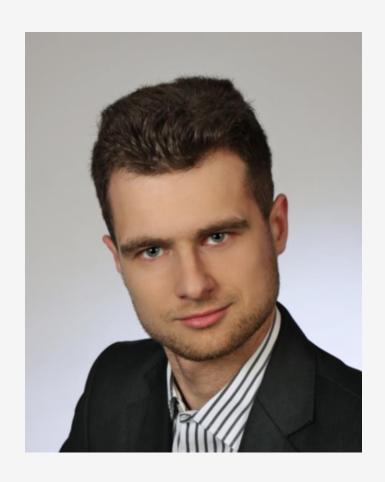


Algorytmy i struktury danych







Senior Software Engineer w EPAM Systems Inc.

Doktorant na wydziale Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej na AGH

Kontakt: ap.lukaszpaluch@gmail.com



1. Jak wyglądać będzie szkolenie?



- 1. Jak wyglądać będzie szkolenie?
- 2. Co z przerwami?



- 1. Jak wyglądać będzie szkolenie?
- 2. Co z przerwami?
- 3. Jak wam się podoba Python???



- 1. Jak wyglądać będzie szkolenie?
- 2. Co z przerwami?
- 3. Jak wam się podoba Python???
- 4. Jakie tematy będą poruszane w trakcie tego bloku?

Plan na blok



- Algorytmy
- 2. Schematy blokowe
- 3. Rekurencja
- 4. Złożoność obliczeniowa
- 5. Wyszukiwanie binarne
- 6. Sortowanie
- 7. Struktury danych



Algorytmy – ćwiczenia

Algorytmy



- Algorytm przepis na rozwiązanie problemu obliczeniowego.
- Różnica między algorytmem a wzorcem projektowym polega na tym, że ten drugi rozwiązuje problem architektury oprogramowania, podczas gdy algorytm skupia się na obliczeniach.
- Algorytm ciąg operacji przeprowadzony na różnych strukturach danych zmierzający do rozwiązania problemu programistycznego.

Algorytmy – sposób postępowania



- 1. Sformułowanie zadania ustalamy jaki problem ma rozwiązywać algorytm.
- 2. **Dane wejściowe** czego oczekujemy, co powinniśmy dostać na wejściu, jaki typ, wartości, ograniczenia.
- 3. Określenie wyniku co powinien zwrócić algorytm (napis? listę? funkcję?)
- 4. Ustalenie metody wykonania zadania tutaj może się pojawić kilka sposobów na rozwiązanie jednego problemu (to my ustalamy, który jest dla nas najodpowiedniejszy).
- 5. Zapisanie algorytmu za pomocą wybranej metody.
- 6. Analiza i testowanie zaproponowanego algorytmu i jego działania.
- 7. Ocena skuteczności algorytmu efektywność, prostota, łatwość implementacji; niech świat się dowie i nas osądzi ☺

Pierwszy powtarzający się znak



Zaproponuj algorytm i napisz funkcję, która przyjmować będzie jeden argument typu string.

Zadaniem funkcji będzie wskazanie litery, która jako pierwsza zostanie powtórzona w stringu (dla uproszczenia przyjmujemy, że napis będzie posiadał tylko małe litery).

Przykład: "akapit" -> 'a'

Liczba pi



Oblicz wartość liczby π.

Algorytmy



Różne sposoby prezentacji algorytmów:

- Słowny
- Schemat blokowy
- Pseudokod
- Kod

In [20]: def add(a, b):
...: return a + b

- 1. Wlej wodę do garnka
- 2. Włóż do garnka x jajek
- 3. Zagotuj wodę

Lampa nie działa

Czy lampa

jest podpięta

Czy żarówka

jest spalona?

Kup nową lampę Podłącz ją

Zmień żarówkę

- 4. Gotuj jajka 8 minut
- 5. Wylej wodę z garnka
- 6. Schłódź jajka zimną wodą

jeżeli numer karty kredytowej jest ważny to
wykonanie transakcji w oparciu o numer karty i zamówienie
w przeciwnym razie
wyświetlenie wiadomości o niepowodzeniu
koniec warunku

Sposoby opisywania algorytmów



Zaproponuj słowny opis algorytmu dodawania do siebie trzech liczb.

Czy było to trudne? Długie? Żmudne?



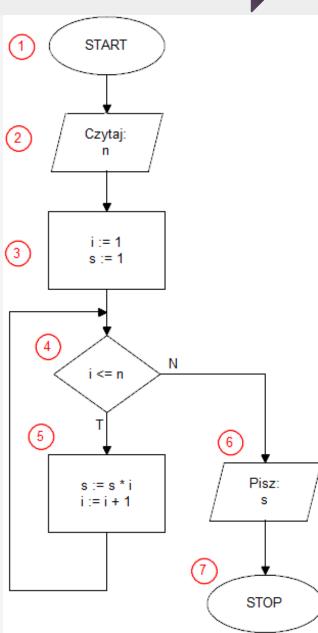
A co w przypadku słownego opisania algorytmu obliczania liczby pi? NWD? Zamiany liczby dziesiętnej na binarną?





Schemat blokowy to graficzny zapis algorytmu na potrzeby rozwiązania konkretnego zadania. Na jego podstawie ustalamy/odczytujemy kolejność wykonywanych instrukcji i przepływ algorytmu.

