



Programowanie 1

Mateusz Duraj

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Agenda

- Interfejs Map w Javie i haszowanie
- Stream API i wyrażenia lambda
- Rekurencja
- Drzewa binarne
- Zaawansowane sortowanie na przykładzie algorytmów quicksort i mergesort

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Map

Mapa to struktura danych typu **klucz - wartość**.

Map<KLUCZ, WARTOŚĆ>

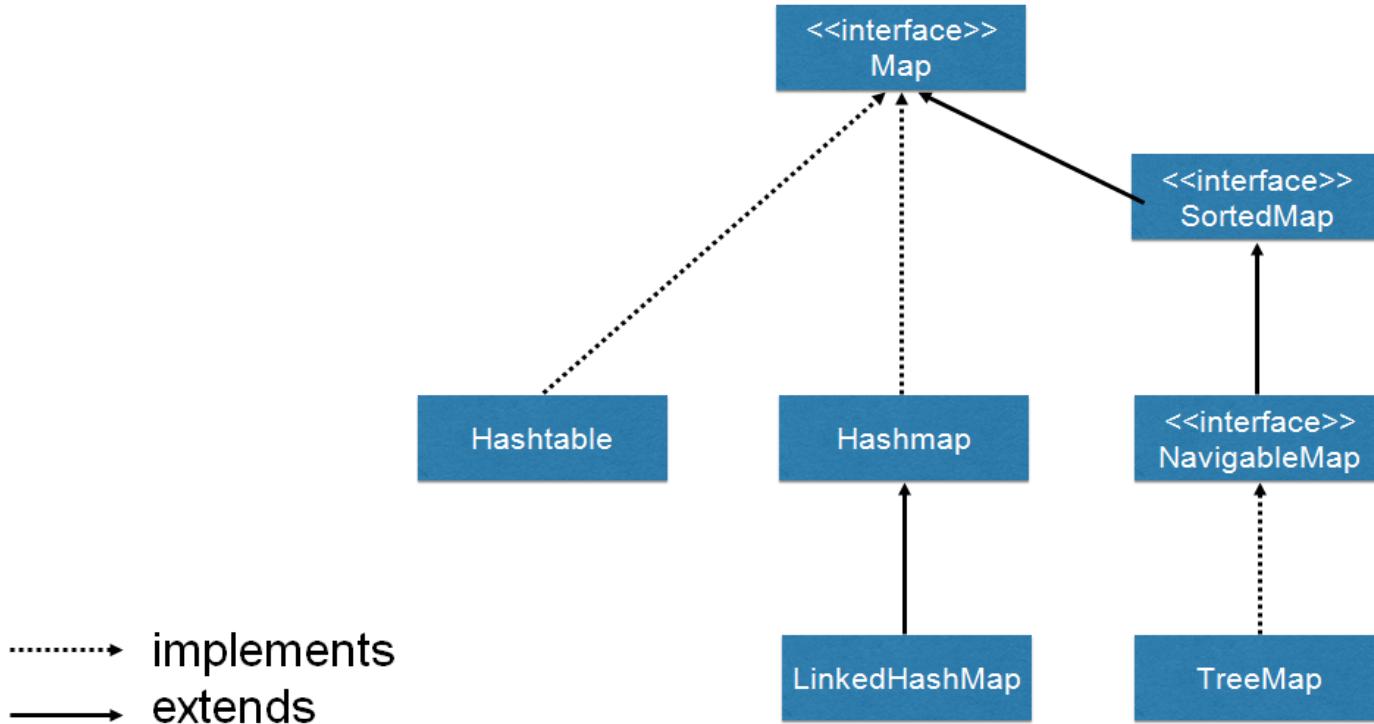
Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Haszowanie – Interfejsy java.util.Map i java.util.SortedMap

Map Interface



Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Haszowanie – Interfejsy java.util.Map i java.util.SortedMap

Interfejs `java.util.Map` zawiera wspólne funkcjonalności dla map łączących klucze z odpowiadającymi im wartościami.

