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

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

Algorytmy wyszukiwania

Ćwiczenie 10

ALGORYTMY
I STRUKTURY DANYCH

Zagadnienia (2h)

- Wyszukiwanie linowe
  - Zadanie: Wyszukiwanie linowe
- Wyszukiwanie binarne
  - Zadanie: Wyszukiwanie binarne
- Metoda bisekcji
  - Zadanie: Metoda bisekcji

Podział języków programowania

Kryterium	Kategoria	Przykłady języków
Poziom abstrakcji	Języki niskiego poziomu	Asembler, język maszynowy
	Języki wysokiego poziomu	C, Java, Python
Paradygmaty programowania	Proceduralne	C, Fortran
	Obiektowe	Java, C++, Python
	Funkcyjne	Haskell, Lisp
	Logiki	Prolog
	Skryptowe	JavaScript, Python, Ruby
Sposób wykonywania	Kompilowane	C, C++, Rust
	Interpretowane	Python, Ruby, JavaScript
	JIT (Just-In-Time)	Java (JVM), C# (CLR)
Zastosowanie	Ogólnego przeznaczenia	Python, Java, C++
	Specjalistyczne	MATLAB, SQL, R
Składnia	C-like	C++, Java, C#
	Pascal-like	Delphi, Ada
	ML-like	Haskell, F#
Inne kryteria	Deklaratywne	SQL, HTML
	Imperatywne	C, Java
	Statycznie typowane	Java, C++
	Dynamicznie typowane	Python, JavaScript

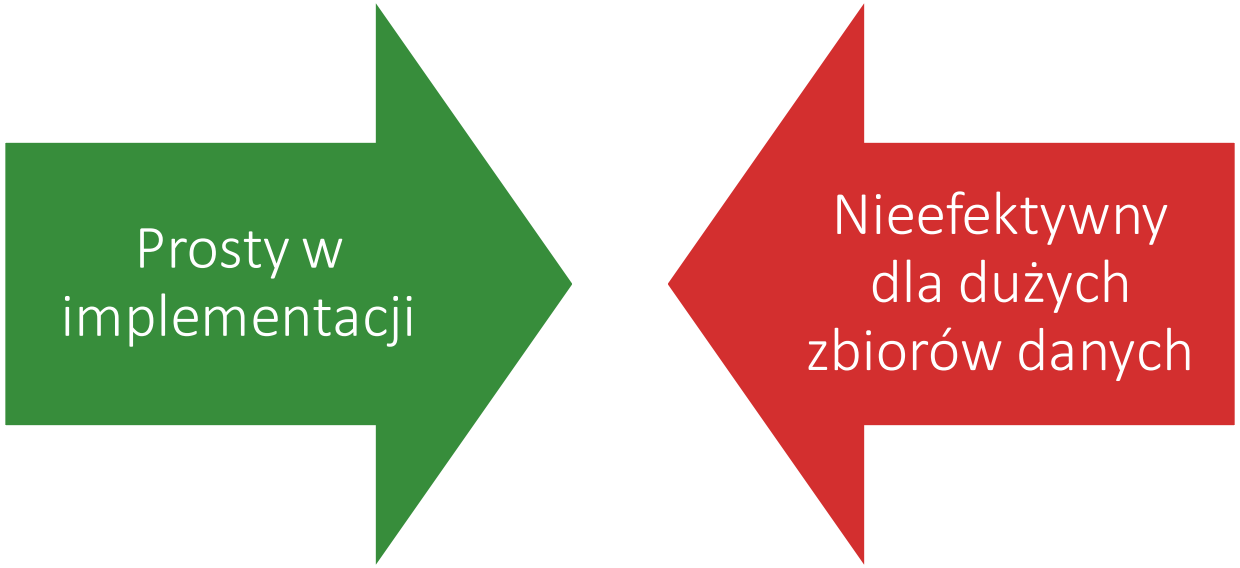
Podział języków programowania

Język	Kategorie
JavaScript	Język wysokiego poziomu, Proceduralne, Obiektowe, Funkcyjne, Interpretowane, JIT (V8 Engine), Ogólnego przeznaczenia, C-like, Imperatywne (z cechami deklaratywnymi), Dynamicznie typowane
Python	Język wysokiego poziomu, Proceduralne, Obiektowe, Funkcyjne, Interpretowane, JIT (PyPy), Ogólnego przeznaczenia, Unikalna składnia, Imperatywne, Dynamicznie typowane
SQL	Język wysokiego poziomu, Deklaratywne, Interpretowane, Specjalistyczne (bazy danych)
HTML	Język wysokiego poziomu, Deklaratywne, Statyczne, Specjalistyczne (tworzenie stron internetowych)