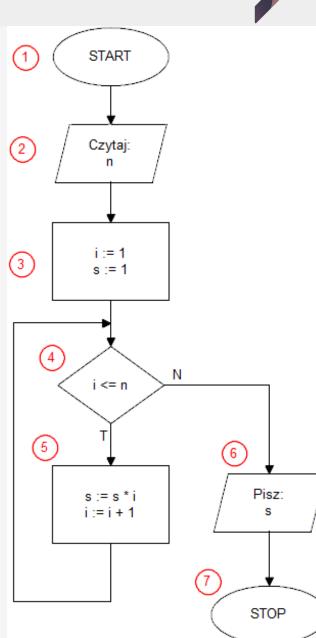
- proste w użyciu reprezentują kolejne czynności w algorytmie,
- prosta budowa mała ilość elementów (bloków),
- można je tworzyć z wykorzystaniem zapisu składniowego dowolnie wybranego języka (różne operatory, np. := vs =),
- przydatne w przypadku tworzenia zaawansowanych algorytmów,
- na ich podstawie dużo prościej o napisanie kodu danego algorytmu





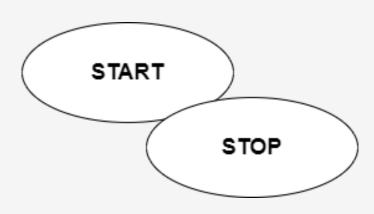
Wyróżniamy kilka rodzajów schematów (związane jest to bezpośrednio z użytymi blokami/stopniem zaawansowania schematu):

- Proste (sekwencyjne) nie używa się w nich bloków warunkowych. W takiej sieci działań kolejność realizacji poszczególnych operacji jest ściśle określona i żadna z nich nie może być pominięta ani powtórzona.
- **Z rozwidleniem** zawiera w sobie wybór jednej z kilku możliwych dróg realizacji danego zadania. Istnieje w nim przynajmniej jeden blok warunkowy.
- Z pętlą często w trakcie realizacji danego zadania konieczne jest powtórzenie niektórych operacji różniących się jedynie zestawem danych. Pętla obejmuje tę część bloków, która ma być powtarzana.
- Złożone będące kombinacją powyższych sieci.





Bloki graniczne



Są to bloki w kształcie owalu, od których rozpoczyna się i na któych kończy reprezentację blokową algorytmu. Z bloku START (zazwyczaj tak nazywanego) wychodzi tylko i wyłącznie jedna strzałka w dół, nie wchodzi za to żadna. Do bloku STOP (KONIEC itd.) wchodzi jedna strzałka, natomiast nie wychodzi już żadna – w momencie kiedy algorytm dotrze do tego bloku, kończy się jego wykonywanie.

Funkcja: oznaczenie początku i końca algorytmu



Strzałka/łącznik

Strzałka (łącznik) wskazuje jednoznacznie kierunek przepływu instrukcji – nigdy nie powinna się rozwidlać, algorytm powinien jasno przechodzić z jednego bloku do drugiego. Należy jednak pamiętać, że możliwy jest powrót za pomocą strzałki z bloku poniżej do bloku wyżej (mamy wtedy do czynienia z zapętleniem instrukcji).

Funkcja: wskazuje kolejność wykonywania się instrukcji w algorytmie



Skrzynka operacyjna

Skrzynka operacyjna jest reprezentowana przez prostokąt. W jej wnętrzu znajdują się wszystkie operacje/instrukcje wykonywane podczas działania algorytmu (za wyjątkiem instrukcji warunkowej). W tym bloku można zdefiniować zmienną, zmienić jej wartość, napisać słownie jakąś prostą komendę (np "dopisz literę na koniec aktualnego stringa").

Funkcja: przechowywanie instrukcji, któe powinny się w danym kroku algorytmu wykonać

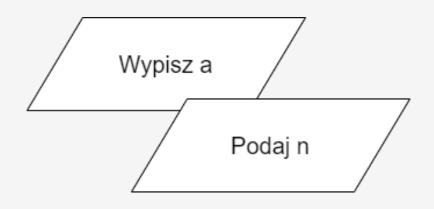


- =, == to operatory porównania logicznego
- :=, =, <- to operatory przypisania wartości
- !=, <> to operatory nierówności dwóch wartości

Najczęściej stosuje się te pogrubione, jednakże spotkać można i schematy z innymi znakami (mogącymi zresztą być zależnymi od wybranego języka programowania) – warto więc być świadomym ich istnienia.



Skrzynka wejścia/wyjścia

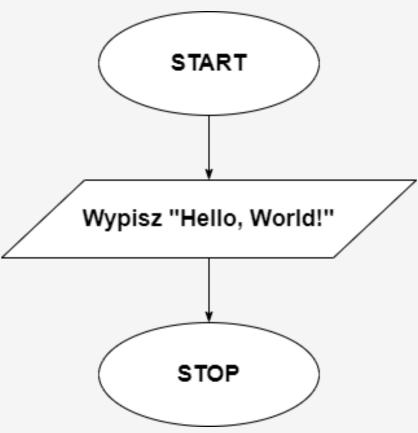


Reprezentowane na schemacie poprzez równoległobok, skrzynki wejścia/wyjścia służą do realizacji kontaktu z użytkownikiem poprzez komendy wypisujące wartości (np. print) bądź proszące użytkownika o podanie takowej (np. input).

Funkcja: wprowadzanie/wyprowadzanie danych

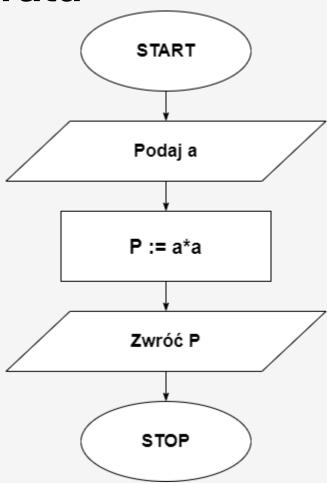


Algorytm Hello World





Obliczanie pola kwadratu



Schematy blokowe - przykład

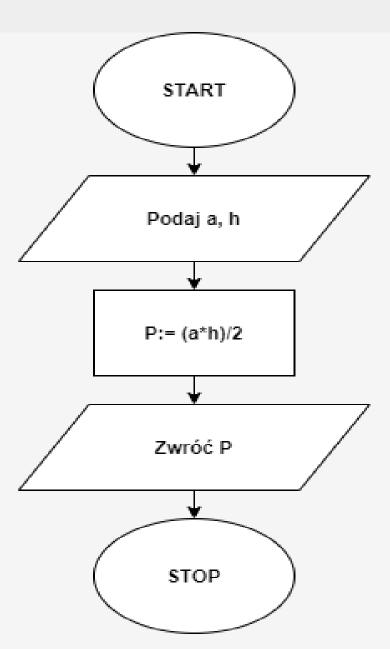


Narysuj schemat blokowy dla algorytmu obliczającego pole

trójkąta (wzór:
$$P = \frac{a*h}{2}$$
).

