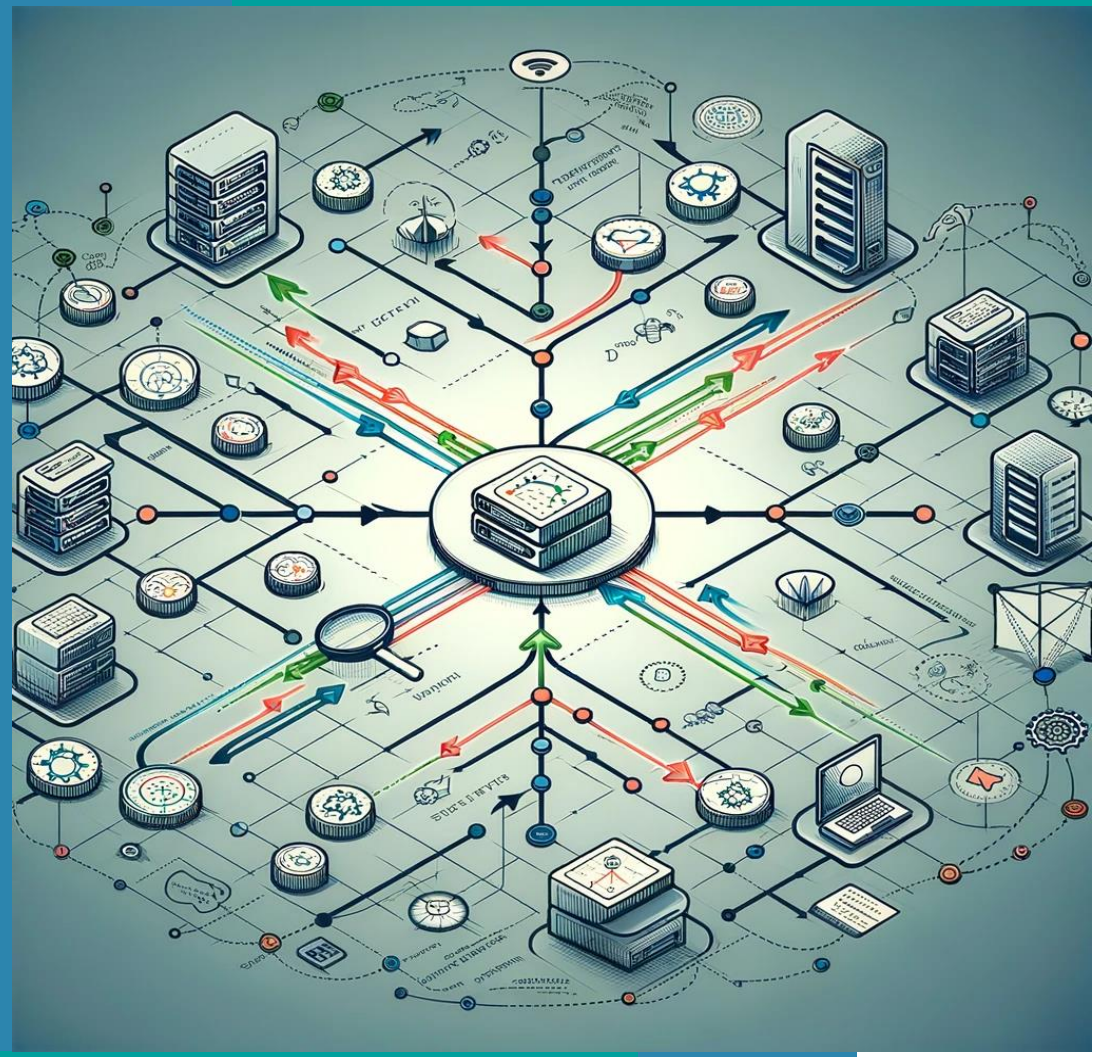
dr inż. Michał Malinowski
michal.malinowski@uth.edu.pl

https://www.linkedin.com/in/michał-malinowski-malkon/

Rekurencja

Ćwiczenie 11

ALGORYTMY
I STRUKTURY DANYCH



Zagadnienia (2h)

- Rekurencja
- Algorytm obliczania silni
  - Zadanie: Algorytm obliczania silni
- Algorytm Euklidesa
  - Zadanie: Algorytm Euklidesa
- Sortowanie szybkie
  - Zadanie: Sortowanie szybkie

