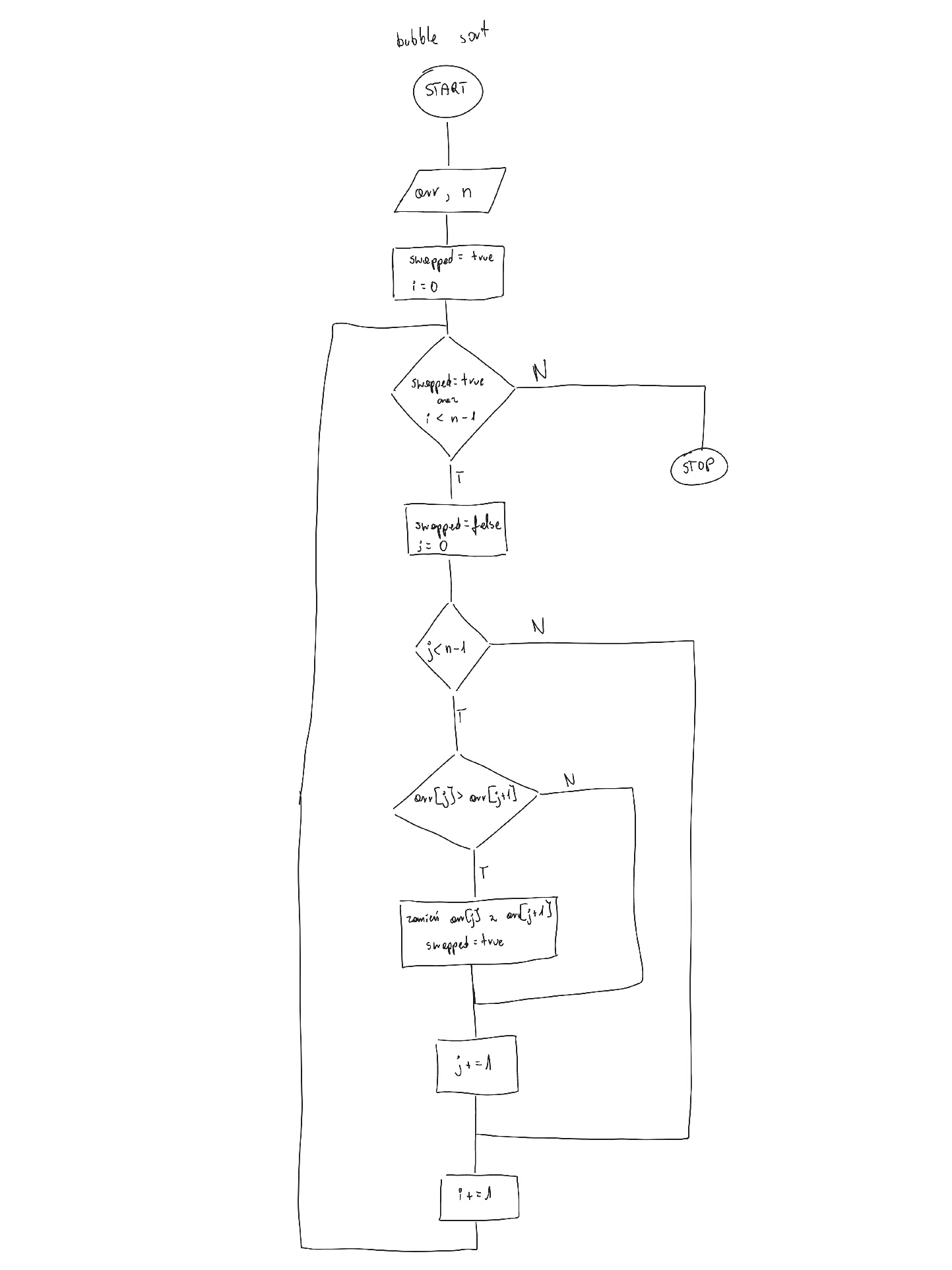
**1. Bubble Sort**

Bubble Sort to jeden z najprostszych algorytmów sortowania. Działa poprzez wielokrotne przechodzenie przez tablicę, porównywanie sąsiednich elementów i zamienianie ich miejscami, jeśli są w złej kolejności. W każdej iteracji ostatni element każdego przejścia (który jest największy z niesortowanych) osiada na odpowiednim miejscu na końcu tablicy, zmniejszając zakres dalszych przeszukiwań. Proces ten jest powtarzany, aż do momentu, gdy w danym przejściu nie wykonano żadnej zamiany, co oznacza, że tablica jest posortowana.