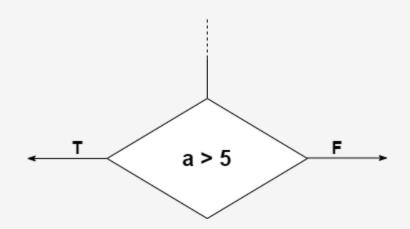
Schematy blokowe – przykład (rozwiązanie)







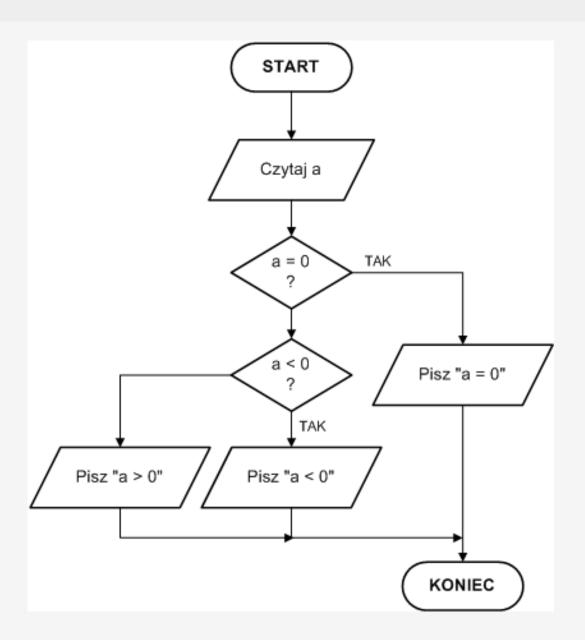
Skrzynka warunkowa



Skrzynka warunkowa przechowuje instrukcje wyboru (w rombie). Wchodzi do niej jedna strzałka, natomiast wychodzą dwie: dla wartości TAK (kiedy warunek zapisany w rombie jest prawdziwy) oraz dla wartości NIE (w momencie, którym nie jest).

Funkcja: sprawdzanie warunku (PRAWDA/FAŁSZ)





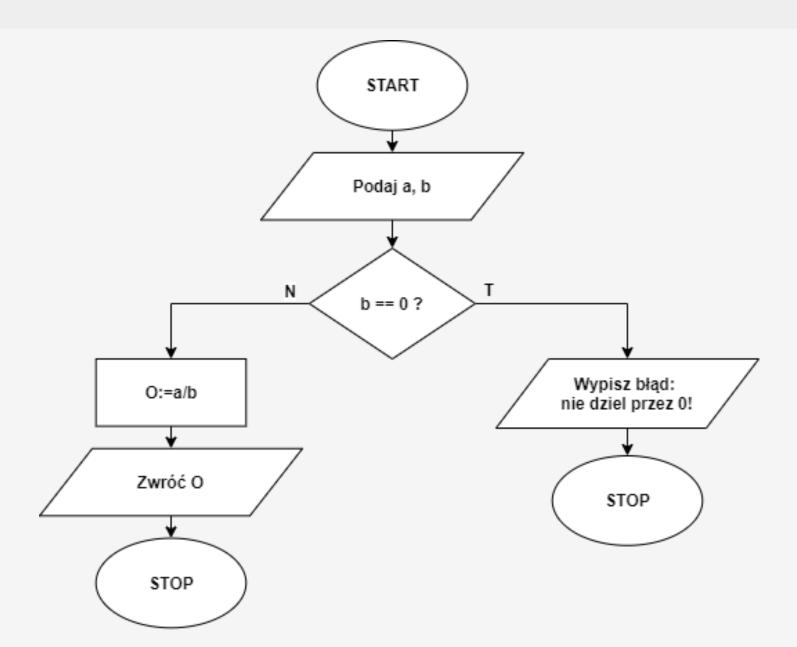
Schematy blokowe - przykład



Zaproponuj schemat blokowy algorytmu dzielącego jedną liczbę przez drugą (pamiętaj o ograniczeniach!)

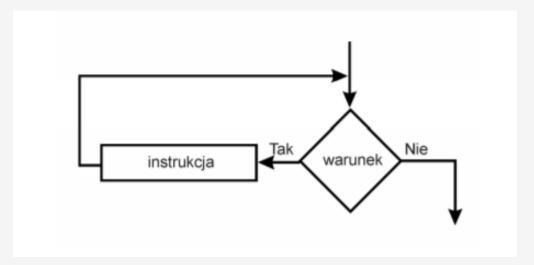
Schematy blokowe – przykład (rozwiązanie)



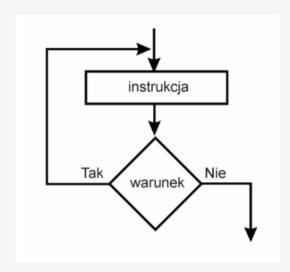




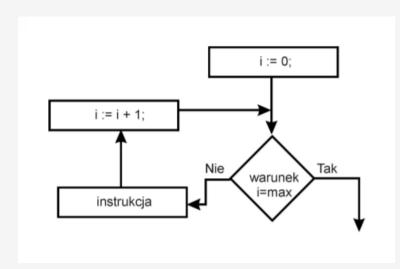
Pętla



Reprezentacja pętli **while**: najpierw sprawdzenie warunku, potem wykonanie instrukcji, ostatecznie wrócenie do punktu sprawdzenia warunku (rozpoczęcie kolejnej iteracji w pętli).