Wyszukiwanie liniowe

Wyszukiwanie liniowe (linear search) sekwencyjne (sequential search)

to podstawowy algorytm wyszukiwania, który polega na przeszukiwaniu sekwencji elementów jeden po drugim, aż do znalezienia szukanego elementu lub dojścia do końca sekwencji.



Wyszukiwanie liniowe

Złożoność obliczeniowa

W najgorszym przypadku, gdy element nie występuje w zbiorze lub jest ostatnim elementem, wynosi $O(n)$, gdzie n to liczba elementów w zbiorze danych. Oznacza to, że czas działania algorytmu wzrasta liniowo wraz ze wzrostem rozmiaru danych wejściowych.
W najlepszym przypadku, gdy szukany element jest pierwszym elementem zbioru danych, złożoność czasowa wynosi $O(1)$, co oznacza, że algorytm znajdzie element natychmiast, niezależnie od rozmiaru zbioru danych.

Średnia złożoność obliczeniowa wynosi $O(n/2)$, ale w analizie złożoności algorytmów stałe są pomijane $(1/2)$, więc także przyjmuje się $O(n)$.

Zauważa się zatem, że średni i najgorszy przypadek mają tę samą złożoność obliczeniową $O(n)$

Gdzie n to to liczba elementów w zbiorze

Kroki wyszukiwania liniowego

Start od pierwszego elementu	Rozpocznij przeszukiwanie sekwencji danych od pierwszego elementu.
Porównaj szukany element z bieżącym	W każdym kroku porównaj szukany element z bieżącym elementem w sekwencji.
Sprawdź, czy to szukany element	Jeśli element sekwencji jest równy szukanemu elementowi, zakończ wyszukiwanie i zwróć indeks tego elementu.
Przejście do następnego elementu	Jeśli element sekwencji nie jest równy szukanemu elementowi, przejdź do następnego elementu w sekwencji.
Powtarzaj, aż do końca	Kontynuuj proces aż do osiągnięcia końca sekwencji.
Zwróć wynik	Jeśli szukany element zostanie znaleziony, zwróć jego indeks. Jeśli koniec sekwencji zostanie osiągnięty, a szukany element nie zostanie znaleziony, zwróć informację o niepowodzeniu (często jest to wartość specjalna jak np. -1, wskazująca, że element nie istnieje w sekwencji).

Zadanie: Wyszukiwanie linowe

Zadanie

Napisz kod JavaScript realizujący wyszukiwanie liniowe dla zbioru danych.

Zbiór do przeszukania: [84, 25, 61, 40, 48, 9, 50, 91, 67, 4, 97, 60, 73, 94, 56, 10, 47, 44, 81, 42]