* swapped – monitoruje czy w danej iteracji pętli doszło do zamiany
* arr- tablica do posortowania
* n - liczba elementów w sortowanej tablicy

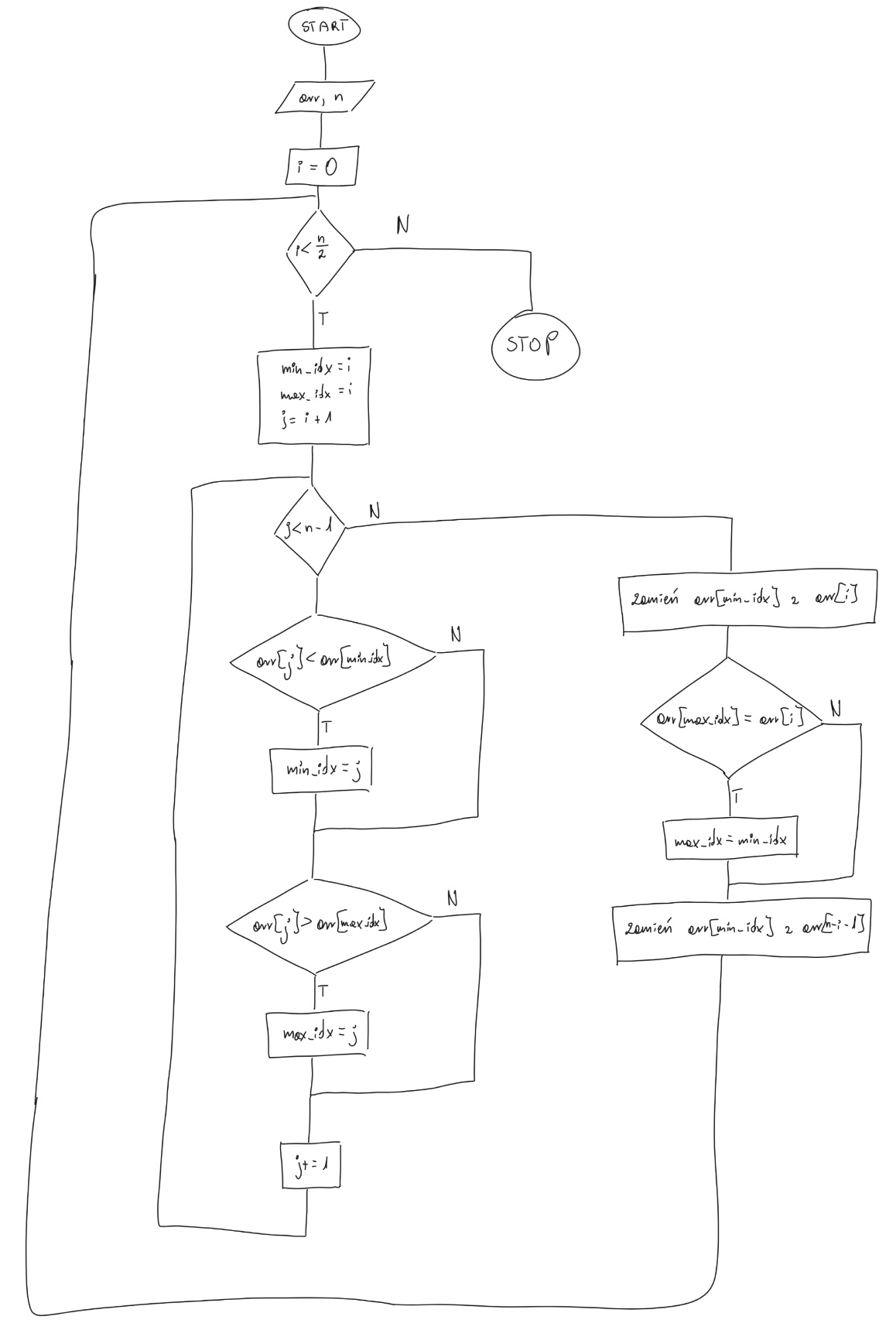


**2. Selection Sort**

Selection Sort w każdej iteracji pętli zewnętrznej jednocześnie wyszukuje najmniejszy i największy element w niesortowanej części tablicy. Po znalezieniu indeksów tych elementów, najmniejszy element jest zamieniany miejscem z elementem na aktualnie rozpatrywanej początkowej pozycji (i), a największy element jest zamieniany z elementem na końcu rozpatrywanego segmentu (n - i - 1).

Jest o tyle lepszy od zwykłego selection sorta że „dzieli tablicę” na dwie części: posortowaną i nieposortowaną, ale robi to z obu końców tablicy jednocześnie.

* arr- tablica do posortowania
* n - liczba elementów w sortowanej tablicy
* min\_idx - indeks najmniejszego elementu znalezionego w przetwarzanym segmencie tablicy
* max\_idx - indeks największego elementu w przetwarzanym segmencie tablicy