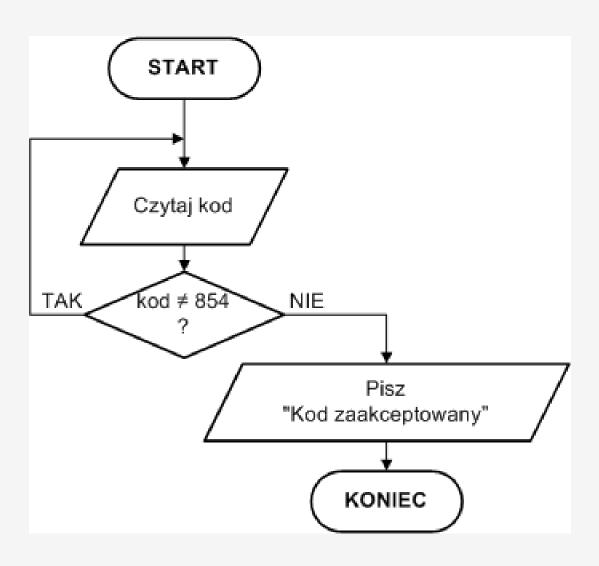


Reprezentacja pętli until (do while): najpierw instrukcja, a potem sprawdzenie warunku (plus potencjalne powtórzenie wykonania instrukcji.

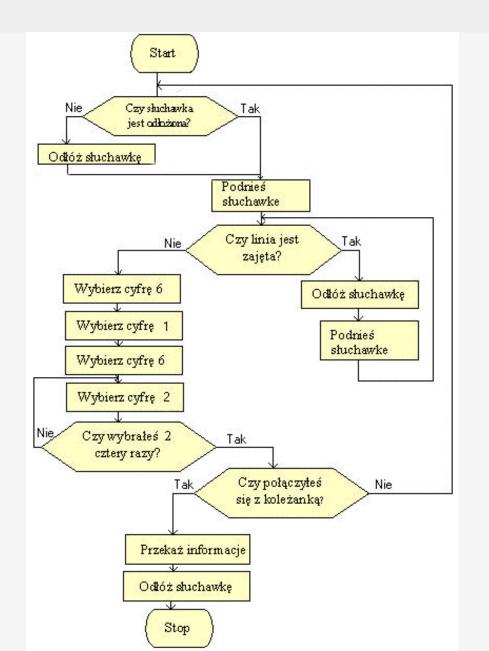


Symulacja pętli **for**: zapętlenie wykonywania się instrukcji określoną na schemacie ilość razy; *i* zwiększane do momentu zrównania się wartością z *max*.









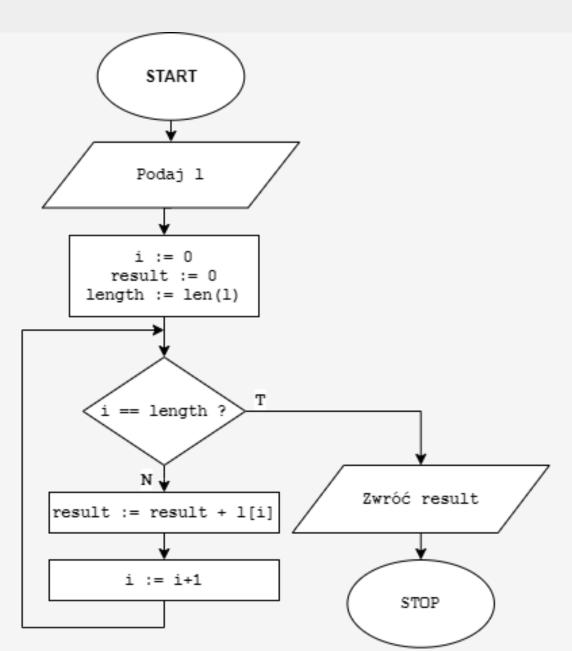
Schematy blokowe - ćwiczenia



Przygotuj schemat blokowy dla algorytmu obliczającego sumę wszystkich liczb w liście podanej na wejściu.

Schematy blokowe – ćwiczenia (rozwiązanie)





Schematy blokowe – sposób postępowania



- 1) Każda operacja jest umieszczona w skrzynce
- 2) Schemat ma tylko jedną skrzynkę "START" i przynajmniej jedną skrzynkę "KONIEC"
- 3) Skrzynki są ze sobą połączone.
- 4) Ze skrzynki wychodzi jedno połączenie; wyjątek stanowią skrzynki: "KONIEC" (z której nie wychodzą już żadne połączenia) oraz "warunkowa" (z której wychodzą dwa połączenia opisane TAK i NIE w zależności od tego czy warunek jest spełniony, czy nie, można wyjść jedną z dwóch dróg)
- 5) Przepływ instrukcji może zostać zakłócony poprzez pętle lub skrzynki warunkowe

System dziesiętny -> system binarny



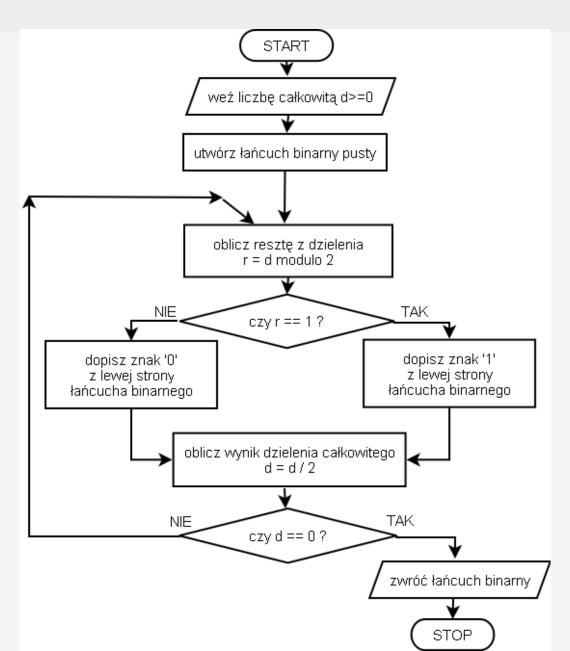
Napisz funkcję, która przyjmować będzie jeden argument: liczbę całkowitą w systemie dziesiętnym.

