzięki właściwościom geometrii ograniczamy liczbę porównań do maksymalnie 7 punktów dla każdego punktu w pasie granicznym. Wynika to z faktu, że punkty w pasie są rozłożone w taki sposób, że mogą zajmować maksymalnie 7 sąsiednich komórek w siatce kwadratów o długości d/2. Żeby otrzymać wynik wystarczy będzie scalić spowrotem podzbiory oraz ustawić tablicę przechowującą odpowiedzi w odpowiedni sposób - tak aby wartości odzwierciedlały te, które zostały podane wcześniej.

3.2. Schemat blokowy







Rysunek 3.1: Schemat blokowy dla drugiej metody rozwiązania problemu.

3.3. Pseudokod

```
    input: punkty, n
    output: index
    jesli n <= 1 wtedy
        zwroc []
    koniec jesli

    posortuj punkty wedlug wspolrzednej x
    wykonaj podzial(punkty, 0, n-1, index)
    zwroc index

    funkcja podzial(punkty, lewa, prawa, index)
        jesli prawa - lewa + 1 <= 3 wtedy
            zwroc bruteForce(punkty, lewa, prawa, index)
        koniec jesli

        mid := (lewa + prawa) / 2
        granica_lewa := podzial(punkty, lewa, mid, index)
        granica_prawa := podzial(punkty, mid+1, prawa, index)
        d := min(granica_lewa, granica_prawa)

        scal(punkty, lewa, mid, prawa, d, index)
        zwroc d
    koniec funkcji

    funkcja scal(punkty, lewa, mid, prawa, d, index)
        granica := []
        dla i od lewa do prawa wykonaj
            jesli abs(punkty[i].x - punkty[mid].x) < d wtedy
                dodaj punkty[i] do granica
            koniec jesli
        koniec dla

        posortuj granica wedlug wspolrzednej y
        dla i od 0 do rozmiar(granica) - 1 wykonaj
            dla j od i+1 do min(i+8, rozmiar(granica)) wykonaj
                jesli granica[j].y - granica[i].y >= d wtedy
                    przerwij
                koniec jesli

                odleglosc := sqrt((granica[i].x - granica[j].x)
^2 + (granica[i].y - granica[j].y)^2)
                jesli odleglosc < d wtedy
                    zaktualizuj index dla granica[i] oraz
granica[j]
                    d := odleglosc
                koniec jesli
            koniec dla
        koniec dla
    koniec funkcji

    funkcja bruteForce(punkty, lewa, prawa, index)
        dla i od lewa do prawa wykonaj
            minodleglosc := nieskonczonosc
            dla j od lewa do prawa wykonaj
```

```
                jesli i != j wtedy
                    odleglosc := sqrt((punkty[i].x - punkty[j].x
)^2 + (punkty[i].y - punkty[j].y)^2)
                    jesli odleglosc < minodleglosc wtedy
                        minodleglosc := odleglosc
                        index[i] := j
                    koniec jesli
                koniec jesli
            koniec dla
        koniec dla
        zwroc minodleglosc
     koniec funkcji


```

Listing 3: Pseudokod Dziel i zwyciężaj

3.4. Przykładowe rozwiązanie

Przejdźmy przez kroki rozwiązania tego problemu na przykładzie, używając takich danych:

tabx = 1, 3, 5, 2 i taby = 2, 4, 1, 2.

Krok 1: Przygotowanie tablicy index = -1, -1, -1, -1

Krok 2: Przeniesienie danych do tablicy punkty i posortowanie rosnąco względem współrzędnej x.

punkty = [(1, 2), (2, 2), (3, 4), (5, 1)]

Krok 3: Rekurencyjne dzielenie tablicy punkty na mniejsze podzbiory. Lewa część: (1,2), (2,2). Prawa część: (3,4), (5,1).

Krok 4: Obliczenie odległości w lewej części metodą Bruteforce (nie dzielimy na mniejsze podzbiory ponieważ liczba punktów jest mniejsza niż 3).

Krok 5: Zapisanie odpowiedzi do tablicy indeks: indeks[0] = 1, indeks[1] = 0.

Krok 6: Analogicznie dla prawej części: indeks[2] = 3, indeks[3] = 2.

Krok 7: Scalanie, sprawdzenie czy w pasie granicznym wokół granicy podziału znajdują się punkty, które mogą mieć mniejszą odległość.

$2 \cdot d = 2$ (minimalna odległość pomiędzy punktami w obu częściach pomnożona x2).

Krok 8: Sprawdzenie odległości pomiędzy punktami, które znajdują się od siebie w takiej o Odległość pomiędzy punktami (2,2) i (3,4) wynosi 2,24. Jest większa niż $2 \cdot (d)$ więc wynik się nie zmienia.