**3. Insertion Sort z wyszukiwaniem binarnym**

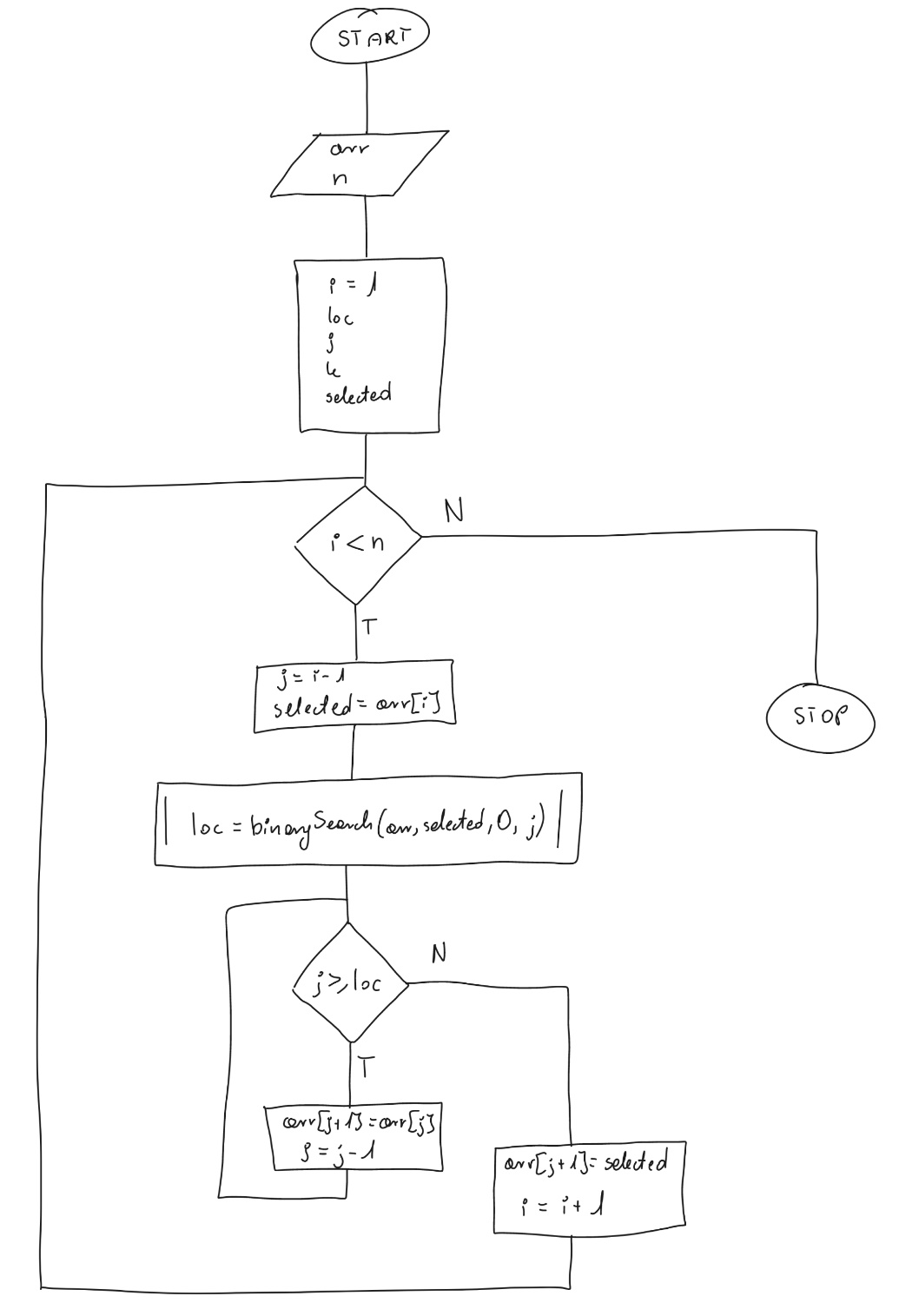
Insertion Sort można usprawnić na dwa popularne sposoby: poprzez wartownika i wyszukiwanie binarne.

Wyszukiwanie binarne jest stosowane do znalezienia pozycji na której powinien znaleźć się bieżący element. W każdej iteracji algorytm bierze element z niesortowanej części tablicy, a następnie używa funkcji ‘binarySearch’ do znalezienia odpowiedniego miejsca w już posortowanej części tablicy. Wyszukiwanie binarne pozwala na znaczną redukcję liczby porównań potrzebnych do znalezienia odpowiedniego miejsca dla każdego elementu.

W moim przypadku algorytm z wyszukiwaniem binarnym działał lekko lepiej dla danych losowych i posortowanych odwrotnie. Natomiast algorytm z wartownikiem okazał się zdecydowanie lepszy dla danych już posortowanych.