Klucze przechowywane w mapie muszą być unikatowe. Dla map zdefiniowane są następujące operacje:

- dodawanie/ustawianie elementów: `put`, `putAll`
- odczytywanie elementów: `get`
- sprawdzenie, czy dany klucz lub wartość znajduje się w mapie: `containsKey`, `containsValue`
- usunięcie danego klucza: `remove`
- pobranie ilości elementów znajdujących się w mapie: `size`
- utworzenie kolekcji z kluczami, wartościami i ich parami: `keySet`, `values`, `entrySet`
- tworzone w ten sposób kolekcje są tak naprawdę widokami, korzystają z danych przechowywanych w mapie, a nie kopią ich.

Autor:



Haszowanie – Interfejsy java.util.Map i java.util.SortedMap

Interfejs Map jest rozszerzany przez interfejsy SortedMap i NavigableMap, będące dla mapy tym czym SortedSet i NavigableSet dla zbioru.

Przykładowe implementacje:

- **HashMap**: nieuporządkowana mapa
- **TreeMap**: NavigableMap, bazująca na drzewie czerwono-czarnym
- **LinkedHashMap**: jak HashMap, ale przechowuje podwójnie dowiązaną listę kluczy – umożliwia to odtworzenie porządku w jakim był wstawiane podczas iterowania.

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Złożoność obliczeniowa na mapach

Struktura	get/put	containsKey	struktura danych
HashMap	O(1)	O(1)	tablica haszująca
TreeMap	O(log n)	O(log n)	drzewo czarne
LinkedHashMap	O(1)	O(1)	tablica haszująca + lista dowiązana

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Haszowanie(ang. Haszing)

Jest metodą super szybkiego wyszukiwania danych w tablicach.

Polega na tym że mamy tzw. **Funkcję haszującą** (ang. Hash function), która dla danego zestawu danych tworzy liczbę zwaną haszem (ang. Hash).

Liczbę tą(hash) używamy jako indeks w **tablicy haszowanej** (ang. Hash table) do dostępu do danych

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Haszowanie - w ujęciu Javy

Hashcode – to numeryczna wartość reprezentująca obiekt

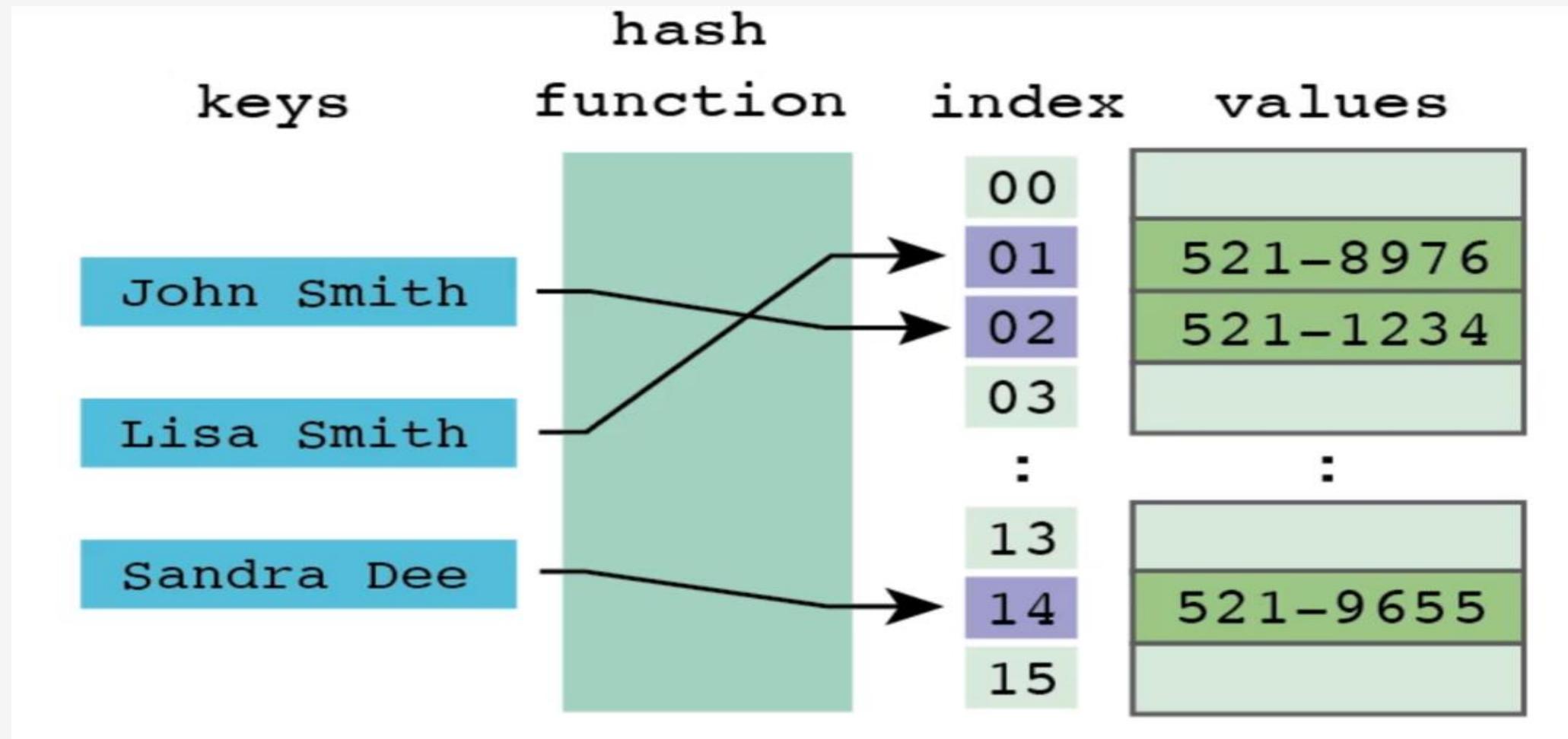
```
public class Person {  
    private int id;  
    private String address;  
  
    public Person(int id, String address) {  
        this.id = id;  
        this.address = address;  
    }  
  
    @Override  
    public int hashCode() {  
        int result = id;  
        result = 31 * result + (address != null ? address.hashCode() : 0);  
        return result;  
    }  
}
```

