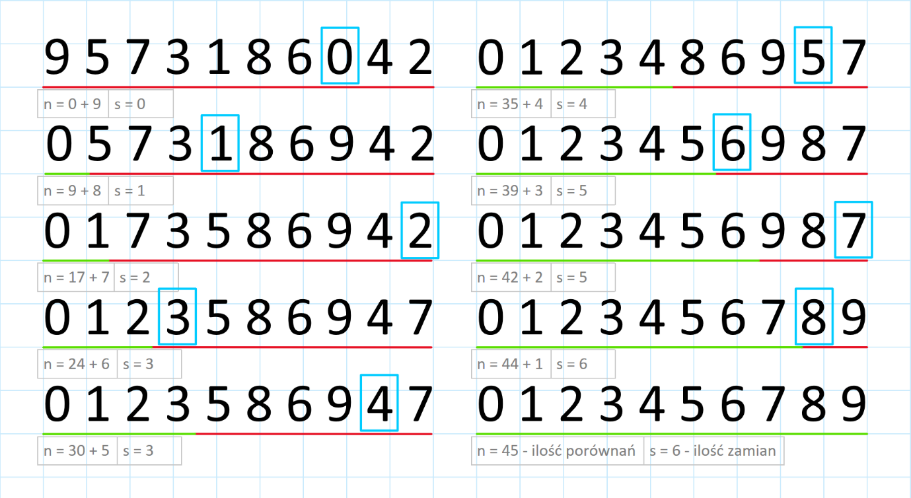
# Wstęp i opis algorytmów

W tym rozdziale opisuję główne celi projektu oraz wykorzystane algorytmy dla ich implementacji.

Głównym zadaniem projektu było porównanie działania dwóch algorytmów sortowania, a mianowicie: sortowania przez wybieranie oraz sortowania kopcowego. Drugorzędnym zadaniem było napisanie algorytmów odczytu danych z plików i generowania danych testowych o różnej złożoności sortowania dla każdego algorytmu.

