Algorytmy i struktury danych

dr inż. Sławomir Samolej
D108 A, tel: 865 1486,
email: ssamolej@prz-rzeszow.pl
WWW: ssamolej.prz-rzeszow.pl

Program komputerowy - dyskusja

- Program komputerowy jest skonkretyzowaniem abstrakcyjnych **algorytmów** przetwarzających określone **struktury danych**,
- Stąd, pojęcia struktury danych i algorytmu są bardzo ściśle związane ze sobą związane – dobór struktury danych narzuca wybór algorytmu i odwrotnie,
- Dane odzwierciedlają zwykle informacje o obiektach rzeczywistych przystosowane ostatecznie do możliwości komputera.

Struktury danych - dyskusja

- Podstawowe **typy danych**, jakie może przechować komputer to liczby stało i zmiennopozycyjne,
- Dane są przechowywane w tzw. **zmiennych** – miejscach w pamięci, które można zapisać ciągiem bitów lub odczytać.
- Większość języków programowania umożliwia w prosty sposób zorganizowanie podstawowych typów danych w **podstawowe struktury danych**:
  - Tablicę,
  - Rekord,
  - Zbiór,
  - Plik sekwencyjny,
- W strukturach podstawowych zmienne zmieniają jedynie wartość, z góry określona jest długość danych – ilość i rodzaj elementów w strukturze.
- Na bazie struktur podstawowych buduje się **struktury złożone** – w nich możliwa jest zmiana zarówno wartości zmiennych jak i przeorganizowanie samej struktury danych w czasie wykonywania programu,
- Typowe struktury złożone to:
  - Lista,
  - Stos,
  - Kolejka
  - Drzewo.

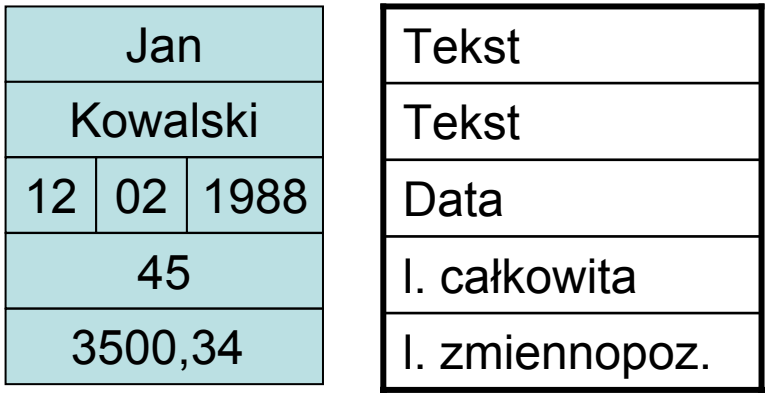
Podstawowe struktury danych - tablica

- Tablica jest strukturą jednorodną – składa się ze składowych tego samego typu.
- Tablica nazywana jest strukturą o dostępie swobodnym – wszystkie składowe mogą być wybrane w dowolnej kolejności i są jednakowo dostępne,
- W celu wybrania pojedynczej składowej stosuje się indeks – numer składowej w tablicy,
- Zastosowania tablic – przechowywanie grupy danych jednakowego typu – wyników pomiarów, tekstów, macierzy, wektorów itp.

x_1	0.5
x_2	0.25
x_3	0.125
x_4	0.0625
x_5	0.03125

Podstawowe struktury danych - rekord

- Służy do łączenia w jedną strukturę elementów o dowolnych typach, np.:
  - Liczby zespolone – para liczb rzeczywistych,
  - Wiersz z bazy danych – imię, nazwisko, data urodzenia, wiek, płaca podstawowa.



- Uwagi:
  - Składowymi rekordu może być tablica (np. tablica znaków zawierająca imię),
  - Rekordy mogą być zgrupowane w tablicę (np. prosta baza danych).

Podstawowe struktury danych - zbiór

- Pojęcie zbioru danych odpowiada matematycznemu pojęciu zbioru,
- Wprowadza się je, aby można było dokonywać operacji na danych identycznych jak operacji na zbiorach matematycznych,
- Kluczowymi operacjami są: zdefiniowanie zbioru, sprawdzenie, czy dana wartość należy do zbioru, możliwość zdefiniowania różnicy, sumy zbiorów oraz zdefiniowania podzbiorów.

Podstawowe struktury danych – plik sekwencyjny

- Plik sekwencyjny jest **ciągiem danych**, w pewnych przypadkach może być traktowanym jako nieskończony, lub wystarczająco długi,
- Specyfiką pliku sekwencyjnego jest, że w danym momencie **dostępna jest tylko jedna składowa** ciągu określona przez **aktualną pozycję,**
- W przetwarzaniu pliku dostępne są następujące **operatory plikowe**:
  - Ustawienie się na początek pliku,
  - Przesunięcie aktualnej pozycji o zadaną ilość elementów ciągu,
  - Odczyt bieżącego elementu ciągu,
  - Dopisanie elementu na końcu ciągu,
  - Dostępny jest również operator: *Zapis w bieżącym elemencie ciągu* (uwaga: następuje zamazanie poprzednich elementów ciągu).

Algorytm - definicja

Algorytm:

- Skończony ciąg/sekwencja reguł, które aplikuje się na skończonej liczbie danych, pozwalający rozwiązać zbliżone do siebie klasy problemów;
- Zespół reguł charakterystycznych dla pewnych obliczeń lub czynności informatycznych.

Słowo "algorytm" pochodzi od nazwiska Muhammed ibn Musa Alchwarizmi (أبو عبد الله محمد بن موسى الخوارزمي) matematyka perskiego z IX wieku i początkowo oznaczało w Europie sposób obliczeń oparty na dziesiętnym systemie liczbowym.

Jeden z pierwszych algorytmów- algorytm Euklidesa na obliczenie NWD

- NWD(**liczba całkowita** a, **liczba całkowita** b)
- **dopóki** b != 0

  c := **reszta z dzielenia** a **przez** b

  a := b

  b := c
- **zwróć** a

Algorytm Euklidesa - przykład

a b c

NWD(1071,1029) = NWD (1029, 1071 mod 1029 =42)

= NWD(1029,42) = NWD(42,1029 mod 42 = 21)

= NWD(42, 21) = NWD(21, 42 mod 21 = 0)

STĄD:

NWD(1071,1029)=21

Algorytm - cechy

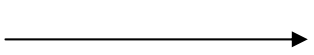
Każdy algorytm:

- Posiada dane wejściowe (w ilości większej lub równej 0) pochodzące z dobrze zdefiniowanego zbioru;
- Produkuje pewien wynik (niekoniecznie numeryczny);
- Jest precyzyjnie zdefiniowany (każdy krok algorytmu musi być jednoznacznie określony);
- Jest skończony (wynik algorytmu musi zostać „kiedyś” dostarczony – mając algorytm *A* i dane wejściowe *D* powinno być możliwe precyzyjne określenie czasu wykonania $T(A)$).