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Haszowanie - wizualizacja

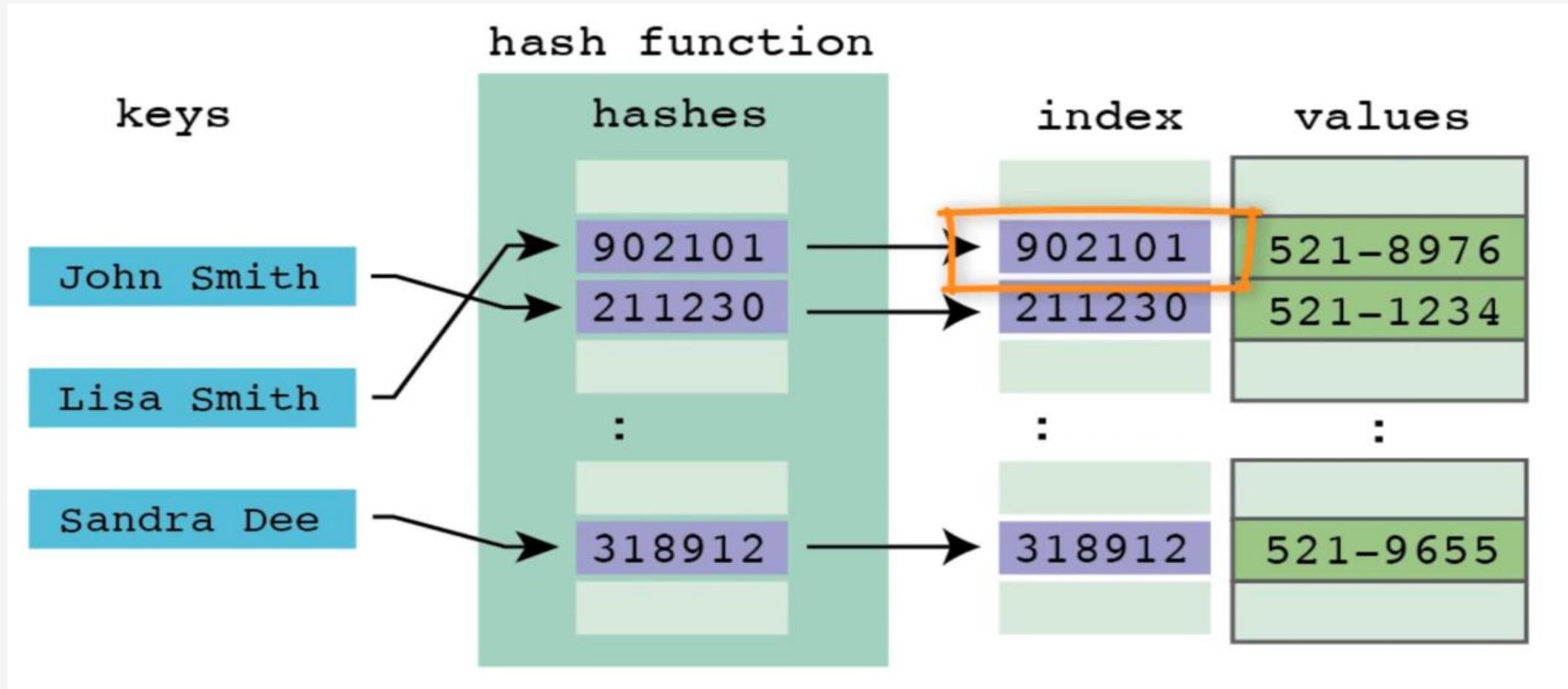


Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Haszowanie – konwersja indeksów