a) Znajdź: $[97]$

b) Znajdź: $[99]$

c) Znajdź: $[94, 56]$

Zadanie: Wyszukiwanie linowe

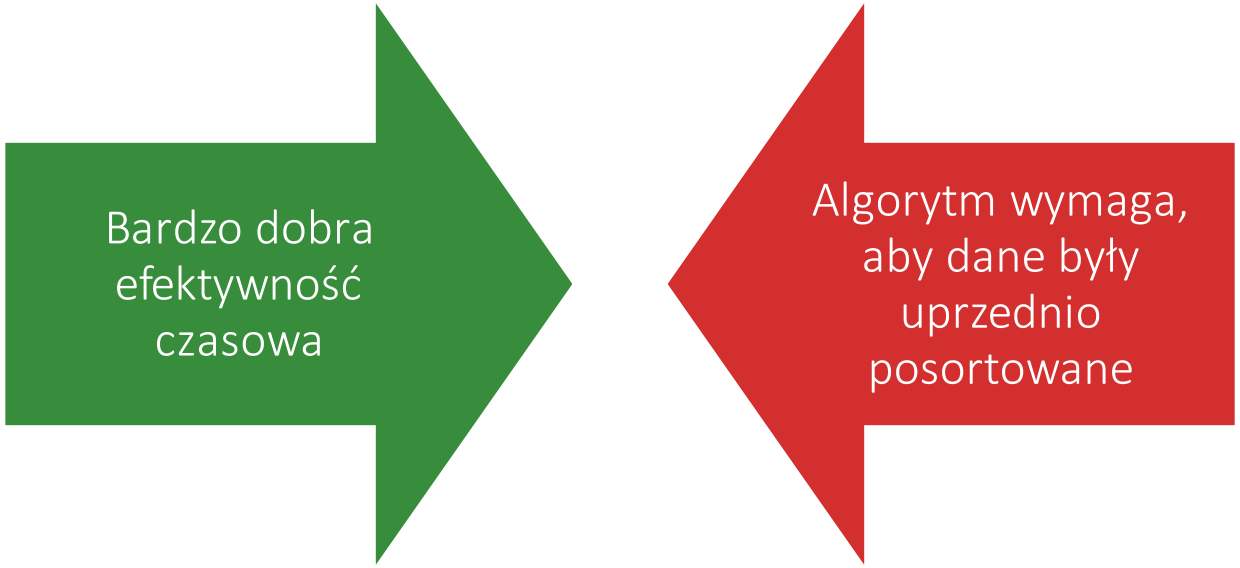
Rozwiązanie

```javascript
function linearSearch(arr, target) {
    for (let i = 0; i < arr.length; i++) {
        if (arr[i] === target) {
            return i;
        }
    }
    return -1;
}

const array = [5, 3, 7, 1, 2, 8, 4, 6];
const target = 7;
const index = linearSearch(array, target);

if (index !== -1) {
    console.log(`Element znaleziony na pozycji: ${index}`);
} else {
    console.log("Element nie znaleziony w tablicy.");
}
```

Wyszukiwanie binarne

Wyszukiwanie binarne (Binary Search)

to algorytm wyszukiwania, który działa na posortowanych listach. Dzieli listę na pół przy każdym kroku, znacznie redukując liczbę porównań.
Bazuje na koncepcji dziel i zwyciężaj (divide and conquer).



Wyszukiwanie binarne

Złożoność obliczeniowa

liczba kroków potrzebnych do znalezienia elementu lub stwierdzenia jego braku rośnie logarytmicznie w stosunku do rozmiaru danych. Dzięki temu, wyszukiwanie binarne jest znacznie szybsze niż wyszukiwanie liniowe, szczególnie w przypadku dużych zbiorów danych.

$O(\log n)$

Gdzie n to liczba postartowanych elementów

Kroki wyszukiwania binarnego

Inicjalizacja indeksów	Ustal lewy indeks low na początku sekwencji (zazwyczaj 0).
	Ustal prawy indeks high na końcu sekwencji (zazwyczaj n-1, gdzie n to liczba elementów).
Wykonaj pętlę dopóki lewy indeks jest mniejszy lub równy prawemu	Oblicz środkowy indeks mid jako średnią low i high (zwykle (low + high) / 2).
	Sprawdź, czy element w środkowym indeksie jest szukanym elementem. Jeśli tak, zwróć indeks mid.
Porównanie i zmiana zakresu	Jeśli szukany element jest mniejszy niż element w środkowym indeksie, zmień indeks high na mid - 1.
	Jeśli szukany element jest większy niż element w środkowym indeksie, zmień indeks low na mid + 1.
Powtórz lub zakończ	Kontynuuj wykonanie pętli, dopóki indeksy się nie przekroczą.
	Jeśli zakres się zmniejszy do punktu, gdzie low jest większe niż high, oznacza to, że szukany element nie znajduje się w sekwencji. W tym przypadku zwróć informację o nieznalezieniu elementu (często jest to wartość specjalna, jak -1).
Zakończenie algorytmu	Zwróć pozycję znalezionego elementu lub odpowiednią wartość wskazującą, że element nie występuje w sekwencji.