Metody opisu algorytmów

- Graficzny – schemat blokowy
- Pseudokod – mieszanka języka naturalnego i form składowych pochodzących z kilku języków programowania
- Kod w języku programowania

Konwencja graficznego opisu algorytmu (1)



Strzałki łączące

Początek lub koniec algorytmu

Odczyt danych

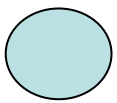
Zapis danych

Decyzja

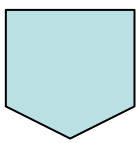
Konwencja graficznego opisu algorytmu (2)

Wykonanie instrukcji, proces

Wywołanie procesu wcześniej zdefiniowanego (procedury/funkcji)

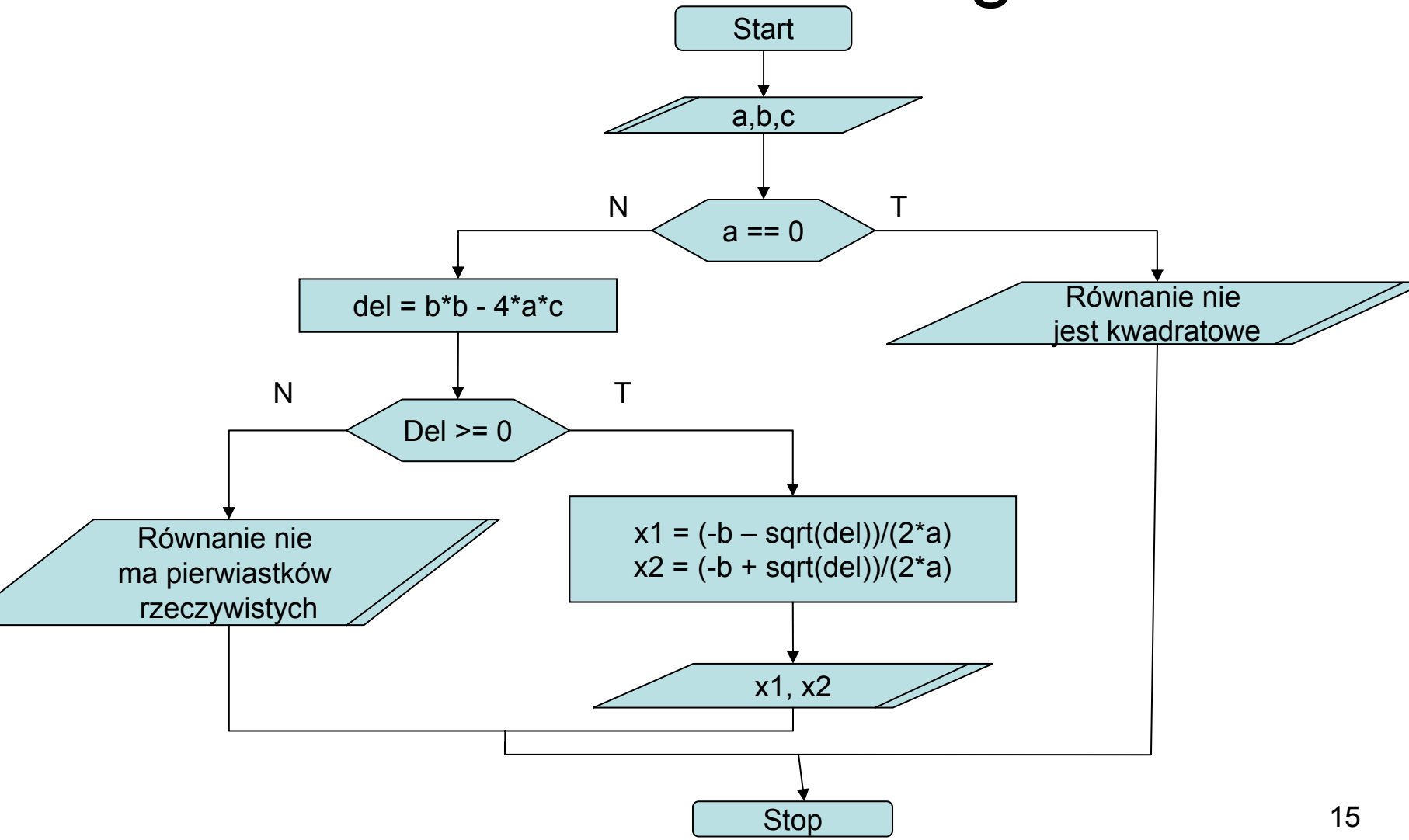


Łącznik stronicowy

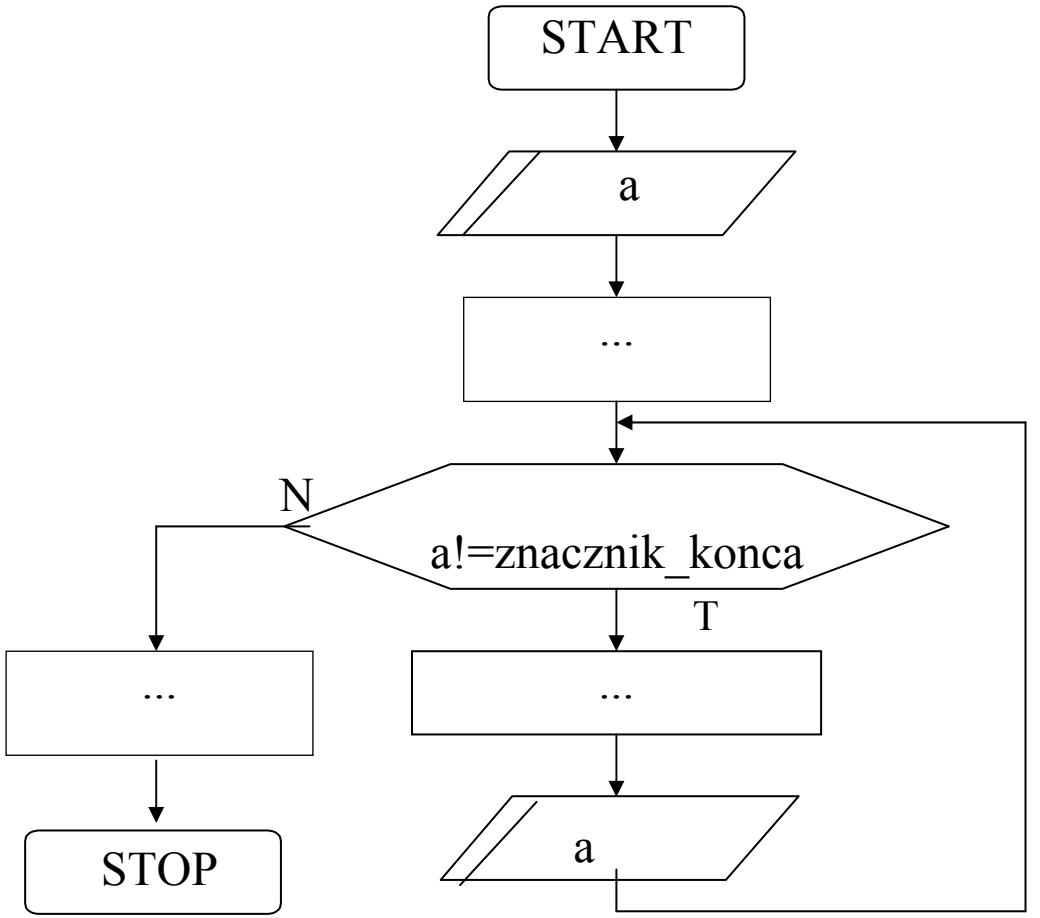


Łącznik międzystronicowy

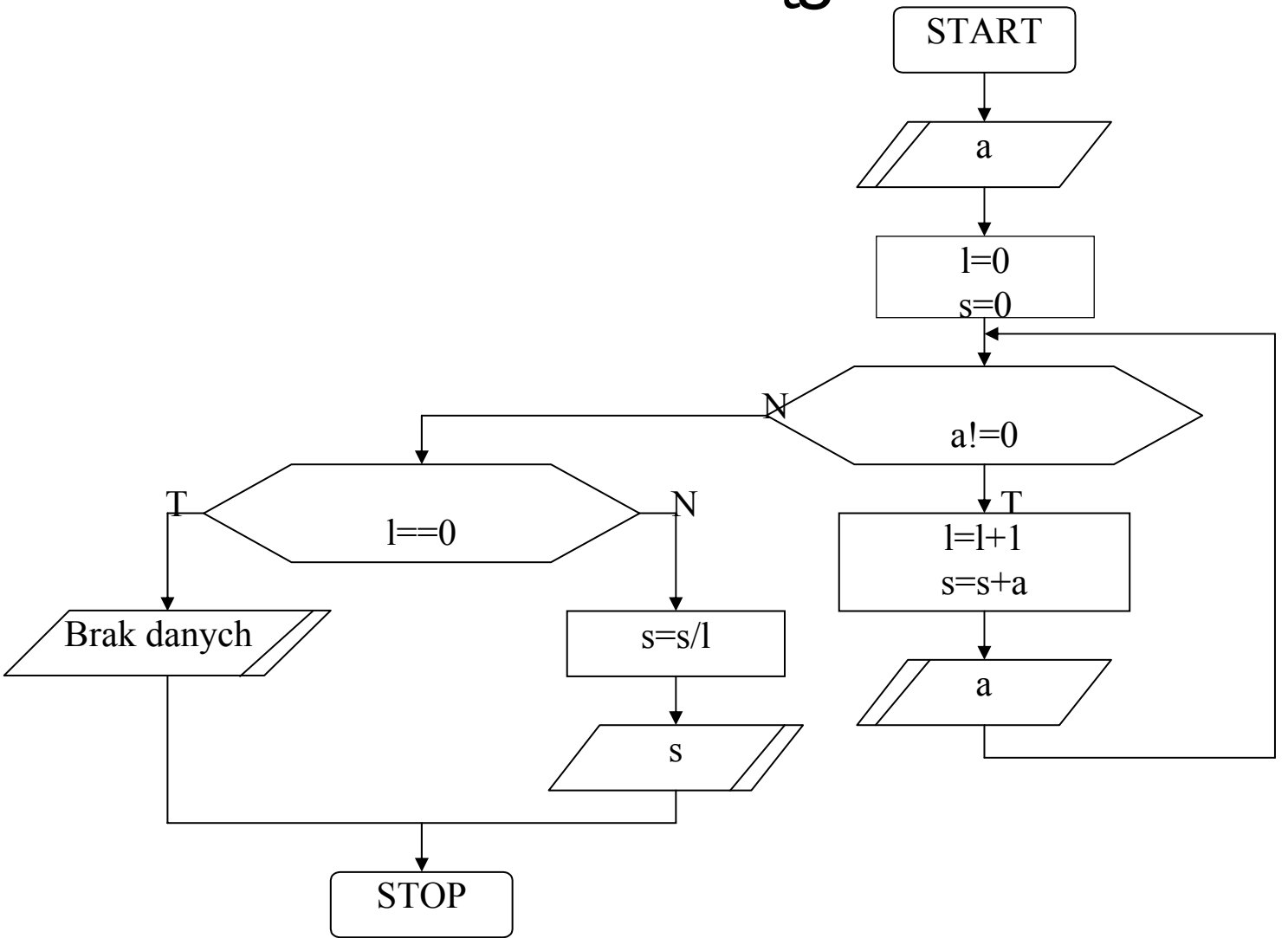
Przykład algorytmu – rozwiązanie rów. kwadratowego



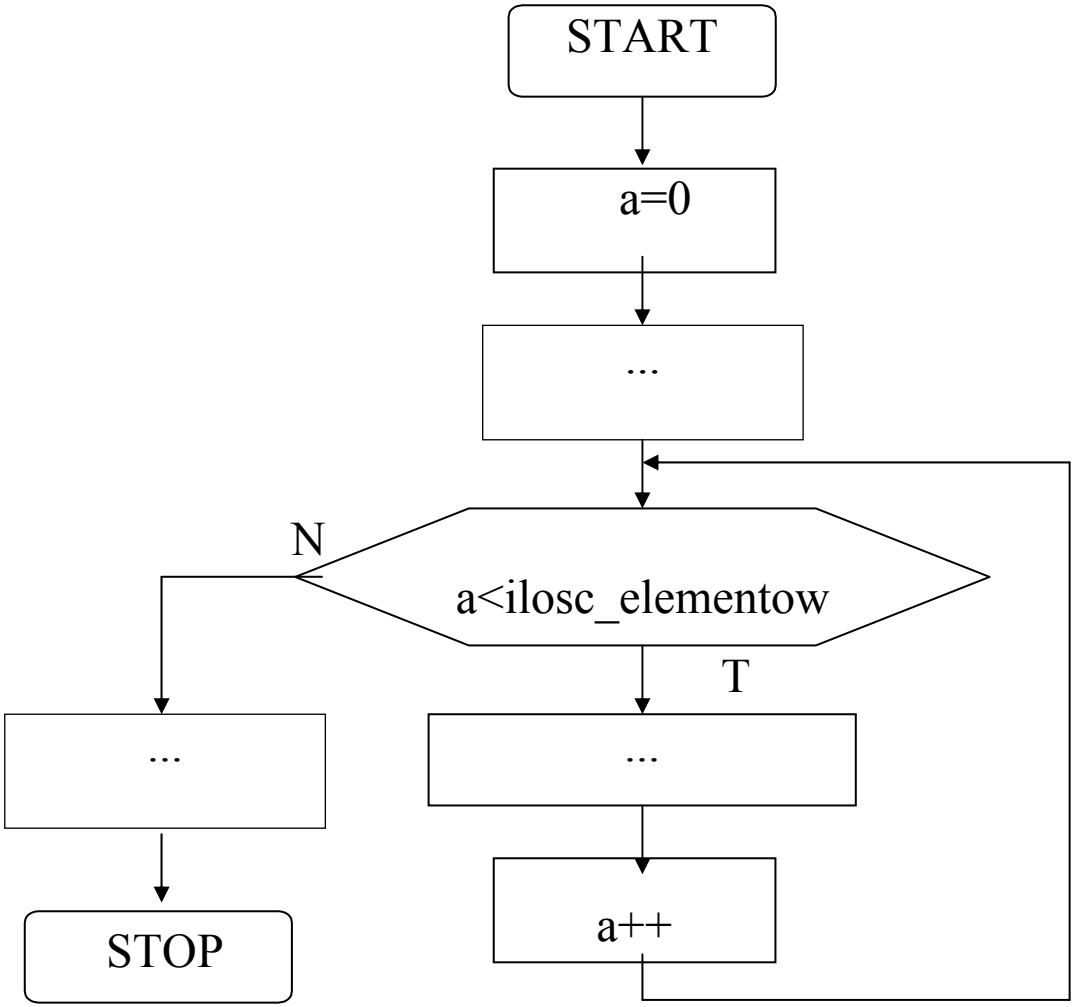
Przetworzenie ciągu danych o nieznanej długości, ale ze znacznikiem końca.