Rekurencja

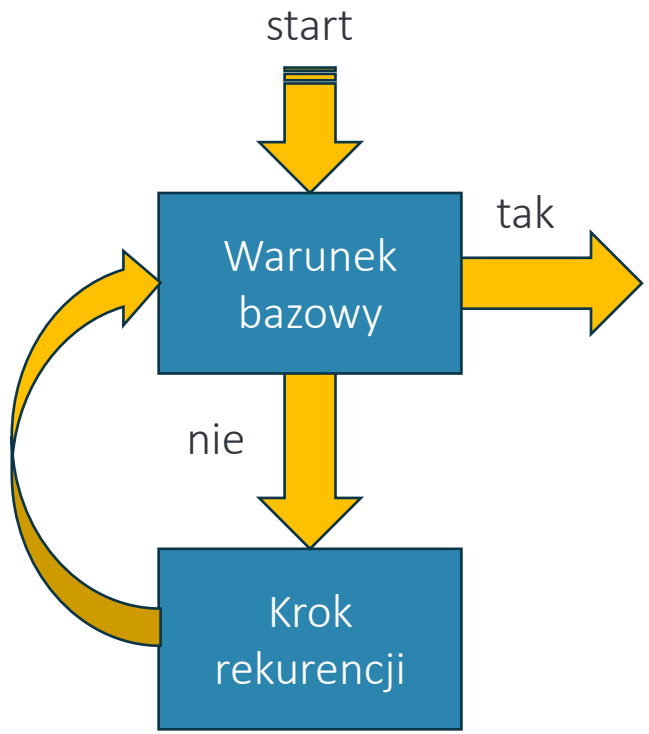
Rekurencja

to metoda w programowaniu, gdzie funkcja wywołuje samą siebie bezpośrednio lub pośrednio, aby rozwiązać problem.

Rekurencyjne rozwiązanie zwykle zawiera dwa główne elementy:

- warunek bazowy, który zatrzymuje rekurencję,
- krok rekurencyjny, który dzieli problem na mniejsze części i wywołuje samą siebie.

```
f_rekurencyjna(parametr) {
        if  (warunek_bazowy(parametr))
                { return    wynik_dla_warunku_bazowego }
        else
                { return f_rekurencyjna(zmieniony_parametr) }
}
```



Algorytm obliczania silni

Algorytm obliczania silni

to metoda obliczeniowa służąca do znalezienia wartości silni dla danej liczby całkowitej nieujemnej.

Silnia liczby n, oznaczona jako $n!$, to iloczyn wszystkich liczb naturalnych od 1 do n. Definiuje się również, że $0! = 1$.

Jest to algorytm rekurencyjny.

Algorytm obliczania silni

Złożoność obliczeniowa

Złożoność obliczeniowa rekurencyjnego algorytmu obliczenia silni dla liczby n jest $O(n)$. Oznacza to, że czas potrzebny na obliczenie silni wzrasta liniowo w stosunku do wartości n. Każde wywołanie rekurencyjne funkcji silni wykonuje stałą liczbę operacji i wywołuje się samo dla $n - 1$, aż do osiągnięcia warunku bazowego, którym jest $n = 0$.

Kroki algorytmu obliczania silni

Krok startowy	Rozpocznij algorytm z daną liczbą całkowitą nieujemną n, dla której chcesz obliczyć silnię.
Warunek bazowy	Sprawdź, czy n jest równa 0. Jeśli tak, zwróć 1, ponieważ silnia z 0 wynosi 1.
Krok rekurencyjny	Jeśli n jest większe od 0, wywołaj funkcję silni rekurencyjnie z argumentem $n - 1$ i pomnóż wynik przez nn.
Powrót z rekurencji	Czekaj na wynik wywołania rekurencyjnego, a następnie pomnóż wynik przez n.
Zakończenie algorytmu	Po otrzymaniu wyniku z ostatniego rekurencyjnego wywołania, zwróć wynik jako silnię z n.

Zadanie: Algorytm obliczania silni

Zadanie

Napisz kod JavaScript realizujący algorytm obliczania silni.