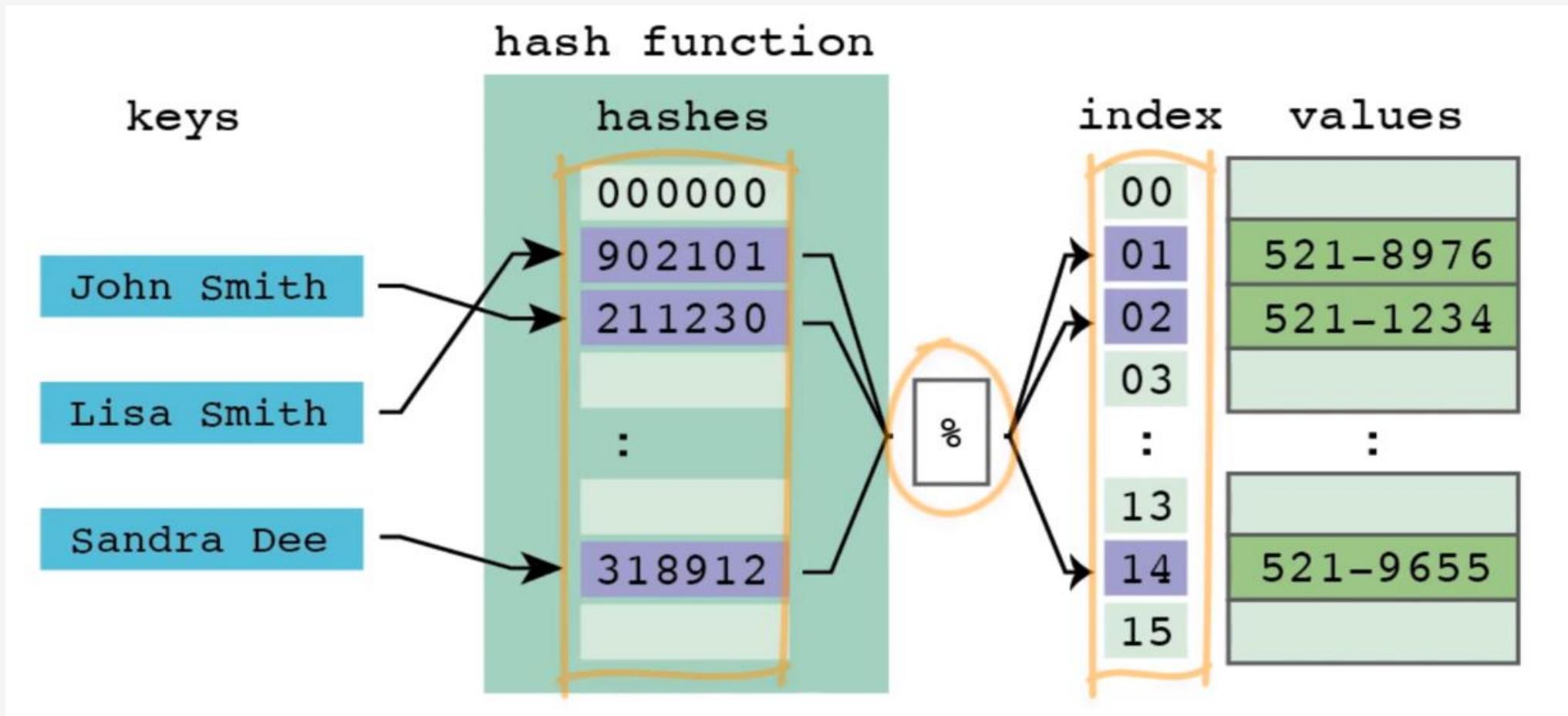


Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Haszowanie – konwersja indeksów



Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Haszowanie – konwersja indeksów operacja modulo

```
int index = hashCode % INITIAL_SIZE;
```