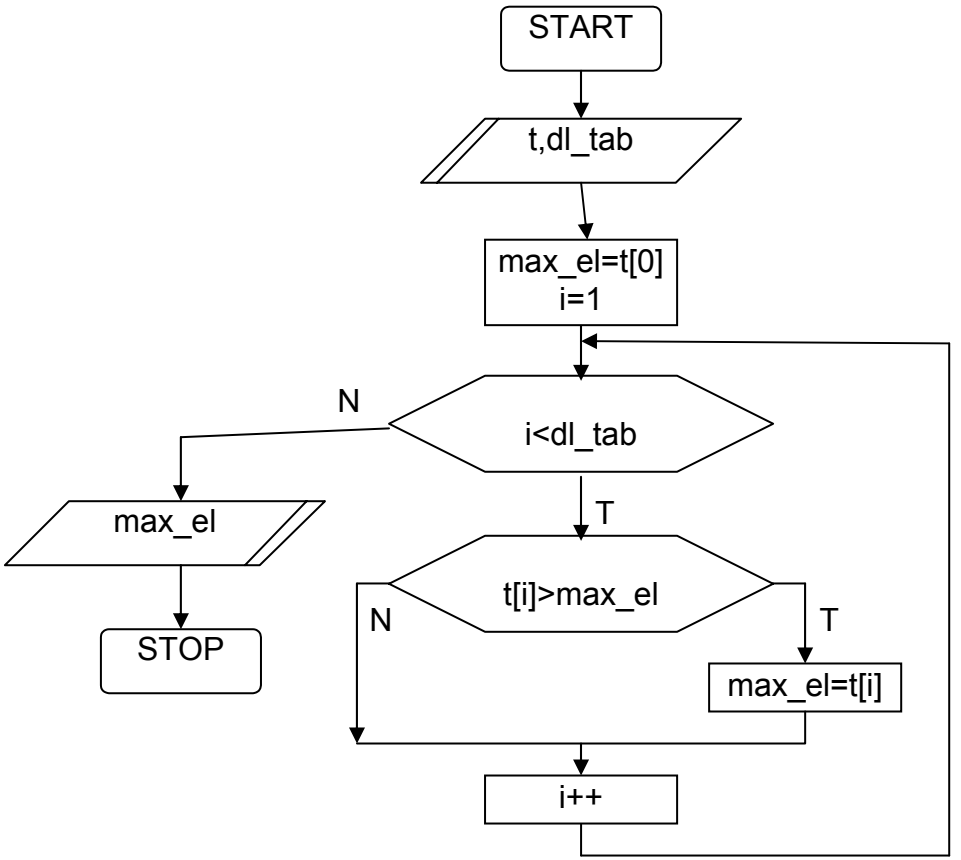
Przykład algorytmu – obliczanie średniej arytmetycznej z ciągu danych, 0 oznacza koniec ciągu



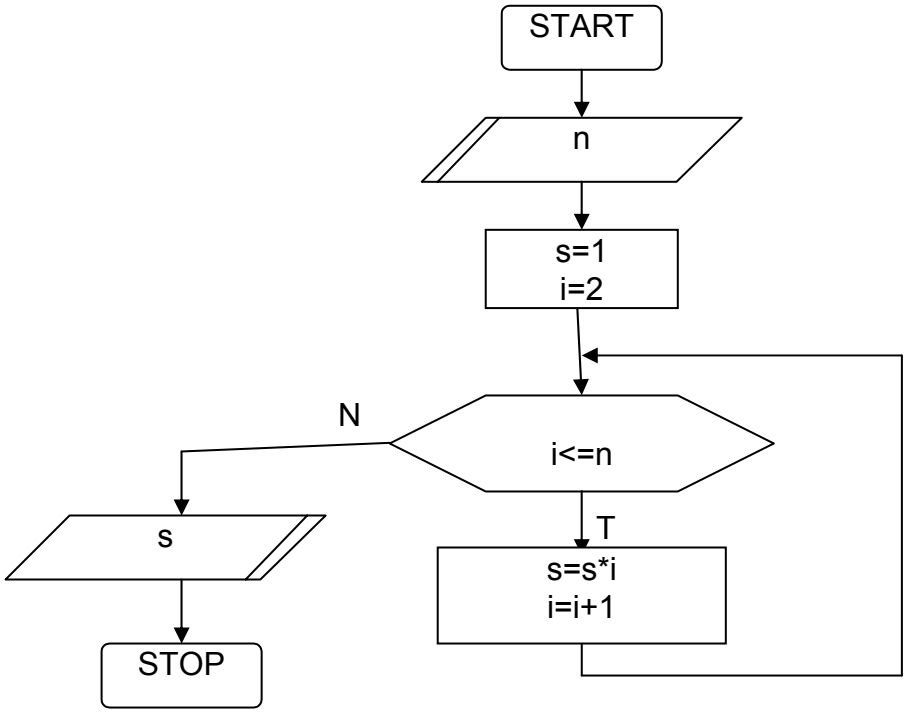
Przetworzenie ciągu danych o znanej długości.



Przykład algorytmu – wyznaczenie maksymalnego elementu ciągu zapisanego w tablicy *t* o długości *dl_tab*