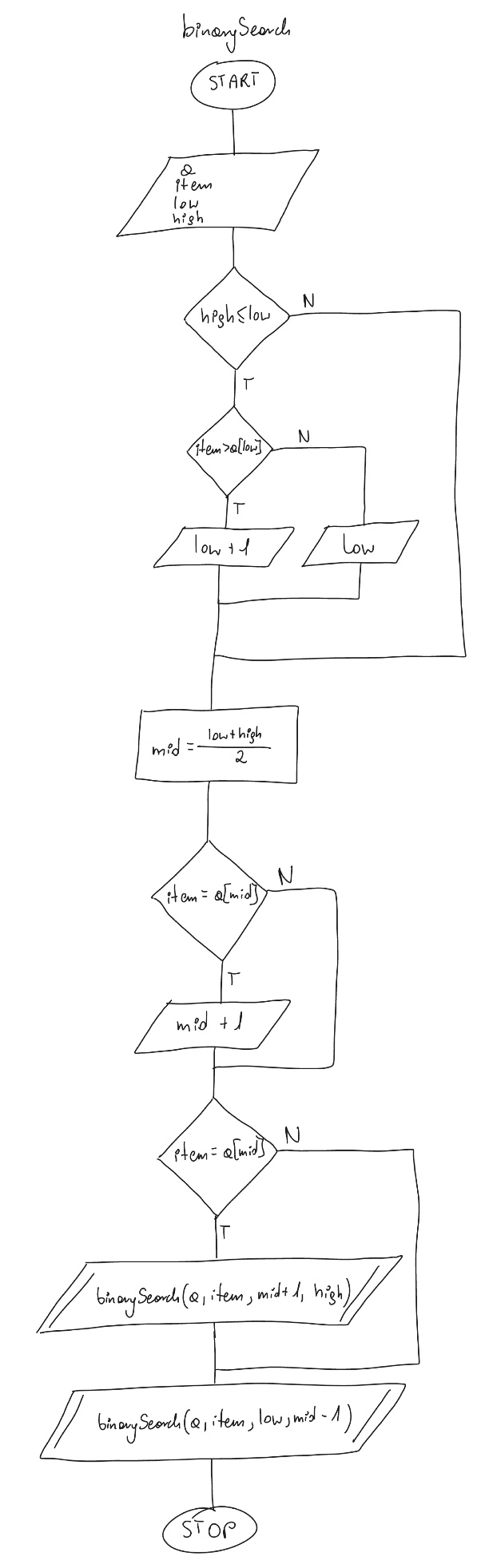
**Funkcja insertionSortBinary:**

* arr[] – Tablica do posortowania
* n - Liczba elementów w tablicy arr[]
* loc - Index gdzie wybrany element powinien zostać wstawiony
* selected - Aktualnie przetwarzany element z tablicy ‘arr[]’, który jest przenoszony do odpowiedniego miejsca w posortowanej części tablicy.



**Funkcja binarySearch:**

* a[] - Tablica, w której jest przeprowadzane wyszukiwanie binarne
* item - Element, którego położenie jest szukane w tablicy
* low, high - Indeksy określające zakres tablicy, w którym jest przeprowadzane wyszukiwanie, ‘low’ to dolna granica zakresu, a ‘high’ to górna granica
* mid - Środkowy indeks aktualnego zakresu wyszukiwania