Krok 9: Wypisanie tablicy indeks = [1, 0, 3, 2].

3.5. Złożoność obliczeniowa

Ponieważ w tym algorytmie obliczamy odległość metodą Brute-Force tylko dla dwóch lub trzech punktów, złożoność obliczeniowa tego procesu w najgorszym przypadku to $O(n)$, ponieważ jeden punkt jest najbliższym sąsiadem drugiego i odwrotnie. Sortowanie względem współrzędnej x jest dokonywane poprzez funkcję sort z biblioteki algorithm, którego złożoność obliczeniowa to $O(n \cdot (log(n)))$. Podobnie dla scalania, którego złożoność to również $O(n \cdot (log(n)))$.

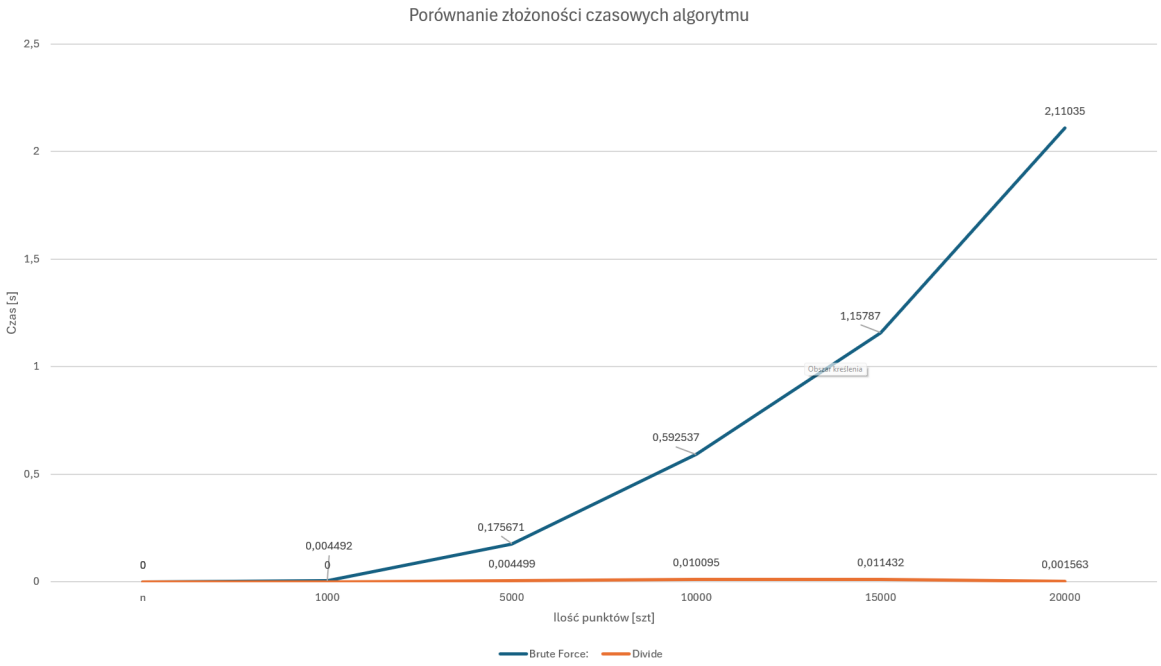
Zatem złożoność całego algorytmu to również $O(n \cdot (log(n)))$.

4. Implementacja obu algorytmów

4.1. Testy wydajności

n	1000	5000	10000	15000	20000
Brute	0.004492s	0.175671s	0.59253s7	1.15787s	2.11035s
Divide	0s	0.004499s	0.010095s	0.011432s	0.001563s

Tabela 4.2: Porównanie czasów dla algorytmów Brute Force i Divide dla różnych wartości n.



Rysunek 4.2: Wykres złożoności czasowej, porównujący dwie metody.

Algorytm wykorzystujący metodę dziel i zwyciężaj wykazał się niesamowitą wydajnością, nawet dla dużych rozmiarów danych, dokładnie tak jak w założeniu.