Przykład algorytmu – obliczanie silni z *n*



Przykład algorytmu – sortowanie przez wstawianie

$a_{i=1}$ = -1, 0, 5, 8, -3, 2, -3

$a_{i=2}$ = -1, 0, 5, 8, -3, 2, -3

$a_{i=3}$ = -1, 0, 5, 8, -3, 2, -3

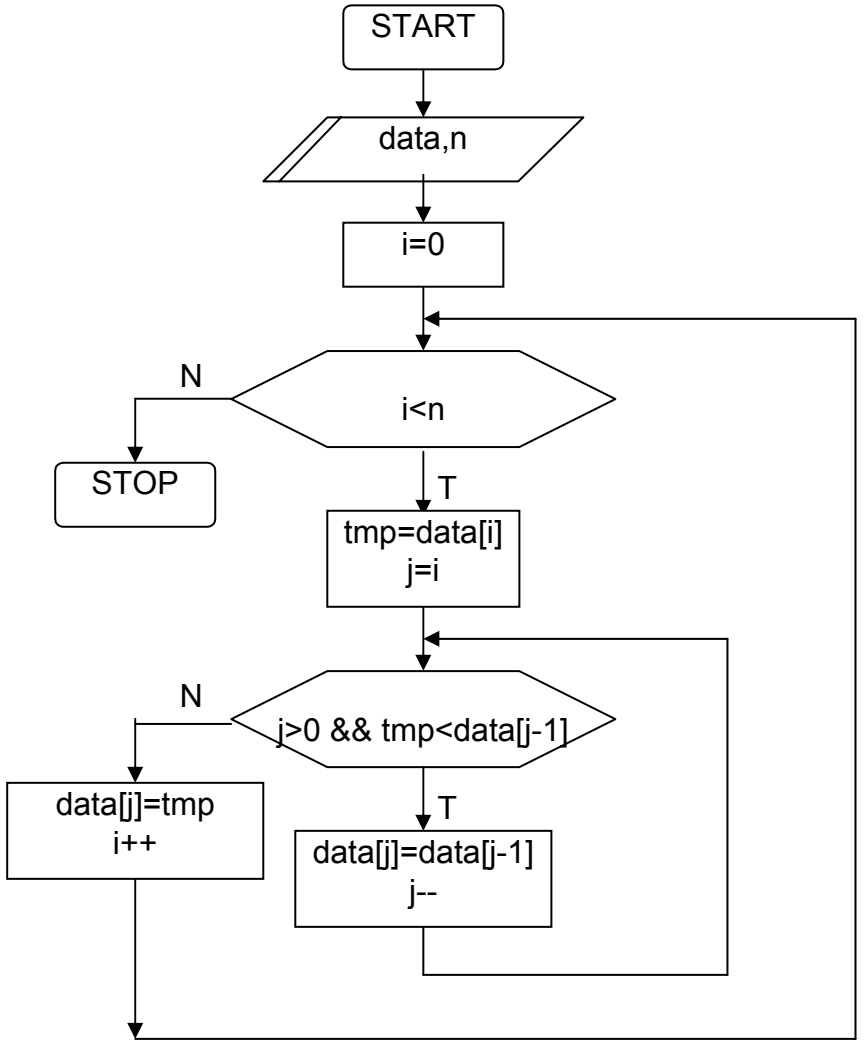
$a_{i=4}$ = -1, 0, 5, 8, -3, 2, -3

$a_{i=5}$ = -3, -1, 0, 5, 8, 2, -3

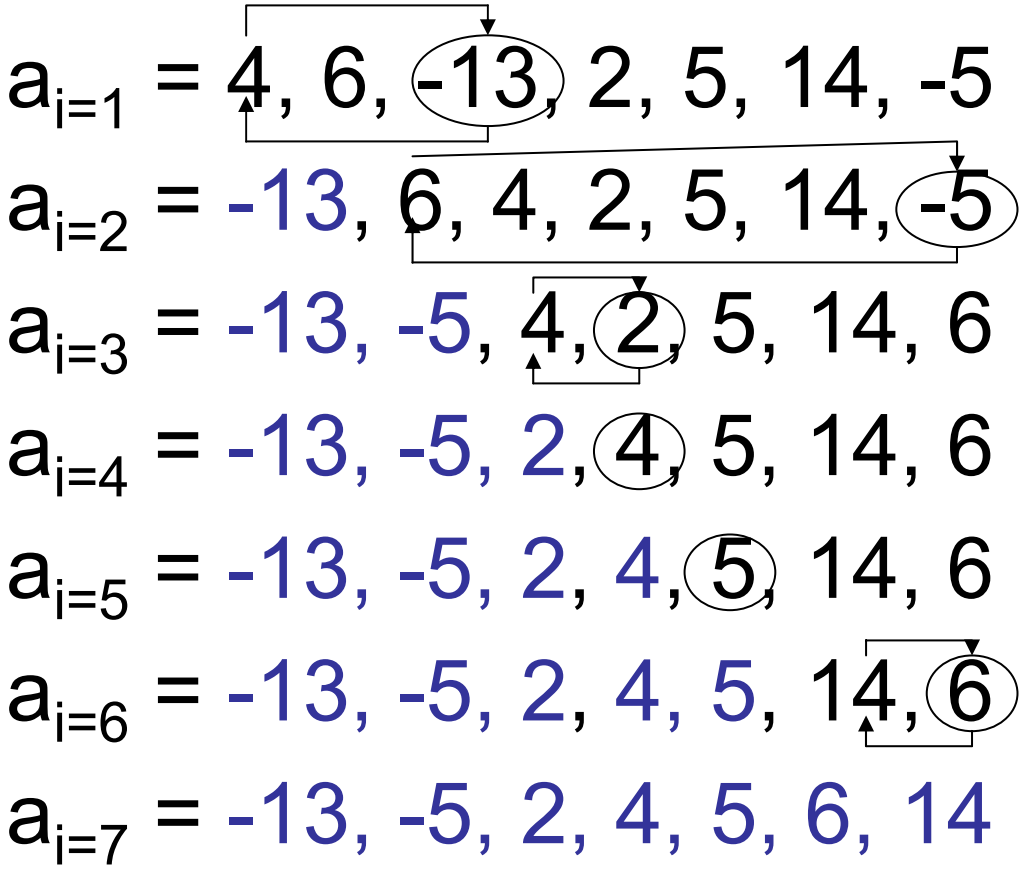
$a_{i=6}$ = -3, -1, 0, 2, 5, 8, -3

$a_{i=7}$ = -3, -3, -1, 0, 2, 5, 8

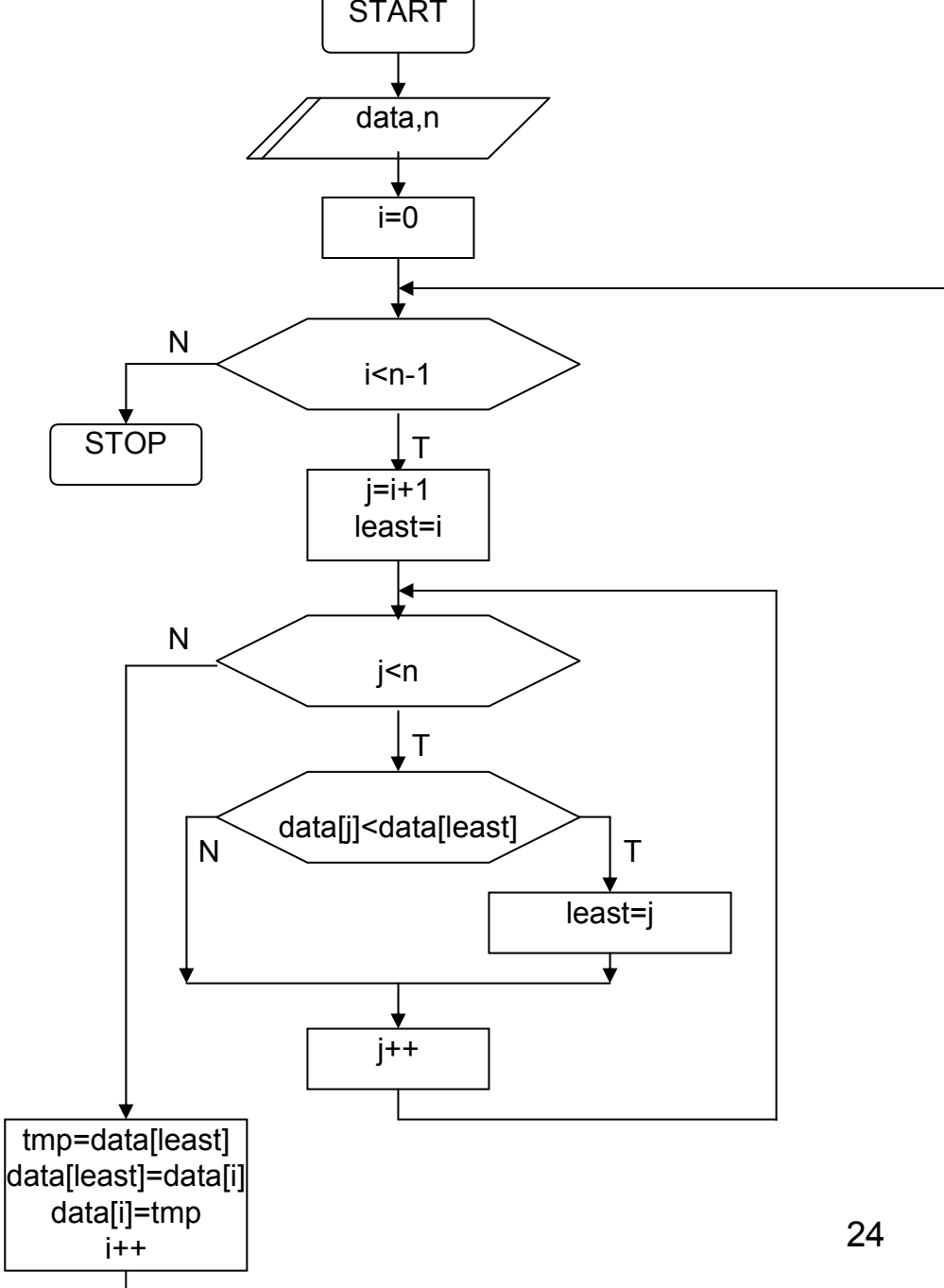
Przykład algorytmu – sortowanie przez wstawianie - implementacja