Zadaniem funkcji będzie zamiana reprezentacji tej liczby na system binarny (dwójkowy).

Funkcja powinna zwracać string, np. '100110'.

System dziesiętny -> system binarny





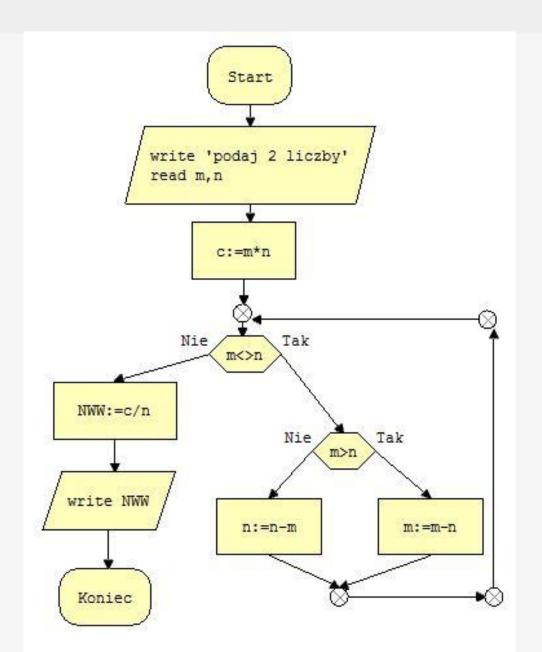
Algorytmy



Na podstawie schematu blokowego napisz funkcję wyliczającą najmniejszą wspólną wielokrotność dwóch liczb podanych jako argumenty.

Algorytmy







Rekurencja

\$

Przykład rekursji "z życia":



Żródło: https://greatfon.com/pl/pictures/101588



- Definicja: odwoływanie się funkcji do samej siebie.
- Najprostszy przykład ze szkoły średniej, silnia:

• Funkcja rekurencyjna zawiera odwołanie się do samej siebie oraz warunek, który kończy funkcję i zwraca ostateczny wynik.

Warunek końcowy dla silni:

$$1! = 1$$
 $0! = 1$

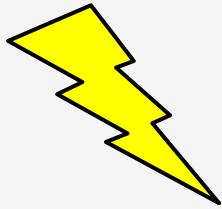


```
In [4]: def silnia(n):
  ...: if n == 1 or n == 0:
  ...: return 1
  ...: else:
  return n * silnia(n-1)
In [5]: silnia(5)
Out[5]: 120
```



Uwaga na nieskończone "pętle" w funkcjach rekurencyjnych!!!

```
In [1]: def inf recursion(number):
  ...: if number != 0:
  inf recursion(number)
       return 0
In [2]: inf_recursion(0)
```





Uwaga na nieskończone "pętle" w funkcjach rekurencyjnych!!!

```
[3]: inf_recursion(1)
lecursionError
                                         Traceback (most recent call last)
<ipython-input-3-992d325bfa87> in <module>
---> 1 inf recursion(1)
<ipython-input-1-08121247265b> in inf recursion(number)
     1 def inf recursion(number):
     if number != 0:
               inf_recursion(number)
        return 0
... last 1 frames repeated, from the frame below ...
<ipython-input-1-08121247265b> in inf_recursion(number)
     1 def inf recursion(number):
         if number != 0:
               inf_recursion(number)
          return 0
 ecursionError: maximum recursion depth exceeded in comparison
```





Limit "głębokości" rekurencji można poszerzać lub zawężać.

```
In [6]: import sys
In [7]: sys.setrecursionlimit(99999999)
```

Rekurencja



Napisz funkcję wyliczającą *n-ty* element ciągu Fibonacciego wykorzystując podejście rekurencyjne.



Czas na typowe zadanie rekrutacyjne z Microsoftu/Amazona/Google'a...



Złożoność obliczeniowa

Złożoność obliczeniowa



- Miara służąca do porównywania efektywności algorytmów.
- Może być zarówno czasowa (jak długo trwa algorytm), jak i pamięciowa (jak dużo miejsca w pamięci zabiera algorytm podczas przeprowadzania obliczeń).
- Ze względu na różne konfiguracje komputerów, pod uwagę bierze się zazwyczaj to, o ile dłużej i o ile pamięci więcej zajmie algorytm wraz ze zwiększaniem się danych wejściowych (np. coraz większa lista danych).

Złożoność obliczeniowa



stała: O(1)

logarytmiczna: O(log n)

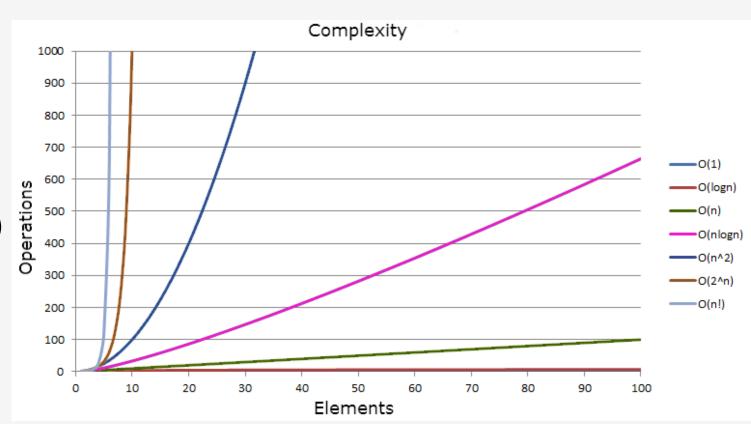
• liniowa: O(n)

liniowo-logarytmiczna: O(n * log n)

wielomianowa: O(n^c)

wykładnicza: O(cⁿ)

• silnia: O(n!)