Liczba: 9

Zadanie: Algorytm obliczania silni

Rozwiązanie

```
function factorial(n) {
    if (n === 0) {
        return 1;
    }
    return n * factorial(n - 1);
}

const number = 9;
const result = factorial(number);
console.log(`Silnia liczby ${number} wynosi: ${result}`);
```

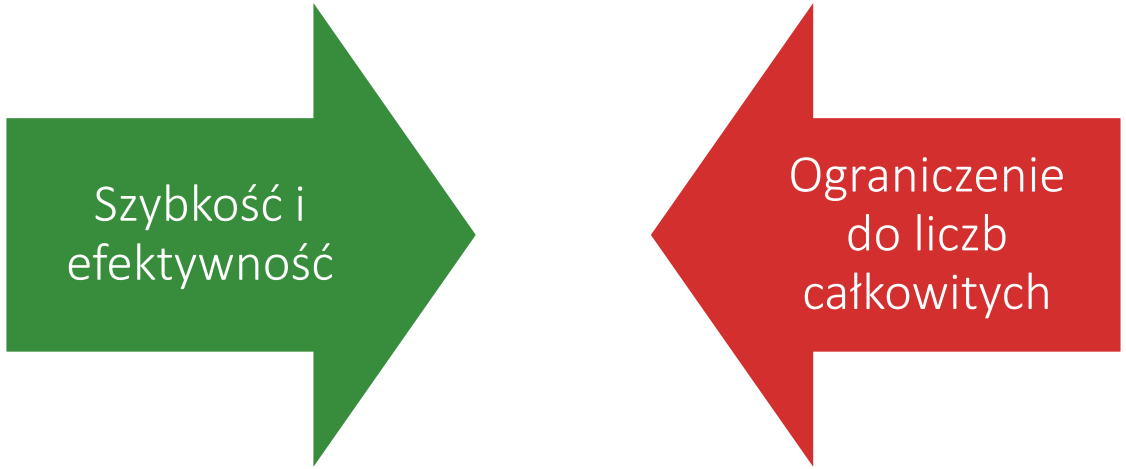
Algorytm Euklidesa

Algorytm Euklidesa

to jedna z najstarszych znanych metod obliczania Największego Wspólnego Dzielnika (NWD) dwóch liczb całkowitych. Jest to podstawowa procedura, która była znana już w starożytnej Grecji i została opisana przez Euklidesa w jego dziele "Elementy" około 300 r. p.n.e.

Algorytm ten opiera się na prostym założeniu, że NWD dwóch liczb nie zmienia się, gdy z większej liczby odejmiemy mniejszą.

Jest to algorytm rekurencyjny.



Algorytm Euklidesa

Złożoność obliczeniowa

w podstawowej formie algorytmu nie jest stała i zależy od wielkości i wzajemnego stosunku przetwarzanych liczb.

W najgorszym przypadku, który występuje, gdy liczby są kolejnymi liczbami Fibonacciego, liczba kroków potrzebna do znalezienia NWD jest proporcjonalna do liczby cyfr mniejszej z liczb.

Można to wyrazić jako $O(\log \min(a, b))$,

Gdzie a i b to liczby, dla których obliczamy NWD

Kroki algorytmu Euklidesa

Krok startowy	Rozpocznij algorytm z dwoma liczbami a i b, dla których chcesz znaleźć NWD.
Sprawdzenie warunku bazowego	Jeżeli $b = 0$, to algorytm osiągnął warunek końcowy, wówczas a jest NWD (ponieważ NWD$(a, 0) = a$), więc zwróć a jako wynik.
	W przeciwnym wypadku, przejdź do kroku rekurencyjnego.
Krok rekurencyjny	Wywołaj funkcję algorytmu Euklidesa rekurencyjnie z b jako nowym a, a resztą z dzielenia a przez b (czyli $a\%b$) jako nowym b.
Powrót z rekurencji	Rekurencyjne wywołania będą kontynuowane, aż do osiągnięcia warunku bazowego, czyli do momentu, gdy drugi argument będzie równy 0.
	W rezultacie ostatnie niezerowe b zostanie zwrócone jako NWD liczb a i b.
Zakończenie algorytmu	Po zakończeniu rekurencji, ostateczny wynik (ostatnie niezerowe b) jest NWD liczb a i b.

Zadanie: Algorytm Euklidesa

Zadanie

Napisz kod JavaScript realizujący algorytm Euklidesa dla dwóch liczb całkowitych.