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Haszowanie - problemy

Podstawowym problemem haszowania to **kolizyjność**.

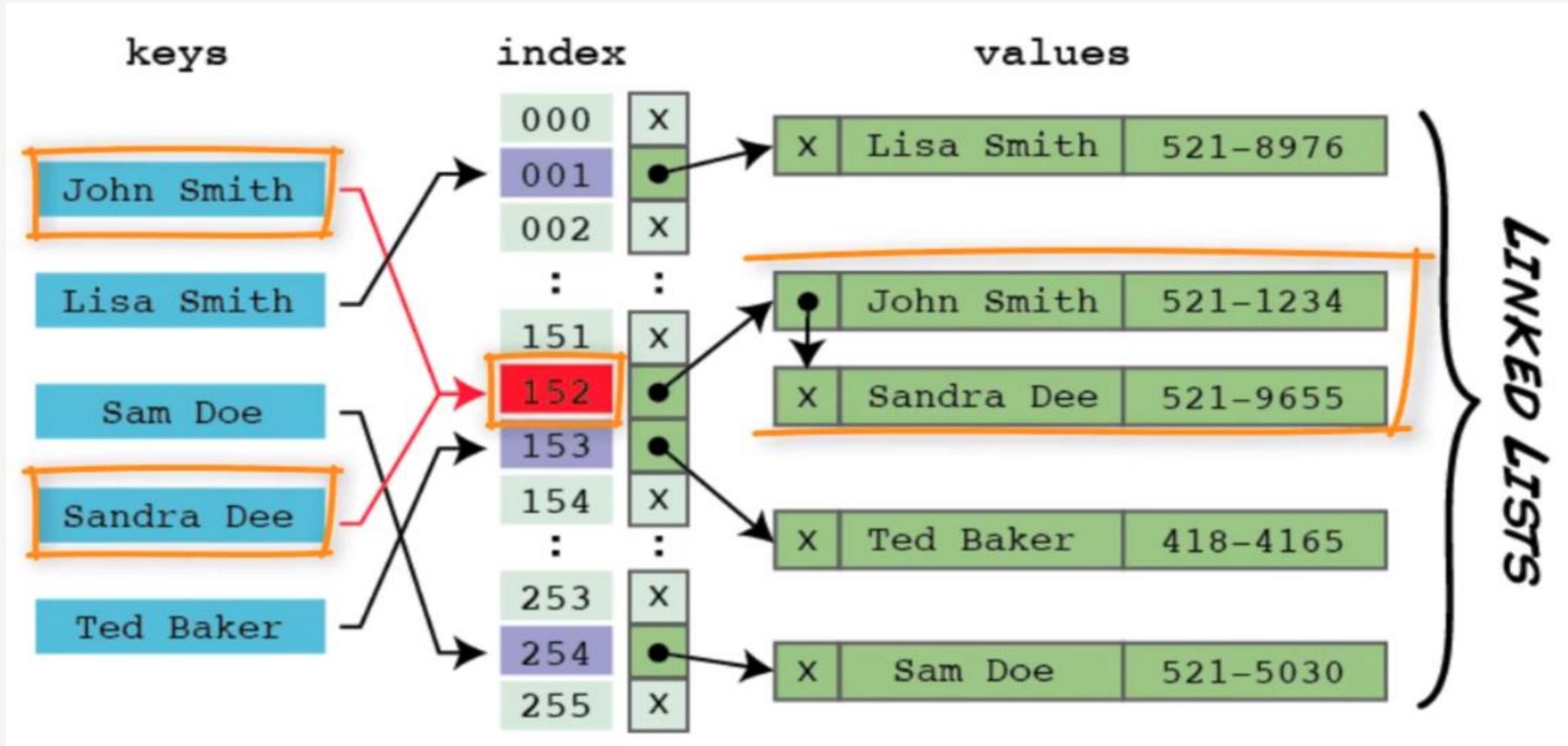
Polega one na tym, iż funkcja haszująca tworzy te same wartości hashy (indeksów w tablicy haszującej) dla wielu różnych danych.