Złożoność obliczeniowa - przykłady



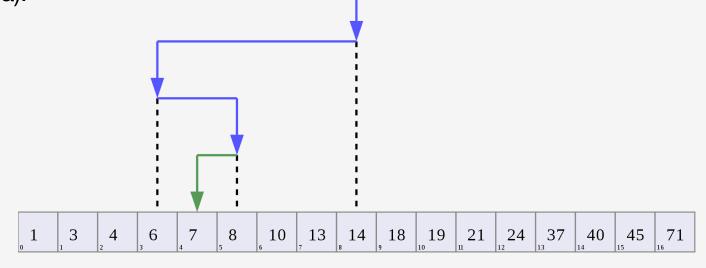
- O(1) zwrócenie pierwszego elementu tablicy, dodawanie, stworzenie nowej zmiennej
- O(log n) wyszukiwanie binarne
- O(n) pojedyncze wykonanie się pętli iterującej po elementach listy
- O(n * log n) algorytm szybkiego sortowania
- O(n^c) sortowanie bąbelkowe; pętla w pętli
- O(cⁿ) rekurencyjny ciąg Fibonacciego
- O(n!) problemy związane z wyznaczaniem permutacyjnym (problem komiwojażera)



- Wyobraźmy sobie posortowaną listę liczb naturalnych. W jaki sposób możemy sprawdzić czy poszukiwana liczba x znajduje się w naszej liście?
- Pomysł 1: iteracja po elementach listy i porównywanie ich z liczbą x.
- Brzmi okej, ale w przypadku kiedy będzie to ostatni element, a lista będzie olbrzymia, stracimy mnóstwo czasu.
- Z tego samego powodu problematyczne jest poszukiwanie od końca czy od środka.
- Złożoność takiego rozwiązania wynosi O(n) w najgorszym przypadku prześledzimy całą listę (np. kiedy ostatni element będzie tym poszukiwanym albo danej liczby nie będzie w liście wcale).
- **Pomysł 2:** zmiana struktury danych z listy na np. kopiec, drzewo, graf.
- Dla dużych danych to może być dobre rozwiązanie, natomiast w przypadku małej ilości elementów niepotrzebnie komplikujemy kod.

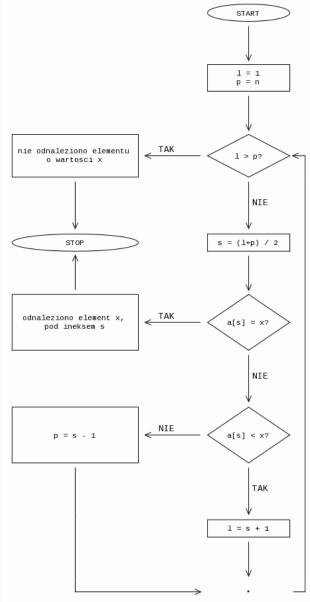


- Rozwiązanie: zastosowanie algorytmu wyszukiwania binarnego (search binary) złożoność: O(log n).
- Pamiętamy lista jest już wstępnie posortowana.
- Żeby jak najszybciej upewnić się, że w liście znajduje się liczba 7, sprawdzamy element znajdujący się
 na środku listy, jeżeli jest on mniejszy od liczby 7, będziemy jej poszukiwać w prawej połówce listy, jeżeli
 większy, w lewej.
- Za każdym razem dzielimy podlisty na pół, połowę wybieramy na podstawie relacji mniejszy/większy między aktualnym elementem a poszukiwaną liczbą (u nas: 7).
- Wykonujemy do momentu odnalezienie poszukiwanej liczby (albo do chwili, kiedy nie będzie już elementów do przeszukiwania – podzielimy listę tak bardzo, że zostanie nam tylko podlista jednoelementowa).





Zaimplementuj algorytm wyszukiwania binarnego na podstawie schematu blokowego.



I i p oznaczają tutaj pierwszy i ostatni indeks w liście (indeks = pozycja, nie wartość na danej pozycji)



- W Pythonie istnieje biblioteka bisect, która implementuje alogrytm wyszukiwania binarnego.
- Funkcja bisect z tego modułu oblicza i wskazuje index, pod jakim powinno się wstawić liczbę podaną jako argument funkcji do listy (również podanej jako argument) celem zachowania posortowanej tablicy.
- Istnieją odmiany funkcji bisect: bisect_right oraz bisect_left.
- bisect_right wskazuje najwyższy możliwy indeks dla nowej liczby (w przypadku powtórzeń tejże w tablicy), natomiast bisect_left najniższy.
- Każda z tych funkcji posiada dwa domyślne argumenty: lo (od lowest index – ten od któego zaczynamy rozpatrywać listę) oraz hi (od highest index – do którego rozpatrujemy)

```
In [1]: import bisect
In [2]: nums = [1,2,2,2,4,7,9]
In [3]: bisect.bisect(nums, 3)
Out[3]: 4
  [4]: bisect.bisect_right(nums, 2)
Out[4]: 4
In [5]: bisect.bisect_left(nums, 2)
Out[5]: 1
In [6]: bisect.bisect(nums, 12)
```