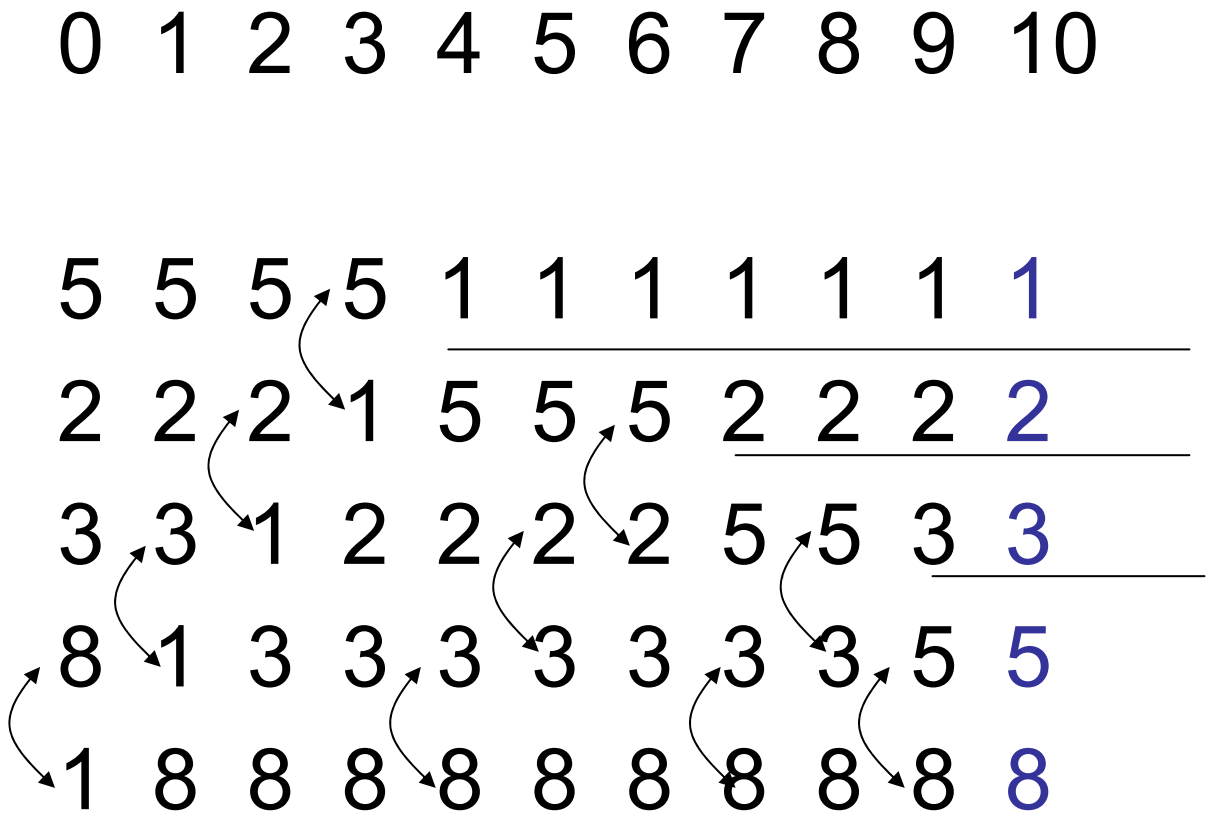
Przykład algorytmu – sortowanie przez wybieranie



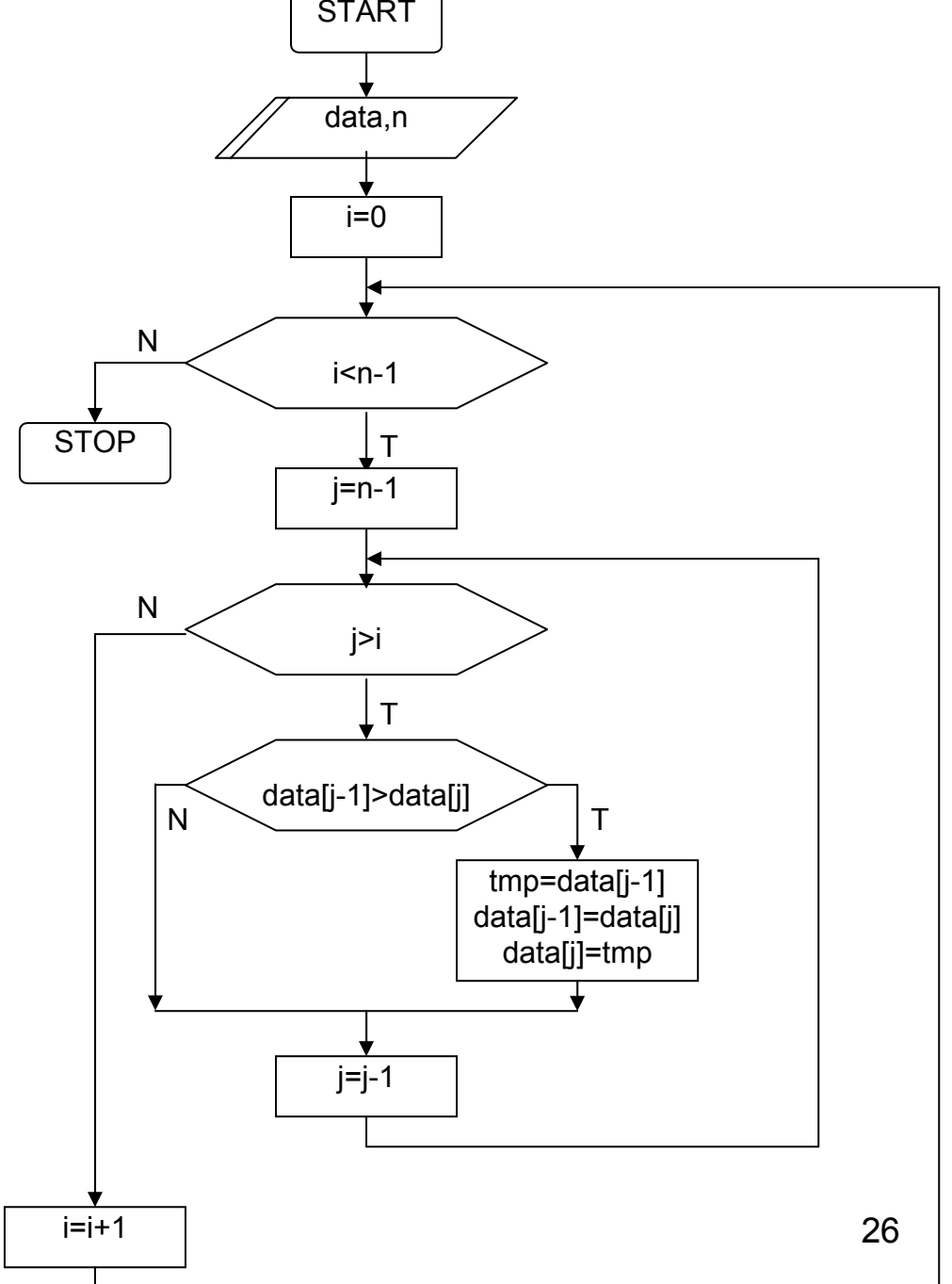
Przykład algorytmu – sortowanie przez wybieranie - implementacja