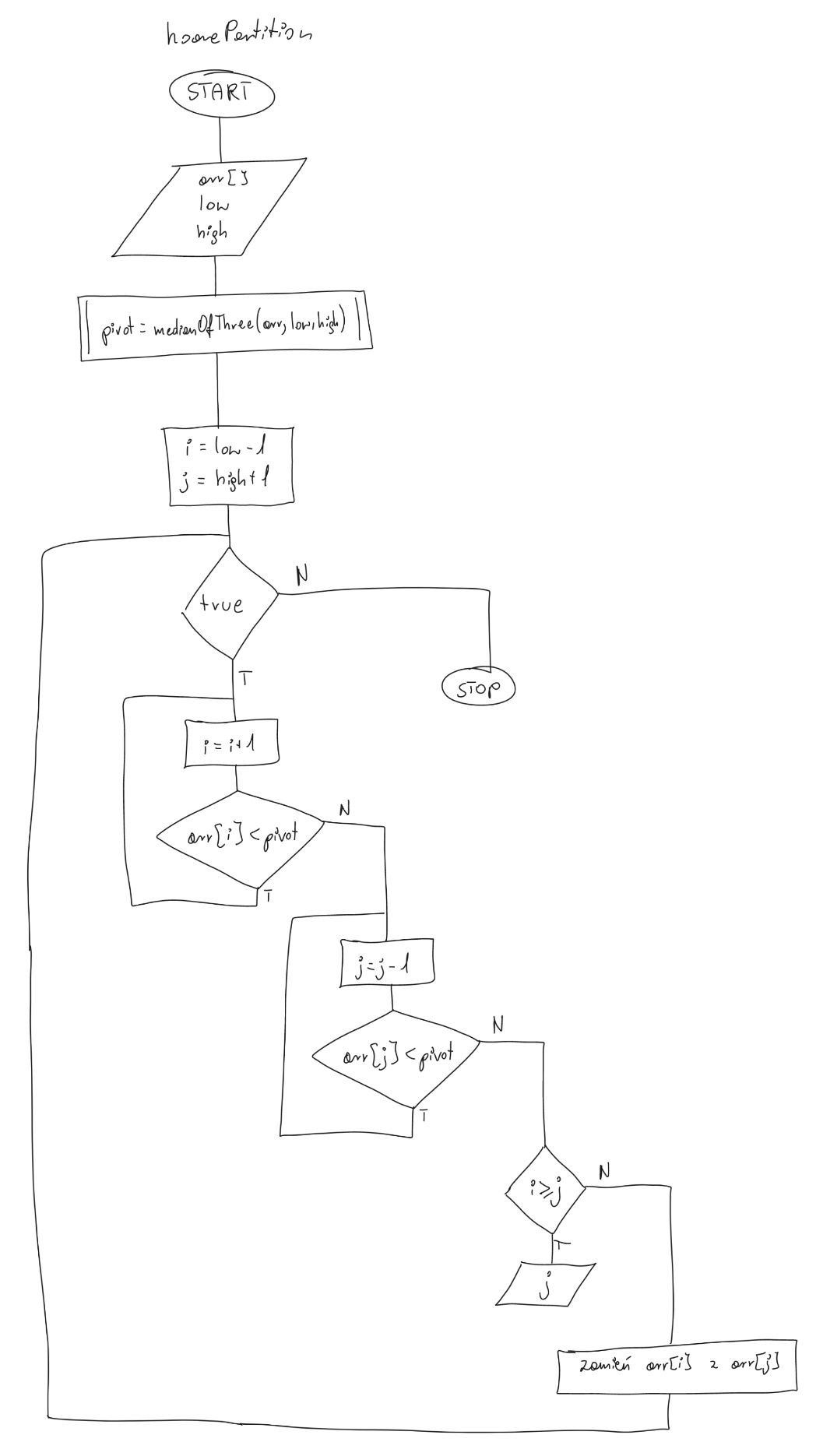


**4. Quicksort**

Quicksort to szybki algorytm oparty na technice „dziel i zwyciężaj”. Używa on mediany spośród pierwszego, środkowego i ostatniego elementu danego podsegmentu tablicy jako pivota, a następnie przegrupowuje elementy tablicy tak, aby wszystkie elementy mniejsze od pivota znalazły się przed nim, a wszystkie większe za nim. Algorytm kontynuuje proces sortowania rekurencyjnie dla obu podsegmentów tworzonych w wyniku partycjonowania.