- Zamiast tylko wskazywać indeks, pod którym możemy włożyć liczbę do listy z zachowaniem kolejności, możemy wykonać operację wstawienia.
- Do tego służy funkcja insort (lub insort_right, insort_left), również z modułu bisect.
- Ekwiwalentną operacją dla insort jest nums.insert(bisect.bisect_left(nums, number), number).
 Metoda insert (wchodząca w skład każdej listy) umieszcza jej drugi argument pod indeksem podanym jako pierwszy argument na liście (tutaj: nums).
- Funkcja insort podobnie jak bisect zawiera dwa domyślne argumenty: lo oraz hi.

```
[13]: nums
Out[13]: [1, 2, 2, 2, 4, 7, 9]
In [14]: bisect.insort(nums, 3)
In [15]: nums
Out[15]: [1, 2, 2, 2, 3, 4, 7, 9]
In [16]: bisect.insort(nums, 10)
In [17]: nums
Out[17]: [1, 2, 2, 2, 3, 4, 7, 9, 10]
```



- W jaki sposób można wykorzystać bibliotekę bisect do wyszukiwania obecności danej liczby w liście?
- Zwykłe wywołanie funkcji bisect z modułu bisect powoduje wskazanie następnego indeksu, gdzie powinno się wstawić pewną liczbę z zachowaniem odpowiedniej kolejności.
- Do realizacji zadania będzie potrzeba implementacji własnej funkcji.

```
[n [8]: find([1,4,6,7,8,9,9,9], 6)
Out[8]: 2
In [9]: find([1,4,6,7,8,9,9,9], 9)
ut[9]: 5
In [10]: find([1,4,6,7,8,9,9,9], 7)
 ut[10]: 3
In [11]: find([1,4,6,7,8,9,9,9], 11)
                                          Traceback (most recent call last)
/alueError
ipython-input-11-814c73e52927> in <module>()
---> 1 find([1,4,6,7,8,9,9,9], 11)
(ipython-input-7-4e972c9b5c9f> in find(nums, num)
           if i != len(nums) and nums[i] == num:
                return i
           raise ValueError
/alueError:
```



Co jeżeli dane nie będą poukładane po kolei?





- Według Wikipedii sortowanie to "jeden z podstawowych problemów informatyki, polegający na uporządkowaniu zbioru danych względem pewnych cech charakterystycznych każdego elementu tego zbioru".
- W algorytmice chodzi przede wszystkim o prostotę algorytmu, łatwość jego implementacji oraz efektywność.
- Powstało mnóstwo algorytmów sortujących zgromadzone dane:
- sortowanie bąbelkowe,
- sortowanie przez wstawianie,
- sortowanie przez scalanie,
- szybkie sortowanie,
- sortowanie przez wybieranie
- sortowanie przez zliczanie,
- sortowanie kubełkowe,
- sortowanie pozycyjne,
- sortowanie biblioteczne,
- i wiele innych...



 W Pythonie mamy już zaimplementowany algorytm sortowania, który jest wołany ilekroć korzystamy z funkcji sorted lub metody sort. Ta pierwsza zwraca nową, posortowaną listę, druga sortuje listę w miejscu.

- Algorytmem używanym w Pythonie do sortowania w funkcjach wbudowanych jest tzw. algorytm Timsort – jest to połączenie algorytmów sortowania przez scalanie oraz sortowania przez wstawianie.
- Więcej na temat algorytmu można przeczytać tutaj.

```
[13]: numbers = [4, 5, 2, 0, 6, 9]
In [14]: sorted(numbers)
ut[14]: [0, 2, 4, 5, 6, 9]
in [15]: numbers
 ıt[15]: [4, 5, 2, 0, 6, 9]
  [16]: numbers.sort()
  [17]: numbers
 ıt[17]: [0, 2, 4, 5, 6, 9]
```

Sortowanie bąbelkowe – bubble sort



Najprostszy algorytm sortowania danych. Polega na porównywaniu dwóch kolejnych elementów listy i podmianie ich, jeżeli pierwsza jest większa od drugiej (w przypadku sortowania rosnącego). Porównywanie wykonuje się od początku do końca tablicy w dwóch pętlach. Algorytm kończy się, kiedy w którejś iteracji z kolei nie zostanie podmieniona żadna liczba.

Złożoność obliczeniowa: O(n^2)

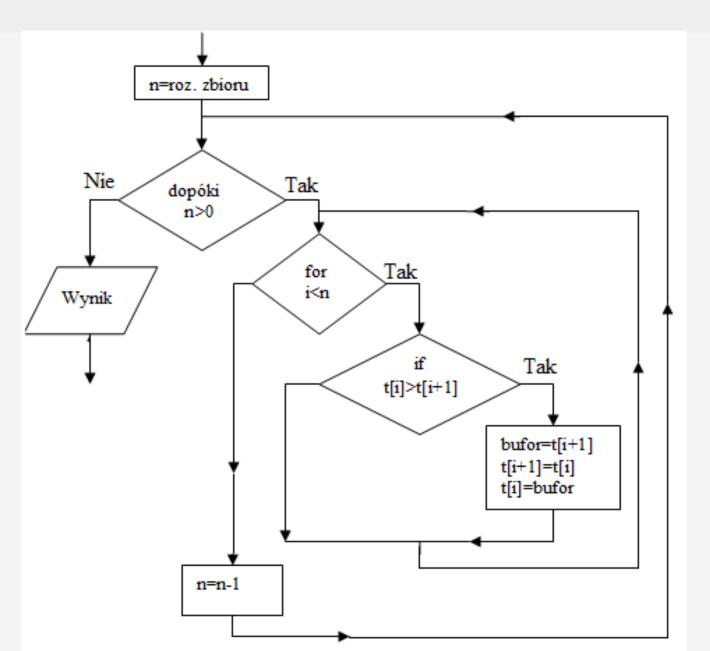
Złożoność pamięciowa: O(1)

Przedstawienie działania algorytmu na animacji:

LINK

Sortowanie bąbelkowe – bubble sort





n oznacza tutaj rozmiar listy, ale taki, by można było ją indeksować za pomocą liczb (0, n), stąd n = len(lista)-1; n to maksymalny indeks w liście

Sortowanie bąbelkowe – bubble sort



Napisz funkcję **bubble_sort**, która przyjmować będzie listę liczb naturalnych i w "miejscu" posortuje ją rosnąco. Należy skorzystać z algorytmu sortowania bąbelkowego.