Przykład algorytmu – sortowanie bąbelkowe

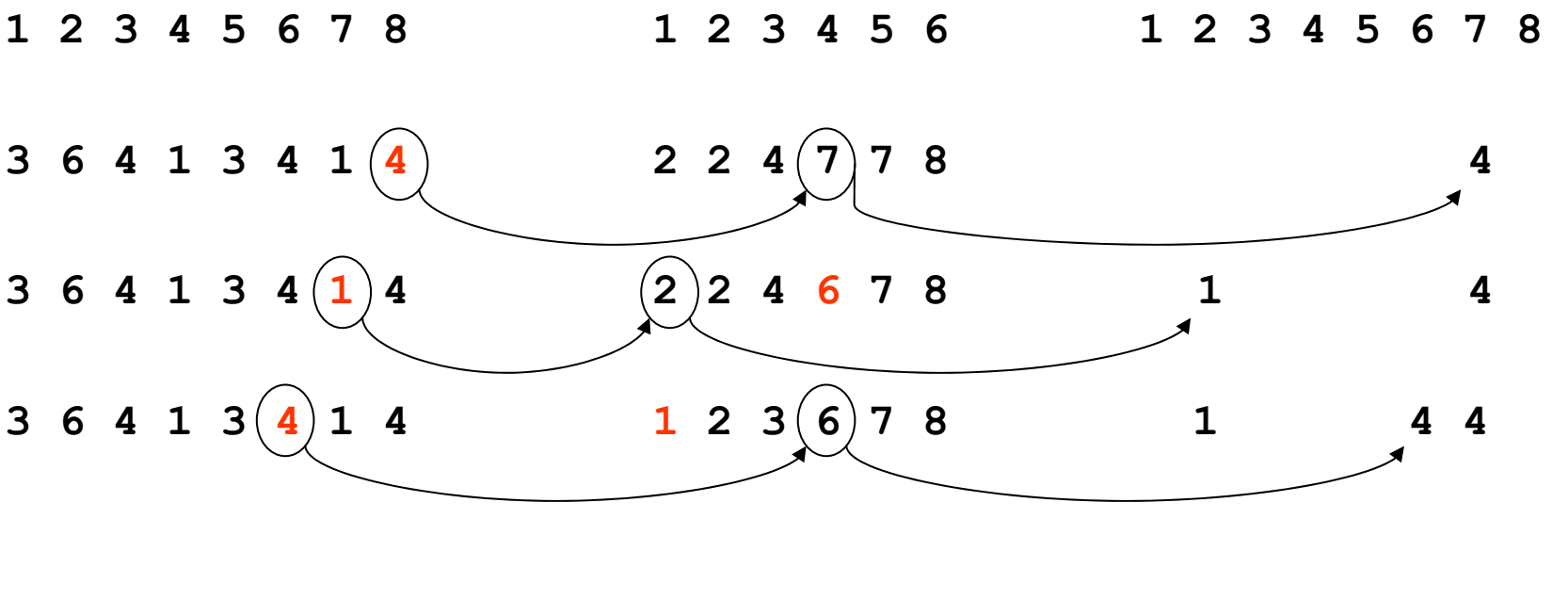


Przykład algorytmu – sortowanie bąbelkowe - implementacja



Przykład algorytmu – sortowanie przez zliczanie

```
      1 2 3 4 5 6 7 8
A   = 3 6 4 1 3 4 1 4
      1 2 3 4 5 6
C1  = 2 0 2 3 0 1
C2  = 2 2 4 7 7 8
```



Przykład algorytmu – wyszukiwanie binarne

Dany jest ciąg posortowany, należy sprawdzić, czy pewna liczba (np. 18) znajduje się w ciągu.

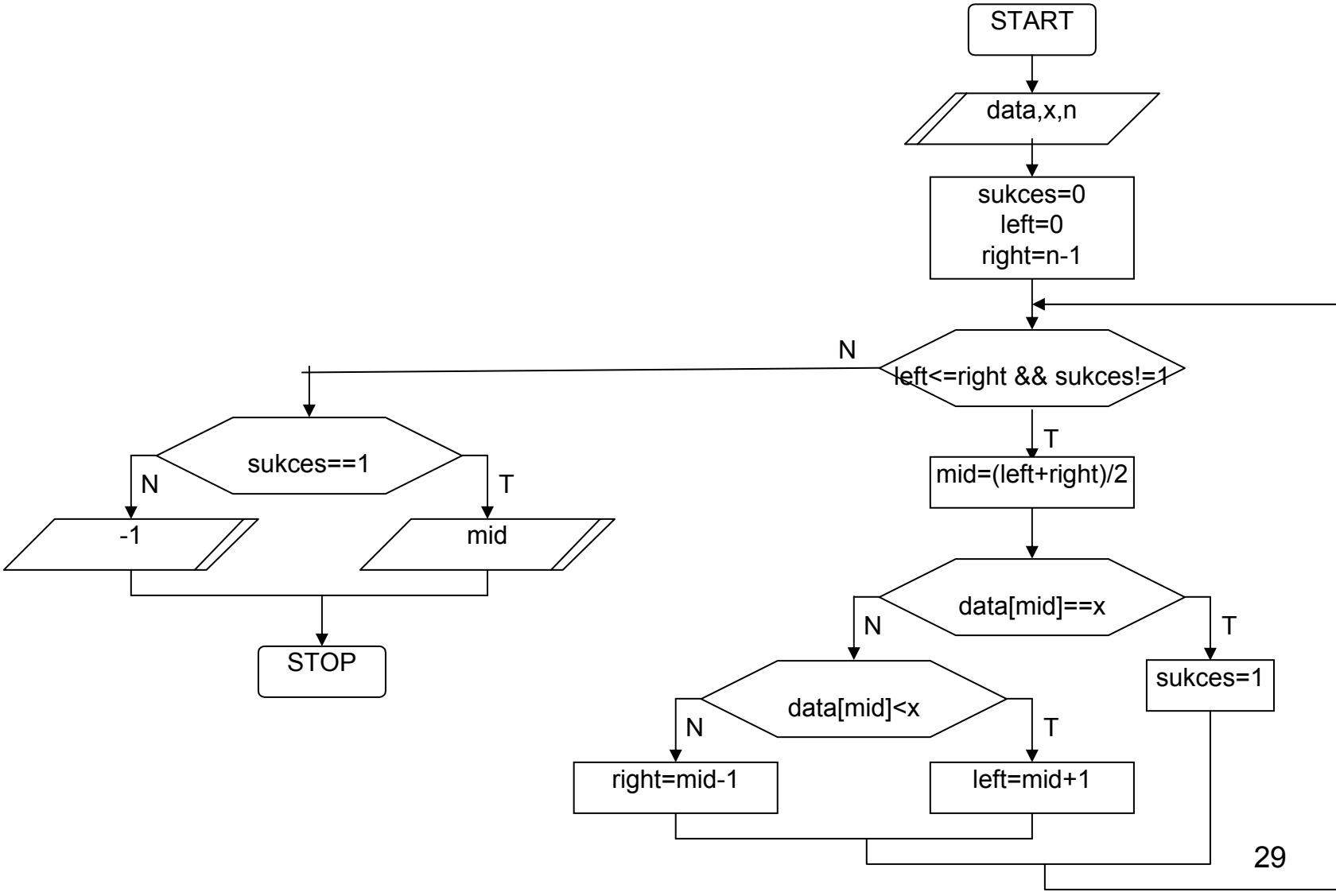
```
l/p:                  l                                   p
numery el. ciągu: 0  1  2  3  4  5  6  7  8  9 10 11
ciąg posortowany: 1, 2, 6,18,20,23,29,32,34,39,40,41
l == 0, p == 11; l < p stąd można kontynuować obliczenia
m = (l + p)/2 = (0 +11)/2 = 5
18 < 23 stąd poszukiwanie będzie się odbywać na lewo od 23
```

```
l/p:                  l           p
numery el. ciągu: 0  1  2  3  4
podciąg:          1, 2, 6,18,20
l == 0, p == 4; l < p => stąd można kontynuować obliczenia
m = (0 + 4) / 2 = 2
18 > 6 stąd poszukiwanie będzie się odbywać na prawo od 6
```

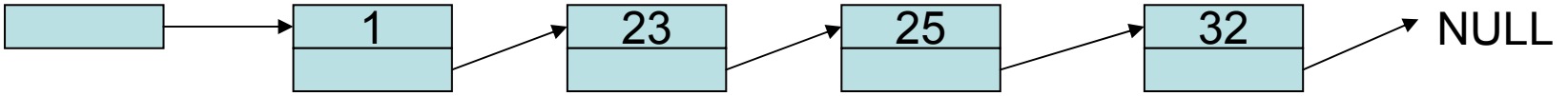
```
 l/p:                  l  p
numery el. ciągu:  3  4
podciąg:          18,20
l == 3, p == 4; l < p => stąd można kontynuować obliczenia
m = (3 + 4) / 2 = 3
```

Liczba 18 została znaleziona!