4.2. Kody programów

4.2.1. BruteForce

```
#include <iostream>
#include <cmath>
#include <ctime>
#include <chrono>
#include <fstream>
#include <cassert>

using namespace std;

int* szukanie(int n, int* tabx, int* taby){
    int *index = new int[n];    // Deklaracja tablicy index
    double odleglosc=0;

    for (int i =0; i<n;i++){   // Uzupelnienie tablicy index wartosciami -1 aby miec pewnosc ze nie beda mialy wplyw na wynik
        index[i]=-1;
    }
    for (int i = 0; i<n;i++){   // Obliczanie i zapisywanie indeksow najblizszych sasiadow
        double minodleglosc=numeric_limits<double>::infinity();
        int in=-1;
        for (int j = 0; j<n; j++){
            if (j!=i){
            odleglosc = (tabx[i] - tabx[j]) * (tabx[i] - tabx[j]) + (taby[i] - taby[j]) * (taby[i] - taby[j]);
            if (odleglosc < minodleglosc) {
                minodleglosc = odleglosc;
                in = j;
                }
            }
        }
        index[i]=in;
    }

    return index;
}

void testy() {   // Funkcja przeprowadzajaca testy przy uzyciu assert
    // Test 1: Prosty przypadek
    int tabx1[] = {0, 1, 2};
    int taby1[] = {0, 0, 0};
    int n1 = 3;
    int* wynik1 = szukanie(n1, tabx1, taby1);
    assert(wynik1[0] == 1);
    assert(wynik1[1] == 0 || wynik1[1] == 2);
    assert(wynik1[2] == 1);
    delete[] wynik1;

    // Test 2: Punkty w pionie
```

```
47          int tabx2[] = {0, 0, 0};
48          int taby2[] = {0, 1, 2};
49          int n2 = 3;
50          int* wynik2 = szukanie(n2, tabx2, taby2);
51          assert(wynik2[0] == 1);
52          assert(wynik2[1] == 0 || wynik2[1] == 2);
53          assert(wynik2[2] == 1);
54          delete[] wynik2;
55
56          // Test 3: Jeden punkt
57          int tabx3[] = {5};
58          int taby3[] = {5};
59          int n3 = 1;
60          int* wynik3 = szukanie(n3, tabx3, taby3);
61          assert(wynik3[0] == -1);
62          delete[] wynik3;
63
64          cout <<endl <<  "Test OK";
65      }
66
67
68      int main(){
69
70          ifstream file("danebrute.txt");   // Obsluga otwierania
    pliku
71          if (!file.is_open()) {
72              cout << "Nie udalo sie otworzyc pliku" << endl;
73              return 1;
74          }
75
76          int n;
77          file >> n;  // Pobieranie liczby punktow z pliku
78          //cout << "Wpisz ilosc punktow: " << endl;
79          //cin >> n;
80
81          if (n <= 0) {
82              cout << "Liczba punktow musi byc dodatnia!" << endl;
83              return 1;
84          }
85
86          int *tabx = new int[n];   // Deklaracja tablic
87          int *taby = new int[n];
88          int *index;    // Deklaracja wskaznika do tablicy index,
     ktora jest stworzona w funkcji
89
90          srand(time(NULL));
91
92          for(int i = 0; i<n; i++){   // Uzupelnienie tablic
93              //tabx[i] = rand() % 10;
94              //taby[i] = rand() % 10;
95              file >> tabx[i];
96          }
97          for(int i = 0; i<n; i++){
98              file >> taby[i];
99          }
100
```

```
101         for(int i = 0;i<n;i++){   // Wypisanie tablic na ekran
102             cout << tabx[i] << " ";
103         }
104 
105         cout << endl;
106 
107         for(int i = 0;i<n;i++){
108             cout << taby[i] << " ";
109         }
110 
111         cout << endl;
112         file.close();  // Zamkniecie pliku -> dane zostaly
    przepisane do tablic
113         std::chrono::high_resolution_clock::time_point t1 = std
    ::chrono::high_resolution_clock::now();
114         index = szukanie(n,tabx, taby); // Przypisanie wskaznika
     do tablicy "zwrotnej" z funkcji szukanie
115         std::chrono::high_resolution_clock::time_point t2 = std
    ::chrono::high_resolution_clock::now();
116 
117         std::chrono::duration<double> time_span = std::chrono::
    duration_cast<std::chrono::duration<double>>(t2 - t1);
118 
119         for (int i=0; i<n;i++){  // Wypisanie wyniku na ekran
120             cout << index[i] << " ";
121         }
122 
123         testy();  // Wywolanie funkcji w ktorej wykonywane sa
    testy
124 
125         cout << endl << "czas: " << time_span.count() << endl;
    // Wypisanie pomiaru czasu
126 
127         delete[] tabx;   // Usuniecie tablic z pamieci
128         delete[] taby;
129 
130         return 0;
131     }
```