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Haszowanie - problemy



Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Haszowanie – co należy pamiętać

- Znać wydajność operacji (put/get)
- Obiekty Immutable jako klucz
- Pojęcie haszowania w ujęciu `HashMap<K,V>` w javie
- Kolizje i co się dzieje w przypadku wystąpienia
- Opisać konwersję indeksu, czyli że `hashCode` za pomocą funkcji haszującej jest konwertowany do indeksu
- Znać strukturę mapy czyli obiekt `HashEntry`
 - Widzieć że przechowuje i klucz i wartość

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Mapy zadanie (bank_account.pdf)

- Znać wydajność operacji (put/get)
- Obiekty Immutable jako klucz
- Pojęcie haszowania w ujęciu `HashMap<K,V>` w javie
- Kolizje i co się dzieje w przypadku wystąpienia
- Opisać konwersję indeksu, czyli że `hashCode` za pomocą funkcji haszującej jest konwertowany do indeksu
- Znać struktury mapy czyli obiekt `HashEntry`
 - Widzieć że przechowuje i klucz i wartość

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Java 8 co nowego

- Wyrażenia lambda np. `(()-> System.out.println("Hi"))`
- Interfejs funkcyjny (*@Functional Interface*)
- Default method w interfejsie (*default void calculate(){}*)
- Java Stream API
- LocalDateApi(np. Klasy LocalDate i Year)

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Java 8 lamda expression

Wyrażenia lambda jest metodą którą możesz przypisywać do zmiennej

(parametry) -> {Ciało funkcji} np.

*(x, y) -> {System.out.println(x * y)}*

Podawanie typów jest opcjonalne:

*(Integer x, Long y) -> System.out.println(x * y)*

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Java 8 lamda expression

Wyrażenia lambda jest metodą którą możesz przypisywać do zmiennej

(parametry) -> Ciało_funkcji np.

*(x, y) -> System.out.println(x * y)*

Podawanie typów jest opcjonalne:

*(Integer x, Long y) -> System.out.println(x * y)*

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy

Java 8 lamda expression



(parametry) -> Ciało_funkcji

Jeśli ciało funkcji zawiera więcej linii należy użyć klamer {}, np.

```
(x) -> {  
  
    if(x % 2 == 0){  
        return x * x;  
    }else{  
        return x;  
    }  
}
```

}

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Java 8 Interfejs funkcyjny

Interfejs funkcyjny - to interfejs który ma jedną
tylko abstrakcyjną metodę

Aby oznaczyć interfejs w Javie jak funkcyjny, należy użyć
adnotacji **@FunctionalInterface**

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Java 8 wbudowane interfejsy funkcyjne

Interfejs	Sygnatura funkcji	Przykład
UnaryOperator<T>	T apply(T t)	String::toLowerCase
BinaryOperator<T>	T apply(T t1, T t2)	BigInteger::add
Predicate<T>	boolean test(T t)	Collection::isEmpty
Function<T,R>	R apply(T t)	Arrays::asList
Supplier<T>	T get()	Instant::now
Consumer<T>	void accept(T t)	System.out::println

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Stream API jako 4 sposób iterowania

Strumień (Steam) reprezentuje sekwencje elementów i pozwala na różne operacje na tych elementach. Operacje te mogą być pośrednie i takie możemy układać w łańcuchy metod, oraz końcowe, zwracające wynik, lub nie.