Przykład algorytmu – wyszukiwanie binarne, implementacja

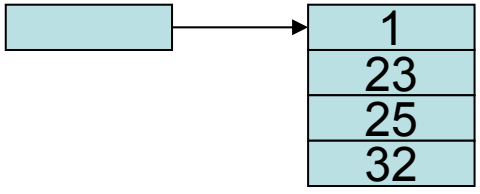


Przykłady złożonych struktur danych - **lista**



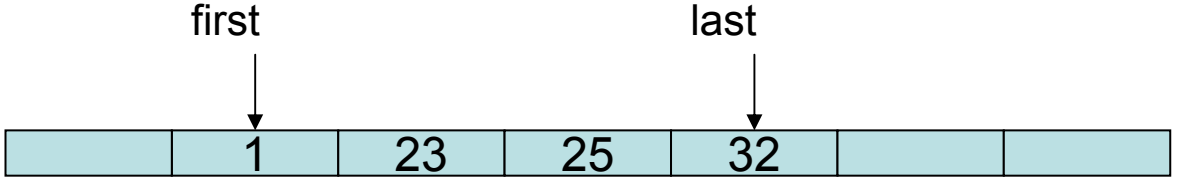
- Lista jest wiązaną strukturą danych,
- Każdy z elementów listy zawiera dane oraz dowiązanie do następnego,
- Istnieje umowa, że jeśli dowiązanie wskazuje na 0 (NULL), to jest to ostatni element listy,
- Pierwszym elementem jest zwykle specjalny obiekt zawierający tylko dowiązanie,
- W odróżnieniu od ciągłych struktur danych (np. tablic) lista nie potrzebuje jednego ciągłego obszaru pamięci do przechowywania informacji – dowiązanie może wskazywać w różne miejsca w pamięci,
- Listę można skonstruować z zastosowaniem tzw. Dynamicznego przydziału pamięci, co pozwala efektywniej zarządzać pamięcią: rezerwowana jest taka liczba pamięci, jaka wymagana jest w danej chwili,
- Najczęstsze zastosowania list:
  Bazy danych,

Przykłady złożonych struktur danych - **stos**



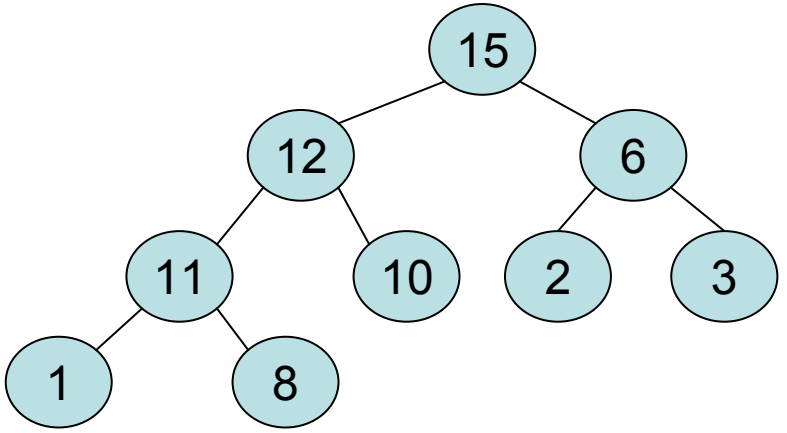
- Stos jest liniową strukturą danych, dostępną do zapisywania i odczytywania tylko z jednego końca,
- Stos można zdefiniować za pomocą operacji, które zmieniają jego stan:
  - initialize(stack) – opróżnienie stosu,
  - empty(stack) – sprawdzenie, czy stos jest pusty,
  - full(stack) – sprawdzenie, czy stos jest zapełniony,
  - push(el,stack) – powoduje umieszczenie el. na stos,
  - pop(stack) – powoduje zdjęcie najwyższego elementu ze stosu,
- Zastosowania:
  przechowywanie rejestrów podczas wywoływania podprogramów, grafika komputerowa – składnie przekształceń przestrzennych, przechowywanie zmiennych lokalnych w języku C, element szeregu algorytmów np. kalkulatora w odwrotnej notacji polskiej:
  2 3 5 * + < = > (3*5)+2
  push(2,stos); push(3,stos); push(5,stos);
  push((pop(stos) * pop(stos)),stos); push((pop(stos) + pop(stos)),stos)

Przykłady złożonych struktur danych – **kolejka FIFO**



- Kolejka jest listą (oczekujących) zwiększającą się przez dodanie elementów na jej koniec, a zmniejszająca się przez wyjmowanie elementów z jej początku.
- W odróżnieniu od stosu w kolejce wykorzystywane są oba końce – jeden do wstawiania nowych elementów, drugi do ich usuwania.
- Kolejkę można zdefiniować za pomocą operacji, które zmieniają jego stan:
  - initialize(queue) – opróżnienie kolejki,
  - empty(queue) – sprawdzenie, czy kolejka jest pusta
  - full(queue) – sprawdzenie, czy kolejka jest zapełniony,
  - enq(el,queue) – powoduje umieszczenie el. na końcu kolejki,
  - deq(queue) – powoduje usunięcie pierwszego elementu z kolejki,
- Zastosowania:
  komunikacja pomiędzy urządzeniami i procesami obliczeniowymi w komputerze

Przykłady złożonych struktur danych – **drzewo**



Przykład drzewa binarnego - kopiec

- Drzewo składa się z wierzchołków i krawędzi,
- Wierzchołki nie posiadające dzieci nazywane są liśćmi,
- Drzewa, w których z każdego wierzchołka mogą „wyrastać” tylko dwa wierzchołki potomne nazywane są drzewami binarnymi,
- Każdy wierzchołek może mieć co najwyżej jednego rodzica,
- Struktura drzewa pozwala nie tylko na porządkowanie danych ale również na ich hierarchizację.
- Zastosowania drzew:

  kompilatory – do przetwarzania tekstu programu, wydajne algorytmy sortowania (np. przez kopcowanie), do opisu wyrażeń matematycznych.

Pominięte zagadnienia

- Przy konstruowaniu algorytmów ocenia się ich złożoność obliczeniową, istnieją problemy obliczeniowe tak złożone, że nigdy nie zostaną rozwiązane przez komputer np.
  - gra w szachy
- Istotnym działem algorytmiki są tzw. algorytmy rekurencyjne wprost pozwalające na implementację rekurencyjnie zdefiniowanych pojęć, np. silnia:
  0! = 1
  n! = n * (n-1)! Gdzie n >= 1

```
unsigned long int silnia(int x)
{   if(x==0) return 0;
    else return x*silnia(x-1);}
```

- Przedstawiono tylko wybrane algorytmy i struktury danych, szerszy przegląd znajduje się w literaturze przedłożonej na pierwszych zajęciach.