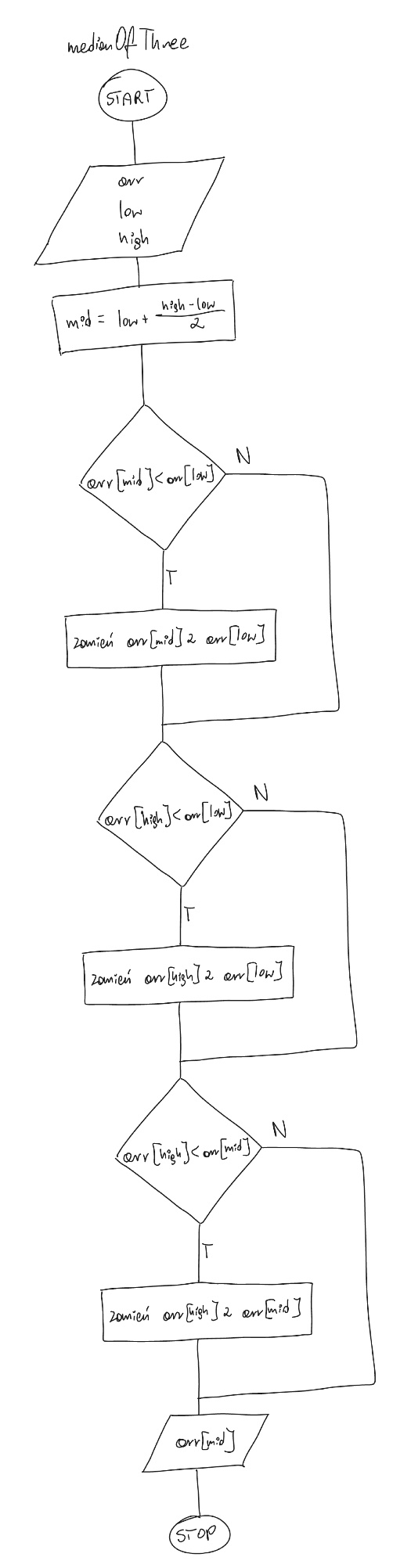
**Funkcja hoarePartition:**

* pivot - wartość, względem której elementy będą partycjonowane

****

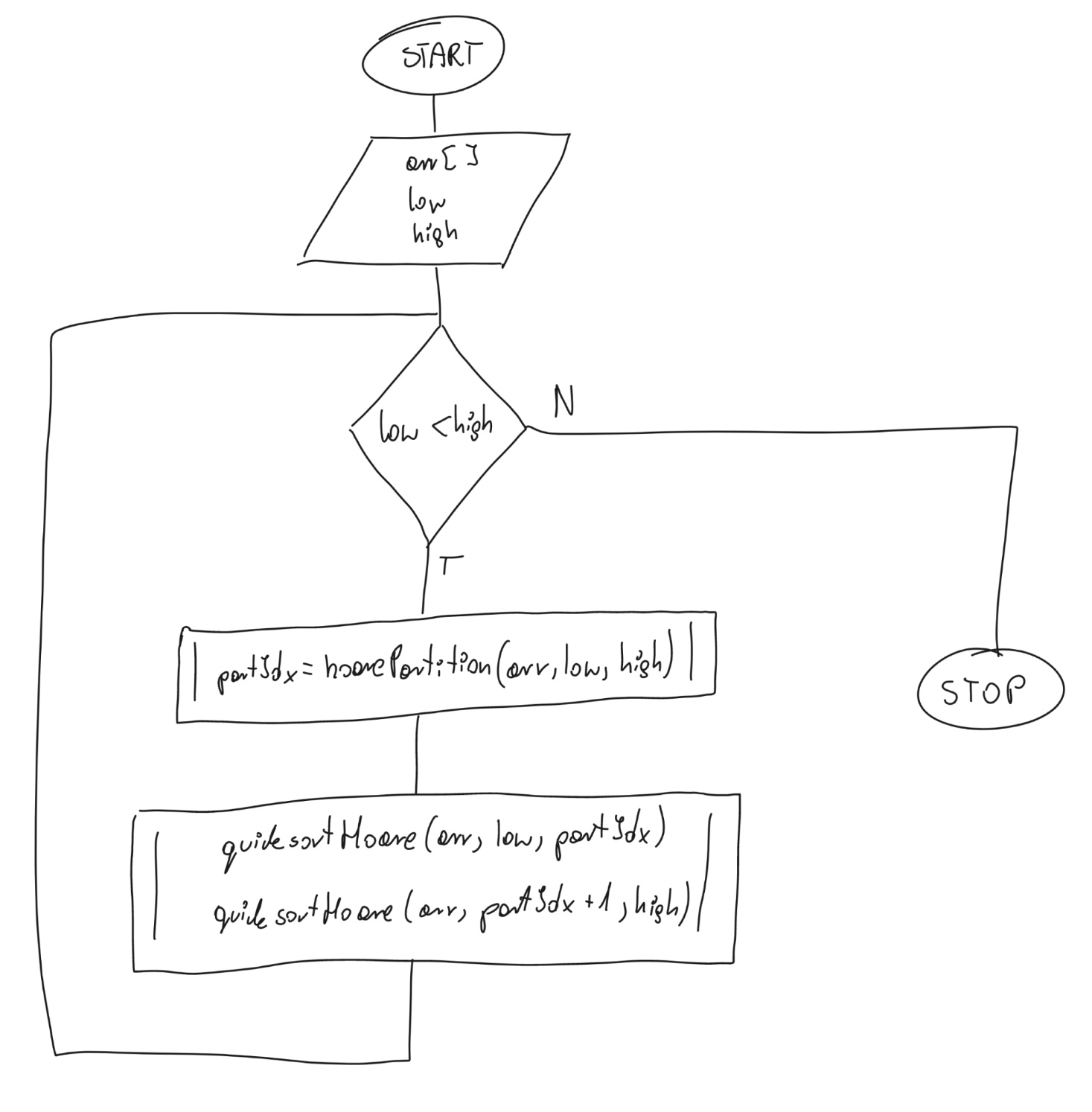
**Funkcja medianOfThree:**

* arr- tablica do posortowania
* low, high - Indeksy określające zakres tablicy, w którym jest przeprowadzane wyszukiwanie, ‘low’ to dolna granica zakresu, a ‘high’ to górna granica
* mid - Środkowy element między ‘low’ i ‘high’



**Funkcja quicksortHoare:**

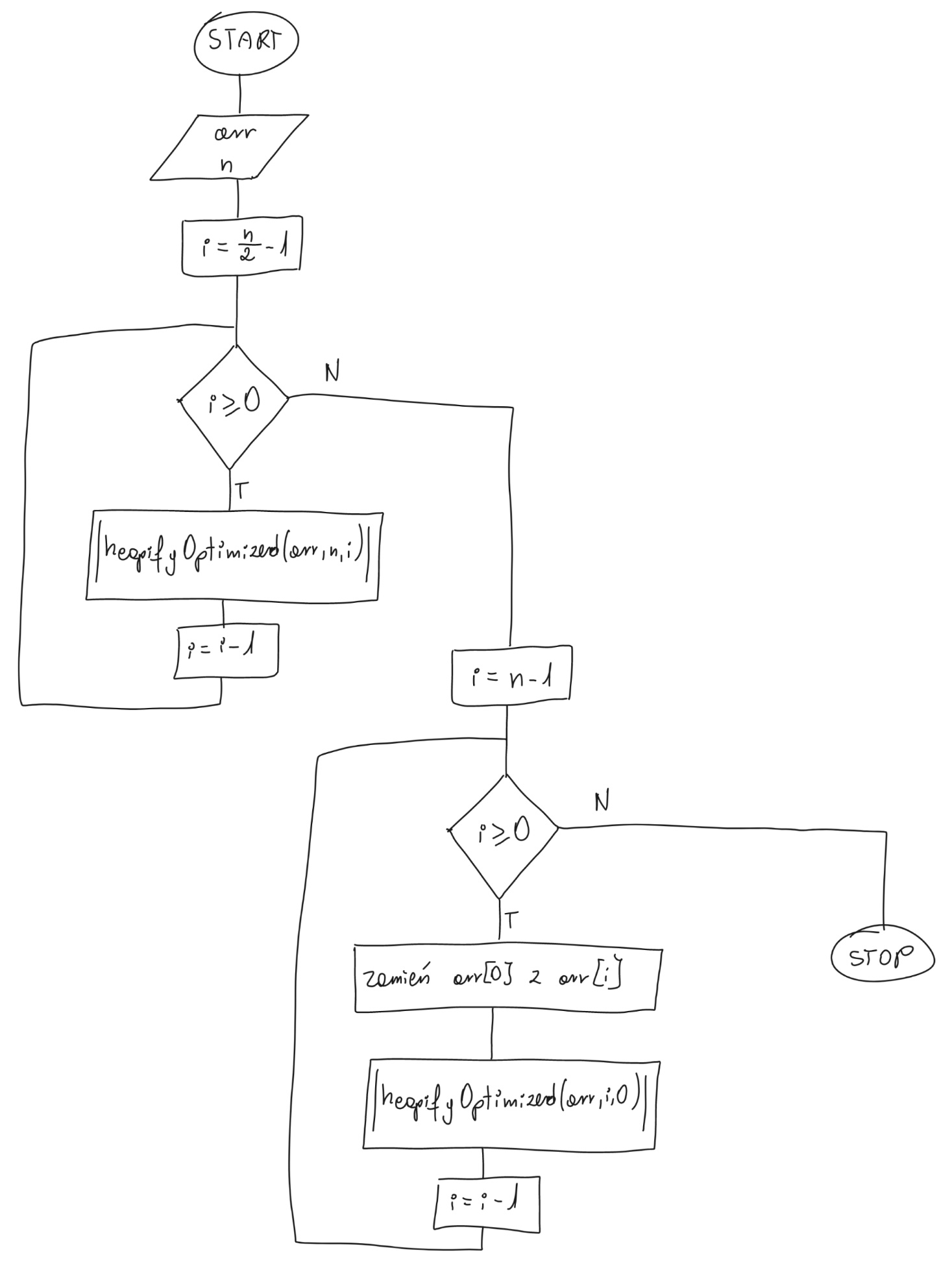
* partIdx - Indeks, gdzie partycja zostaje podzielona przez funkcję ‘hoarePartition’
* low, high - Indeksy określające zakres tablicy, w którym jest przeprowadzane wyszukiwanie, ‘low’ to dolna granica zakresu, a ‘high’ to górna granica