Sortowanie przez wstawianie – insertion sort



Jeden z najprostszych algorytmów sortujących. Jego działanie jest podobne do sposobu w jaki ludzie układają sobie na ręce karty do gry. Algorytm polega na porównywaniu nowego elementu z danymi w posortowanej liście do momentu, kiedy nie napotka się elementu mniejszego lub równego (w przypadku kolejności rosnącej).

Złożoność obliczeniowa: O(n^2)

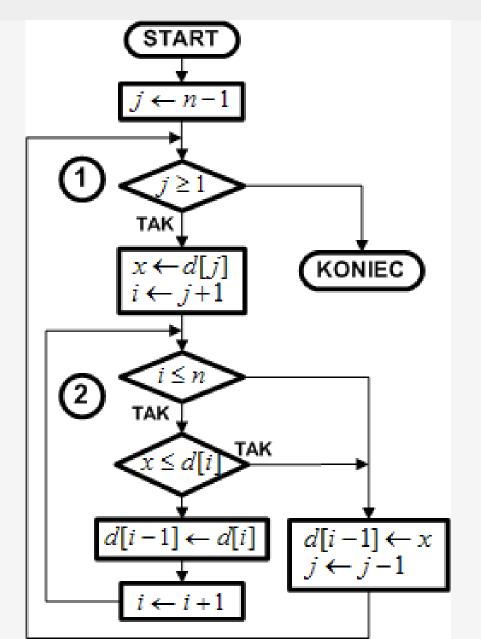
Złożoność pamięciowa: O(1)

Przedstawienie działania algorytmu na animacji:

LINK

Sortowanie przez wstawianie – insertion sort





n oznacza tutaj ostatnią pozycję, ostatni element – w około programistycznych rozważaniach można to uznać za to samo, natomiast podczas pisania kodu należy pamiętać, że ostatnią pozycją nie jest długość listy, tylko długość listy-1, stąd n = len(lista)-1

j >= 1 oznacza nic innego jak pierwszą pozycję, czyli pozycję 0.

Sortowanie przez wstawianie – insertion sort



Napisz funkcję insertion_sort, która przyjmować będzie listę liter (stringów) od A-Z, a następnie posortuje ją w miejscu rosnąco, jeżeli wartość drugiego parametru reversed będzie ustawiona na False albo malejąco, jeżeli argument reversed będzie ustawiony na True. Należy skorzystać z algorytmu sortowania przez wstawianie.

Sortowanie przez scalanie – merge sort



Algorytm sortujący w sposób rekurencyjny. Działanie polega na rozbiciu listy na mniejsze podlisty, posortowaniu podlist i ponowne scalenie ich w jedną, pełną listę.

Złożoność obliczeniowa: O(n * log n)

Złożoność pamięciowa: O(n)

Przedstawienie działania algorytmu na animacji:

<u>LINK</u>

Sortowanie przez wybór – selection sort



Prosty algorytm sortujący polegający na wyszukiwaniu najmniejszej wartości w zbiorze i umieszczeniu jej na pozycji pierwszej. Następnym krokiem jest wyszukanie najmniejszej wartości w pomniejszonej liście (już bez tego pierwszego, ustawionego na właściwej pozycji elementu) i ustawieniu go na pozycji drugiej listy wejściowej. I tak do końca tablicy.

Złożoność obliczeniowa: O(n²)

Złożoność pamięciowa: O(1)

Przedstawienie działania algorytmu na animacji:

LINK

Sortowanie szybkie – quick sort



Jeden z najszybszych algorytmów sortujących. Polega na wybraniu elementu rozdzielającego z listy (tzw. pivota) oraz podzieleniu listy na dwie części: w pierwszej znajdą się liczby mniejsze, a w drugiej większe od elementu rozdzielającego. Potem sortuje się rekurencyjnie tak podzielone listy. Rekursja zakończy się, gdy uzyskane zostaną listy jednoelementowe.

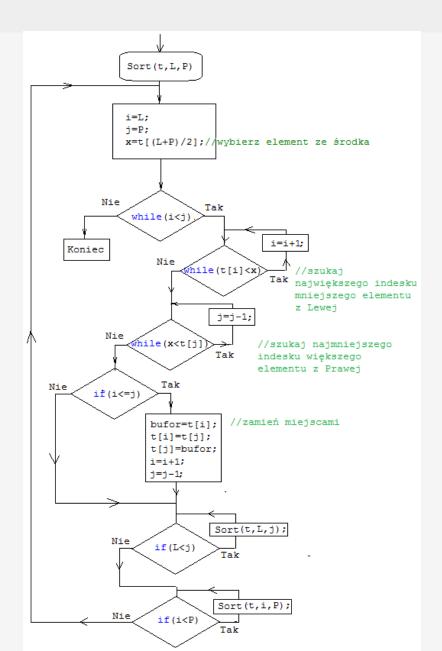
Złożoność obliczeniowa: O(n * log n)

Przedstawienie działania algorytmu na animacji:

<u>LINK</u>

Sortowanie szybkie – quick sort





Sortowanie szybkie – quick sort



Napisz funkcję quick_sort, która implementować będzie algorytm sortowania szybkiego.



Ciekawostką może się okazać jednoczesne śledzenie postępów działania dopiero co poznanych algorytmów sortowania.

Który jest najszybszy?

LINK