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Podstawowe operacje na streamach

- *filter()* - zwraca strumień zawierający tylko te elementy dla których filtr zwrócił wartość true,
- *map()* - każdy z elementów może zostać zmieniony do innego typu, nowy obiekt zawarty jest w nowym strumieniu,
- *peek()* - pozwala przeprowadzić operację na każdym elemencie w strumieniu, zwraca strumień z tymi samymi elementami,
- *limit()* - zwraca strumień ograniczony do zadanej liczby elementów, pozostałe są ignorowane.

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Kończenie strumienia(terminal operations)

- **forEach()** - wykonuje zadaną operację dla każdego elementu,
- **count()** - zwraca liczbę elementów w strumieniu,
- **allMatch()** - zwraca flagę informującą czy wszystkie elementy spełniają warunek. Przestaje sprawdzać na pierwszym elemencie, który tego warunku nie spełnia,
- **collect()** - pozwala na utworzenie nowego typu na podstawie elementów strumienia. Przy pomocy tej metody można na przykład utworzyć listę. Klasa Collectors zawiera sporo gotowych implementacji.

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy

Zadanie (plik notebook.pdf)



Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Co na następnych zajęciach - algorytmy

- Sortowanie na przykładzie algorytmów QuickSort i MergeSort:
 - <https://pl.khanacademy.org/computing/computer-science/algorithms/quick-sort/a/overview-of-quicksort>
 - <https://pl.khanacademy.org/computing/computer-science/algorithms/merge-sort/a/divide-and-conquer-algorithms>
- Rekurencja
 - <https://javastart.pl/baza-wiedzy/darmowy-kurs-java/zaawansowane-programowanie/rekurencja-rekursja>
 - <https://www.baeldung.com/java-recursion>

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Co na następnych zajęciach - algorytmy

- Co nowego w Java 8:
 - StreamAPI na przykładzie struktur danych
 - Wyrażenia lambda w javie
 - @FunctionalInterface
- Mapy i haszowanie
 - <https://www.geeksforgeeks.org/hashing-in-java/>
- Drzewa binarne (BST):
 - <https://www.baeldung.com/java-binary-tree>
 - <http://informatyka.wroc.pl/node/483>

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Rekurencja(*ang. Recursion*)

- To odwołanie się funkcji do samej siebie
- Używana najczęściej w zadaniach, gdzie rozwiązanie problemu zależy od rozwiązań mniejszych instancji tzw. "podproblemów" tego samego problemu -> strategia dzieli i zwyciężaj.

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Rekurencja(*ang. Recursion*)

- Funkcja rekurencyjna w celu zwrócenia prawidłowego wyniku , wywołuję samą siebie(a mówiąc dokładniej, tworzy swoje kopie) aż do napotkania tzw. przypadku podstawowego