Listing 4: Kod BruteForce

4.2.2. DivideAndConquer

```
1     #include <iostream>
2     #include <cmath>
3     #include <ctime>
4     #include <chrono>
5     #include <algorithm>
6     #include <vector>
7     #include <fstream>
8     #include <cassert>
9 
10    using namespace std;
11 
12    class Punkt {   // Klasa za pomoca ktorej zapisywane beda
    punkty przy zachowaniu obu wspolrzednych i oryginalnego
```

```
    indeksu
13      public:
14          double x;
15          double y;
16          double indeks;
17  };
18
19  double odleglosc(const Punkt& p1, const Punkt& p2) {   //
    Funkcja obliczajaca odleglosc tradycyjna metoda -> lepsza
    modularnosc kodu
20      return sqrt((p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y
    ) * (p1.y - p2.y));
21  }
22
23  void sortX(Punkt* punkty, int lewa, int prawa) {  // Funkcja
     sortujaca po wspolrzednych X uzywajac polecenia sort z
    biblioteki algorithm
24      sort(punkty + lewa, punkty + prawa + 1, [](const Punkt&
    a, const Punkt& b) {
25          return a.x < b.x;
26      });
27  }
28
29
30  double bruteForce(Punkt* punkty, int lewa, int prawa, int*
    index) {   // Funkcja obliczajaca indeksy najblizszych
    sasiadow
31      double minOdle = numeric_limits<double>::infinity();  //
     Ustawienie zmiennej na wartosc inf
32      for (int i = lewa; i <= prawa; i++) {
33          double aktminOdle = numeric_limits<double>::infinity
    ();
34          int in = -1;
35
36          for (int j = lewa; j <= prawa; j++) {  // ustalanie
    indeksu najblizszego sasiada
37              if (i != j) {
38                  double odleglosc1 = odleglosc(punkty[i],
    punkty[j]);
39                  if (odleglosc1 < aktminOdle) {
40                      aktminOdle = odleglosc1;
41                      in = j;
42                  }
43              }
44          }
45
46          index[i] = in;
47          minOdle = min(minOdle, aktminOdle);
48      }
49      return minOdle;
50  }
51
52  double scalanie(Punkt* punkty, int lewa, int mid, int prawa,
     int* index, double d) {
53      vector<pair<Punkt, int> > granica;   // Stworzenie
    zbioru granica
```

```
54          double mediana = punkty[mid].x;
55
56
57          for (int i = lewa; i <= prawa; i++) {    // "Wlozenie"
    do zbioru granica punktow ktore sie kwalifikuja na
    potencjalnie blizsze po drugiej stronie tej granicy
58              if (abs(punkty[i].x - mediana) < d) {
59                  granica.push_back({punkty[i], i});
60              }
61          }
62
63          sort(granica.begin(), granica.end(),    // Sortowanie po
     wspolrzednych Y zbioru granica
64               [](const pair<Punkt, int>& a, const pair<Punkt, int
    >& b) {
65                   return a.first.y < b.first.y;
66               });
67
68          double minOdle = d;
69
70
71          for (int i = 0; i < granica.size(); i++) {  // Ustalanie
     czy w zbiorze granica sa punkty blizsze niz ustalone
    wczesniej
72              for (int j = i + 1; j < min(i + 8, (int)granica.size
    ()); j++) {   // Skorzystanie z dowodu geometrycznego na to,
    ze nie moze byc wiecej niz 7 punktow ktore sa potencjalnie
    blizsze
73                  if (granica[j].first.y - granica[i].first.y >=
    minOdle) break;
74
75                  double dist = odleglosc(granica[i].first,
    granica[j].first);
76                  if (dist < minOdle) {
77                      int idx1 = granica[i].second;
78                      int idx2 = granica[j].second;
79
80                      if (dist < odleglosc(punkty[idx1], punkty[
    index[idx1]])) {  // Podmienianie indeksow w tablicy
    wynikowej index jesli zachodzi taka potrzeba
81                          index[idx1] = idx2;
82                      }
83                      if (dist < odleglosc(punkty[idx2], punkty[
    index[idx2]])) {
84                          index[idx2] = idx1;
85                      }
86
87                      minOdle = dist;
88                  }
89              }
90          }
91
92          return minOdle;
93      }
94
```

```
95      double podzial(Punkt* punkty, int lewa, int prawa, int*
    index) {  // Funkcja odpowiedzialna za podzial zbioru danych
    na mniejsze
96          if (prawa - lewa + 1 <= 3) {
97              return bruteForce(punkty, lewa, prawa, index);   //
    Podzial dokonuje sie az osiagniemy zbiory o ilosci elementow
    3 i mniej
98          }
99