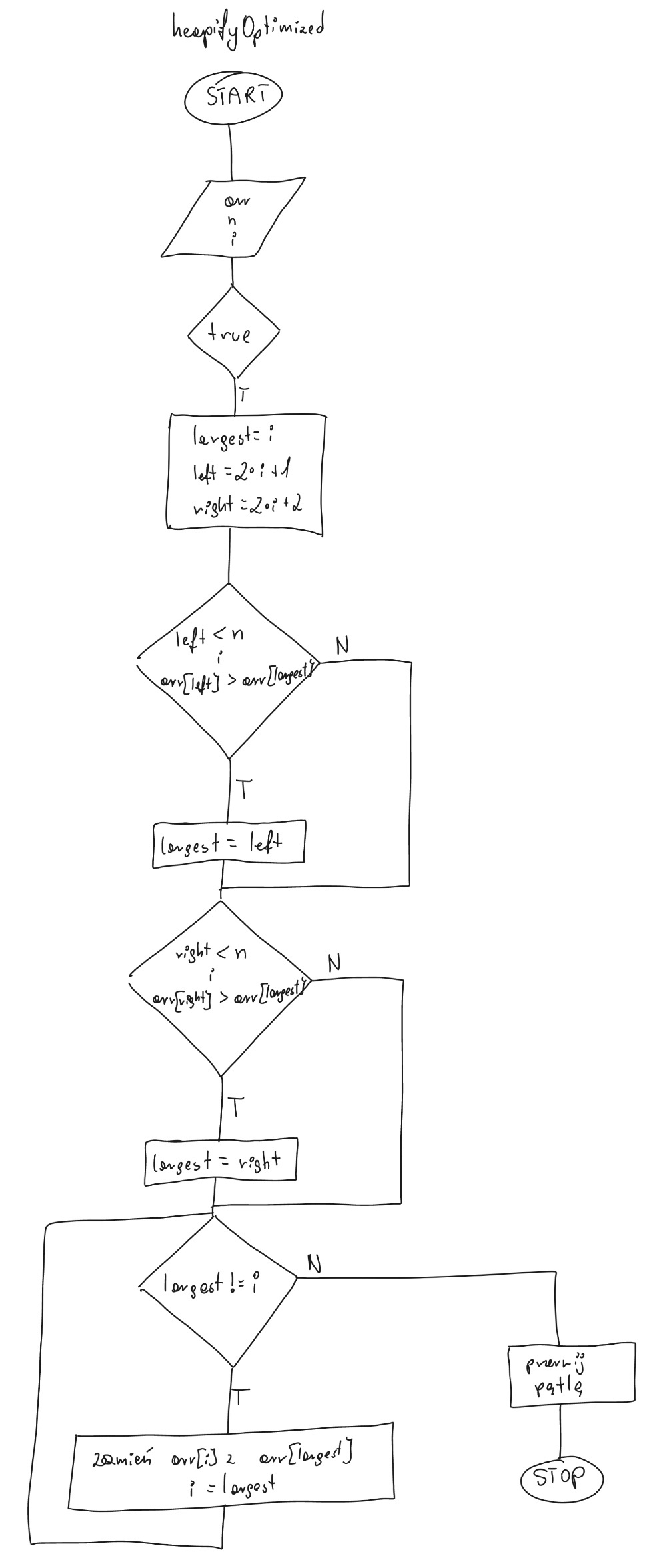


**5. Heapsort**

Heapsort to algorytm sortujący wykorzystujący strukturę danych zwaną kopcem, aby sortować elementy. Algorytm rozpoczyna od zbudowania kopca maksymalnego, a następnie przenosi największy element (korzeń kopca) na koniec nieposortowanej sekcji tablicy. Po przeniesieniu, przywraca strukturę kopca maksymalnego i powtarza proces, aż cała tablica zostanie posortowana.