Zadanie: Wyszukiwanie binarne

Zadanie

Napisz kod JavaScript realizujący wyszukiwanie binarne dla posortowanego zbioru danych.

Zbiór do przeszukania: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Znajdź: [5]

Znajdź: [11]

Zadanie: Wyszukiwanie binarne

Rozwiązanie

```javascript
function binarySearch(arr, target) {
    let left = 0;
    let right = arr.length - 1;

    while (left <= right) {
        let mid = Math.floor((left + right) / 2);

        if (arr[mid] === target) {
            return mid; // element znaleziony
        }
        if (arr[mid] < target) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }

    return -1; // element nie znaleziony
}
```

```javascript
// Przykładowe użycie:
const array = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
const target = 5;
const index = binarySearch(array, target);

if (index !== -1) {
    console.log(`Element znaleziony na pozycji: ${index}`);
} else {
    console.log("Element nie znaleziony w tablicy.");
}
```

Metoda bisekcji

Metoda bisekcji

znana również jako metoda podziału na pół, jest prostą, lecz potężną techniką numeryczną służącą do znajdowania pierwiastków równań nieliniowych.

Metoda bisekcji jest prosta i zawsze zbiega do prawdziwego pierwiastka, o ile warunki są spełnione,

Jest bardzo stabilna i łatwa do zrozumienia oraz implementacji, co sprawia, że jest często używana jako pierwsze podejście do rozwiązywania równań nieliniowych.

Metoda bisekcji

Złożoność obliczeniowa

w metodzie bisekcji, przedział, w którym poszukujemy pierwiastka, jest dzielony na pół w każdej iteracji. Oznacza to, że zakres poszukiwań zmniejsza się o połowę za każdym razem, co daje nam złożoność logarytmiczną względem liczby kroków potrzebnych do osiągnięcia określonej dokładności.

$$O(\log(1/\epsilon))$$

Gdzie ϵ to dokładność (tolerancja)

Kroki metody bisekcji

Wybór przedziału początkowego	Aby metoda bisekcji była stosowana, potrzebny jest przedział początkowy $[a, b]$, w którym funkcja $f(x)$ zmienia znak, czyli $f(a)$ i $f(b)$ mają przeciwne znaki. To oznacza, że według twierdzenia o wartości pośredniej, istnieje przynajmniej jeden pierwiastek funkcji $f(x)$ w tym przedziale.
Sprawdzenie warunków	Musimy się upewnić, że funkcja $f(x)$ jest ciągła w przedziale $[a, b]$, co jest warunkiem koniecznym dla istnienia pierwiastka.
Podział przedziału na pół	Obliczamy wartość funkcji w środku przedziału, $c = \frac{a+b}{2}$. Następnie obliczamy wartość $f(c)$.
Decyzja o kierunku dalszego poszukiwania	Jeśli wartość $f(c)$ ma ten sam znak co $f(a)$, wówczas pierwiastek musi się znajdować w przedziale $[c, b]$, więc nowym przedziałem staje się $[c, b]$. Jeśli $f(c)$ ma ten sam znak co $f(b)$, to pierwiastek znajduje się w przedziale $[a, c]$, więc nowym przedziałem staje się $[a, c]$.
Powtarzanie procesu	Proces jest powtarzany, za każdym razem dzieląc przedział na pół i wybierając podprzedział, w którym zmienia się znak funkcji, do momentu osiągnięcia zadanej dokładności (np. różnica między a i b jest mniejsza niż zadany ϵ - epsilon) lub do osiągnięcia maksymalnej liczby iteracji.
Znalezienie przybliżonego pierwiastka	Proces kończy się, gdy różnica między a i b jest dostatecznie mała, a punkt środkowy c jest uznawany za przybliżenie pierwiastka równania $f(x) = 0$.

Zadanie: Metoda bisekcji

Zadanie (2 pkt)

Napisz kod JavaScript realizujący szukanie pierwiastka dla funkcji nieliniową metodą bisekcji.