Liczby: 56, 98

Zadanie: Algorytm Euklidesa

Rozwiązanie

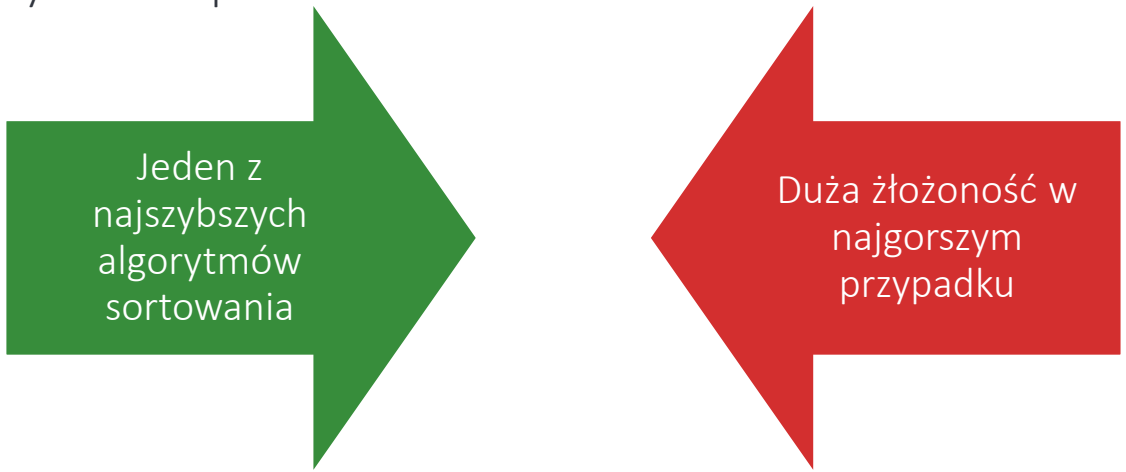
```
function gcd(a, b) {
  if (b === 0) {
    return a;
  } else {
    return gcd(b, a % b);
  }
}

// Przykładowe liczby do obliczenia NWD
const num1 = 56;
const num2 = 98;

console.log(`NWD(${num1}, ${num2}) =`, gcd(num1, num2));
```

Sortowanie szybkie

Sortowanie szybkie (Quick Sort)

to popularny algorytm sortowania, który działa na zasadzie "dziel i zwyciężaj". Jego efektywność wynika z wyboru elementu osiowego (zwanego pivotem) i podziału zbioru na dwie części: jedną zawierającą elementy mniejsze od pivotu, a drugą zawierającą elementy większe lub równe pivotowi.

Proces ten jest następnie rekurencyjnie powtarzany dla obu podzbiorów.



Sortowanie szybkie

Złożoność obliczeniowa

podobnie jak innych algorytmów sortowania typu "dziel i zwyciężaj", ma charakterystykę czasową zależną od kilku czynników, takich jak wybór pivotu i charakter danych wejściowych. W przypadku optymistycznym i średnim, gdy podział danych jest w miarę równomierny, złożoność czasowa wynosi $O(\mathrm{n}\log n)$, co oznacza, że czas potrzebny do wykonania algorytmu rośnie logarytmicznie wraz ze wzrostem liczby elementów do posortowania (n).

Kroki sortowania szybkiego

Krok startowy	Rozpocznij z całą tablicą lub podtablicą, którą chcesz posortować.
Wybór pivotu	Wybierz element osiowy (pivot) z tablicy. Może to być pierwszy element, ostatni, środkowy, losowy element lub mediana trzech.
Partycjonowanie	Przeprowadź proces partycjonowania, który polega na przeniesieniu wszystkich elementów mniejszych od pivotu na jego lewą stronę, a wszystkich większych lub równych na prawą.
Rekurencyjne sortowanie lewej podtablicy	Zastosuj algorytm Quick Sort rekurencyjnie do lewej części tablicy, która zawiera elementy mniejsze od pivotu.
Rekurencyjne sortowanie prawej podtablicy	Zastosuj algorytm Quick Sort rekurencyjnie do prawej części tablicy, która zawiera elementy większe lub równe pivotowi.
Warunek bazowy rekurencji	Rekurencja kończy się, gdy podtablica ma zero lub jeden element, ponieważ jest już posortowana.
Scalanie	W Quick Sort krok scalania jest niepotrzebny, ponieważ wszystkie operacje wykonuje się na oryginalnej tablicy i jest ona sortowana "in-place".