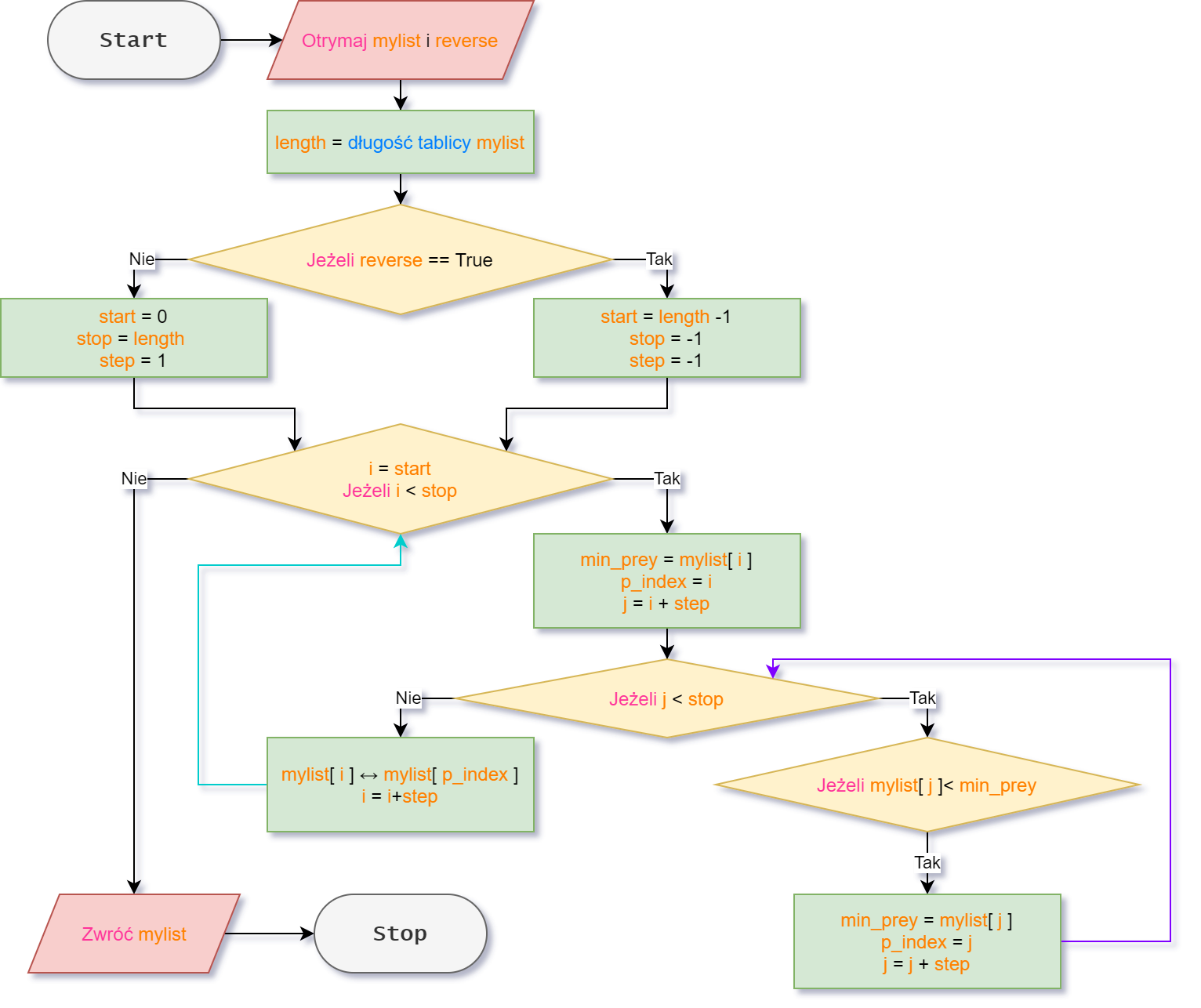
## Sortowanie przez wybieranie

Sortowanie przez wybieranie to niestabilny porównujący algorytm sortowania na miejscu. Ma złożoność czasową , co oznacza, że jest nieefektywnym w przypadku dużych list. Taki algorytm charakteryzuje się prostotą i ma przewagę wydajnościową nad bardziej złożonymi algorytmami w pewnych sytuacjach, zwłaszcza gdy pamięć pomocnicza jest ograniczona.

Algorytm dzieli listę wejściową na dwie partycji: posortowaną i nieposortowaną. Początkowo posortowana część jest pusta, a nieposortowana partycja zajmuje całą listę wejściową. Algorytm jest wykonywany poprzez znalezienie najmniejszego elementu w niesortowanej części, zamianę go na pierwszy nieposortowany element i przesunięcie podziału listy. Działanie algorytmu krok po kroku pokazano na rysunku.

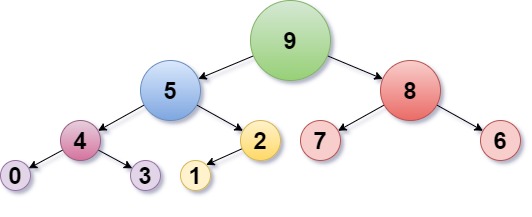
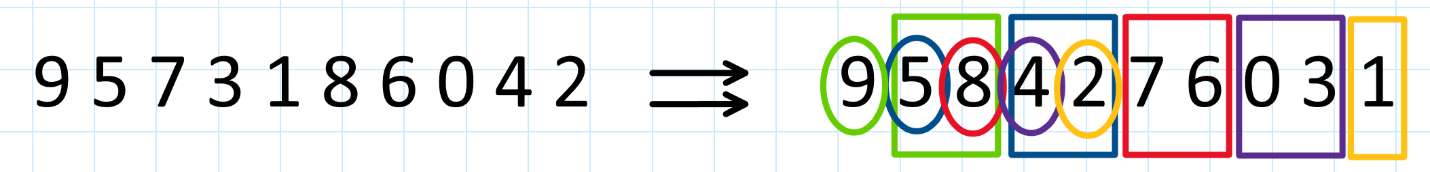
Pseudokod algorytmu sortowania przez wybieranie:

1. Otrzymaj = mylist i reverse
2. length = długośc tablicy mylist
3. Jeżeli reverse == True wykonuj K04:K06
4. | start = 0
5. | stop = length
6. | step = 1
7. Inaczej wykonuj K08:K10
8. | start = length - 1
9. | stop = -1
10. | step = -1
11. Dla i = start, start + step, start + 2\*step, start + 3\*step, … , stop: wykonuj K12:K18
12. | min\_prey = mylist[ i ]
13. | p\_index = i
14. | Dla j = i + step, i + 2\*step, i + 3\*step, … , stop: wykonuj K15:K17
15. | | Jeżeli mylist [ i ] < min\_prey wykonuj K16:K17
16. | | | min\_prey = mylist[ j ]
17. | | | p\_index = j
18. | mylist[ i ] ↔ mylist[ p\_index ]
19. Zwróć substring
20. Zakończ

Schemat blokowy tego algorytmu podano na rysunku.

Największą zaletą takiego algorytmu jest minimalna możliwa liczba zamian elementów (w najgorszym przypadku ). Mimo to efektywność czasowa sortowania według wyboru jest kwadratowa, więc istnieje wiele algorytmów, które mają mniejszą złożoność czasową.

## Sortowanie przez kopcowanie

Jednym z takich algorytmów jest sortowanie przez kopcowanie. Podobnie do algorytmu sortowania przez wybieranie, kopcowy algorytm sortuje na miejscu metodą porównań, nie jest stabilnym i analogicznie dzieli dane wejściowe na posortowaną i nieposortowaną partycji. Sortowanie kopcowe wybiera największy element z części niesortowanej i wstawiając go do posortowanej części, iteracyjnie zmniejsza nieposortowaną partycję. W przeciwieństwie do sortowania przez wybieranie, sortowanie kopcowe nie skanuje liniowo cały nieposortowany obszar. Ten algorytm obsługuje nieposortowaną część w postaci kopca (rysunek), aby szybko znaleźć największy element, który zawsze będzie znajdować się na górze. Ale dla roboty takiego algorytmu dane wejściowe muszą być ułożone w postaci kopca. Takie początkowe przygotowanie danych wejściowych wykonuje algorytm Heapify, sprawdzając czy potomki danego elementu są mniejsze od niego. Przykład kopca z rysunku w postaci tablicy podano na rysunku.

Sortowanie kopcowe ma pesymistyczną złożoność czasową , co jest nawet lepsze niż szybkie sortowanie (), chociaż w praktyce działa wolniej (oczekiwana złożoność obu algorytmów jest taka sama i wynosi ).

Sortowanie kopcowe zostało wynalezione przez J. Williamsa w 1964 roku.