* arr- tablica do posortowania
* n - liczba elementów w sortowanej tablicy
* i - Indeks obecnego węzła (korzenia) kopca, który jest sprawdzany, czy utrzymuje właściwości kopca
* largest - Indeks największego elementu spośród bieżącego węzła
* left - Indeks lewego dziecka węzła
* right - Indeks prawego dziecka węzła



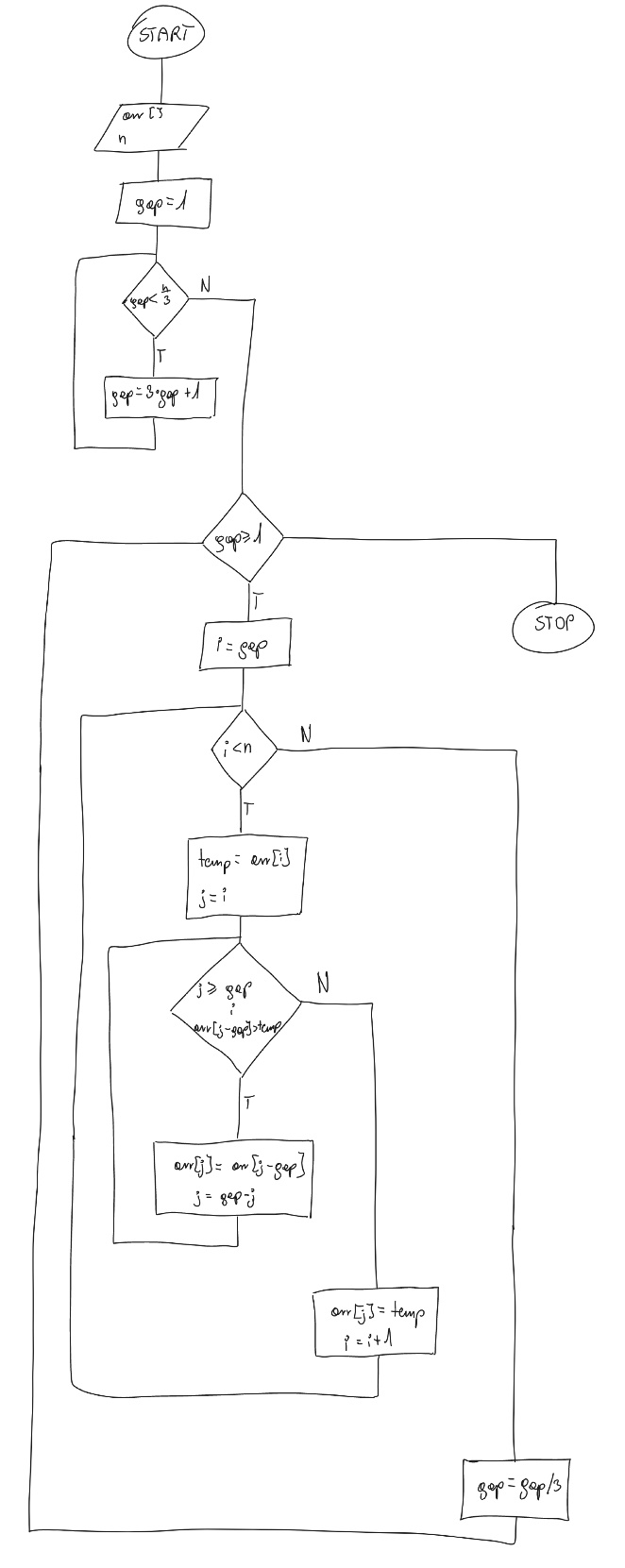


**6. Shell Sort (z wykorzystaniem sekwencji Knutha)**

Shell Sort jest ulepszoną wersją Insertion Sort, która pozwala na zamianę elementów oddalonych o dużą liczbę pozycji. Algorytm używa sekwencji odstępów (w tym przypadku sekwencji Knutha, czyli 3\*gap+1), aby stopniowo zmniejszać odstępy między porównywanymi elementami, aż nie przekroczy jednej trzeciej liczby elementów do sortowania.

Następnie, dla każdego odstępu, wykonuje sortowanie przez wstawianie, gdzie elementy są porównywane i zamieniane w odstępach określonych przez aktualną wartość ‘gap’. Proces ten powtarza się, aż ‘gap’ osiągnie wartość

* arr- tablica do posortowania
* n - liczba elementów w sortowanej tablicy
* gap - Zmienna określająca odstęp między porównywanymi elementami
* temp - Tymczasowa zmienna do przechowywania wartości elementu ‘arr[i]’, który jest aktualnie wstawiany w posortowaną sekwencję w ramach danej wartości ‘gap’



**Wykresy**