Zadanie: Sortowanie Szybkie

Zadanie

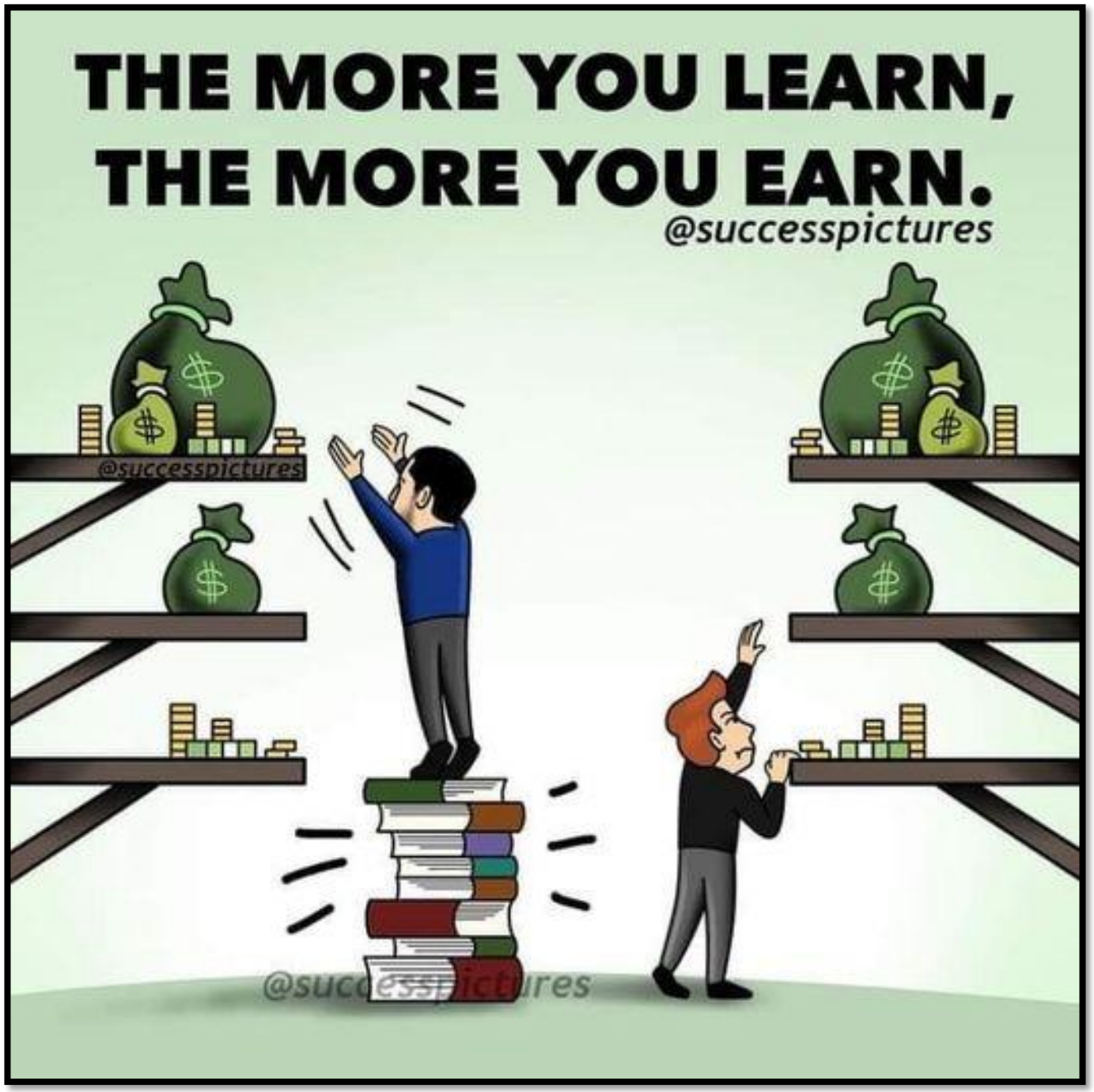
Napisz kod JavaScript realizujący sortowania szybkiego nieposortowanej tablicy.

Tablica: [9, -3, 5, 2, 6, 8, -6, 1, 3]

Zadanie: Sortowanie Szybkie

Rozwiązanie

```
function quickSort(arr) {
    if (arr.length <= 1) {
        return arr;
    }

    const pivot = arr[arr.length - 1];
    const leftArr = [];
    const rightArr = [];

    for (const el of arr.slice(0, arr.length - 1)) {
        el < pivot ? leftArr.push(el) : rightArr.push(el);
    }

    return [...quickSort(leftArr), pivot, ...quickSort(rightArr)];
}

// Przykładowa tablica do posortowania
const exampleArray = [9, -3, 5, 2, 6, 8, -6, 1, 3];

// Wywołanie sortowania i wydrukowanie wyniku
const sortedArray = quickSort(exampleArray);
console.log(sortedArray);
```


THE MORE YOU LEARN,
THE MORE YOU EARN.
@successpictures
@successpictures
@successpictures