Funkcja nieliniowa: $f(x) = x^3 - x - 2$

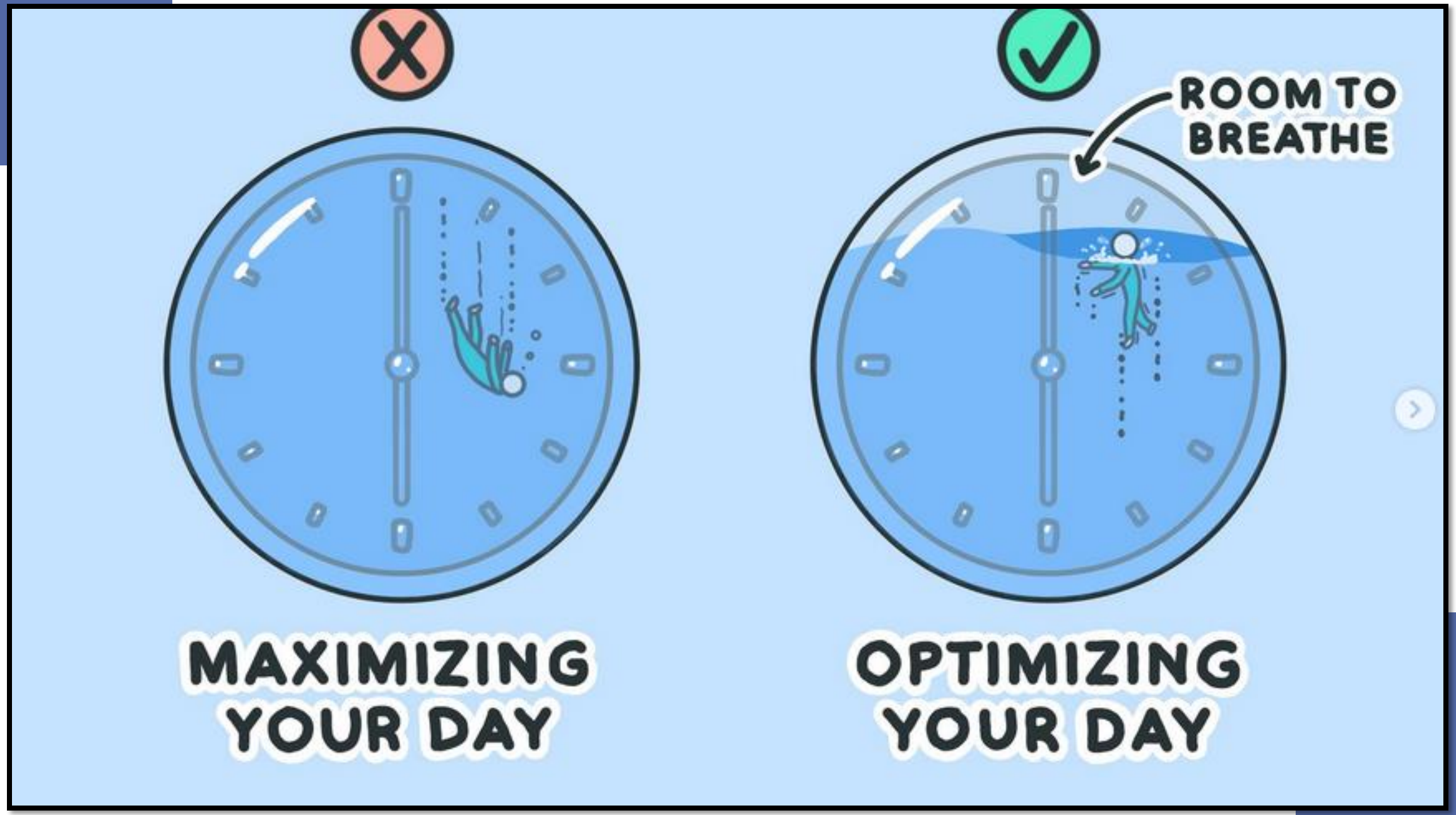
Przedział startowy: $[1,2]$

Przedstawienie wyniku

- Dokumenty HTML / ZIP
- Umieszczenie wyniku w Moodle

Termin

- Podany przy zadaniu w Moodle (około 2 tygodnie)


ROOM TO BREATHE
MAXIMIZING YOUR DAY
OPTIMIZING YOUR DAY