100         int srod = lewa + (prawa - lewa) / 2;
101
102         double granlew = podzial(punkty, lewa, srod, index);
    // Ustalenie granic
103         double granpraw = podzial(punkty, srod + 1, prawa, index
    );
104
105         return scalanie(punkty, lewa, srod, prawa, index, min(
    granlew, granpraw));   // Wywowalanie funkcji scalanie ktora
    zapewni ostateczny wynik
106     }
107
108     void sasiedzi(Punkt* punkty, int n, int* index) {
109         sortX(punkty, 0, n - 1);
110         podzial(punkty, 0, n - 1, index);
111     }
112
113     void testy() {
114
115         // Test 1: Prosty przypadek
116         Punkt punkty[3] = {{0, 0, 0}, {1, 1, 1}, {2, 2, 2}};
117         int index[3] = {-1, -1, -1};
118         sasiedzi(punkty, 3, index);
119         assert(index[0] == 1);
120         assert(index[1] == 0 || index[1] == 2);
121         assert(index[2] == 1);
122
123         // Test 2: Punkty w linii poziomej
124         Punkt punkty1[4] = {{0, 0, 0}, {1, 0, 1}, {2, 0, 2}, {3,
    0, 3}};
125         int index1[4] = {-1, -1, -1, -1};
126         sasiedzi(punkty1, 4, index1);
127         assert(index1[0] == 1);
128         assert(index1[1] == 0 || index1[1] == 2);
129         assert(index1[2] == 1 || index1[2] == 3);
130         assert(index1[3] == 2);
131
132         // Test 3: Jeden punkt
133         Punkt punkty2[1] = {{0, 0, 0}};
134         int index2[1] = {-1};
135         sasiedzi(punkty2, 1, index2);
136         assert(index2[0] == -1);
137
138         cout << endl << "Test OK" << endl;
139     }
140
141     int main() {
```

```
142         ifstream file("danedivide.txt");   // Obsluga otwierania
    pliku
143         if (!file.is_open()) {
144             cout << "Nie udalo sie otworzyc pliku" << endl;
145             return 1;
146         }
147 
148         int n;
149         file >> n;
150         //cout << "Wpisz ilosc punktow: " << endl;
151         //cin >> n;
152 
153         if (n <= 0) {
154             cout << "Liczba punktow musi byc dodatnia!" << endl;
155             return 1;
156         }
157 
158         int *tabx = new int[n];
159         int *taby = new int[n];
160 
161         srand(time(NULL));
162 
163        /*for(int i = 0; i < n; i++) {
164                 //cin >> tabx[i];
165                 //cin >> taby[i];
166             tabx[i] = rand() % 10;
167             taby[i] = rand() % 10;
168         }   */
169 
170         for(int i = 0; i<n; i++){   // Uzupelnienie tablic
171             file >> tabx[i];
172         }
173         for(int i = 0; i<n; i++){
174             file >> taby[i];
175         }
176 
177         for(int i = 0;i<n;i++){
178             cout << tabx[i] << " ";
179         }
180 
181         cout << endl;
182 
183         for(int i = 0; i < n; i++){
184             cout << taby[i] << " ";
185         }
186         cout << endl;
187 
188         Punkt* punkty = new Punkt[n];
189         for (int i = 0; i < n; i++) {
190             punkty[i].x = tabx[i];
191             punkty[i].y = taby[i];
192             punkty[i].indeks = i;
193         }
194 
195         file.close();  // Zamkniecie pliku -> dane zostaly
    przepisane do tablic
```

```
196         int* index = new int[n];
197 
198         auto t1 = chrono::high_resolution_clock::now();
199         sasiedzi(punkty, n, index);
200 
201         auto t2 = chrono::high_resolution_clock::now();
202 
203         int* ostindeks = new int[n];    // Nowa tablica
    ostindeks w ktorej bedzie przechowywany ostateczny wynik - w
    tablicy index wynik jest odpowiedni dla posortowanych punktow
     wzgledem x
204         for(int i = 0; i < n; i++) {
205             int ind = punkty[i].indeks;   // Powrot do
    oryginalnej kolejnosci osiagany jest za pomoca zapisanego
    wczesniej indeksu
206             ostindeks[ind] = punkty[index[i]].indeks;
207         }
208 
209 
210         for(int i = 0; i < n ; i++){   // Wypisanie wyniku na
    ekran
211             cout << ostindeks[i] << " ";
212         }
213 
214         testy();    // Wywolanie funkcji testujacej
215 
216         chrono::duration<double> time_span = chrono::
    duration_cast<chrono::duration<double>>(t2 - t1);
217         cout << endl << "czas: " << time_span.count() << endl;
218 
219         delete[] index;
220         delete[] punkty;
221         delete[] tabx;
222         delete[] taby;
223 
224         return 0;
225     }
```

Listing 5: Kod Dziel i Zwyciężaj