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



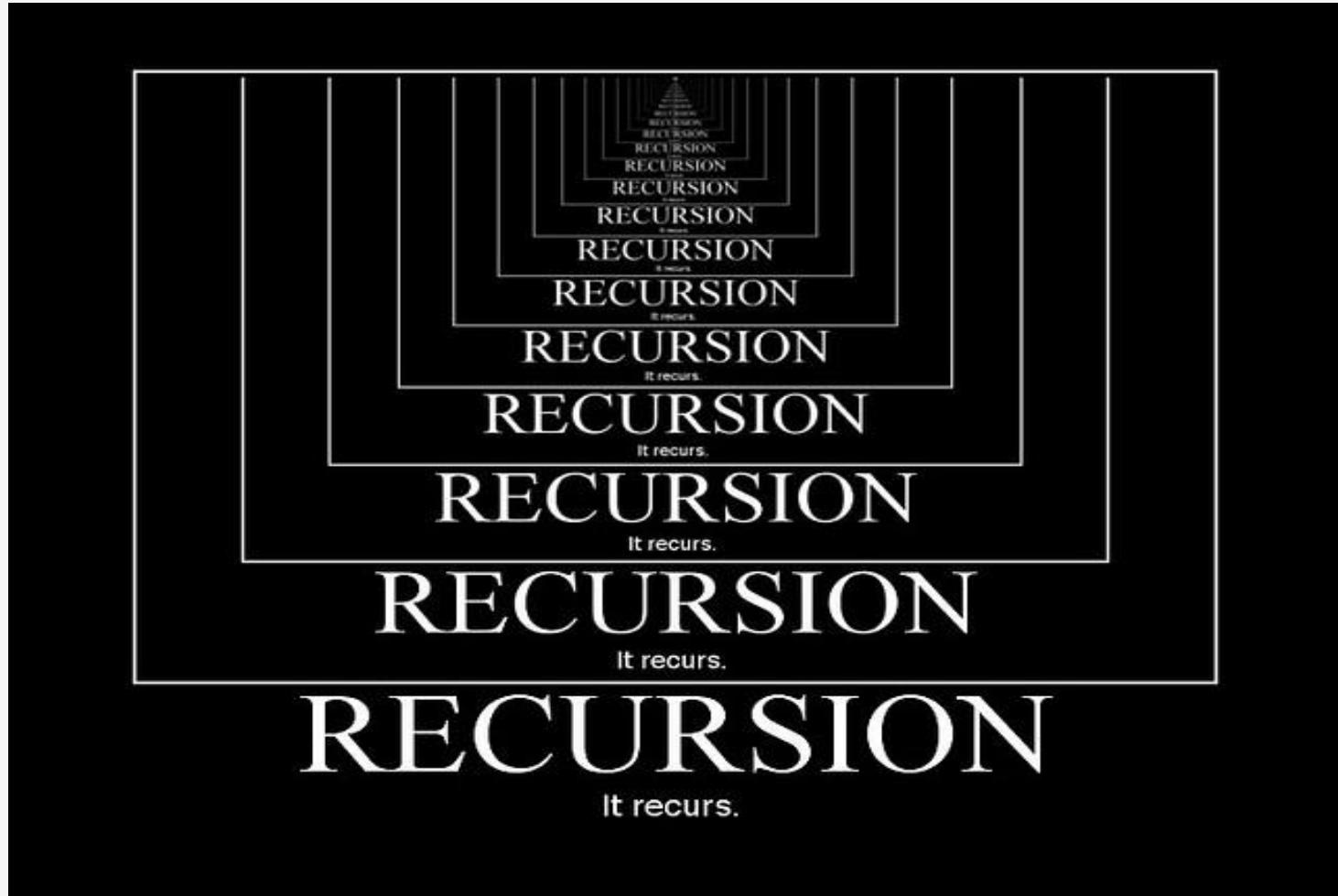
Rekurencja(*ang. Recursion*)

- Podczas używania rekurencji należy pamiętać o zdefiniowaniu przypadków podstawowych(w celu uniknięcia `java.lang.StackOverflowError`)

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy

Rekurencja(ang. *Recursion*)



Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Rekurencja(*ang. Recursion*)

**Rekurencja
funkcja wywołuje samą siebie?**



Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Rekurencja(ang. *Recursion*) wywołanie dla $n = 3$



$$f(3) = f(2)+2$$

```
int f (int n)
{
    if (n==0) return 3;
    else return f (n-1)+2;
}
```

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Rekurencja(ang. Recursion)

```
int f (int n)
{
    if (n==0) return 3;
    else return f (n-1)+2;
}
```



$$f(3) = f(2)+2$$



$$f(2) = f(1)+2$$



Rekurencja(*ang. Recursion*)

```
int f (int n)
{
    if (n==0) return 3;
    else return f (n-1)+2;
}
```



$$f(3) = f(2)+2$$



$$f(2) = f(1)+2$$



$$f(1) = f(0)+2$$

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Rekurencja(ang. Recursion)

```
int f (int n)
{
    true
    if (n==0) return 3;
    else return f (n-1)+2;
}
```



$$f(3) = f(2)+2$$



$$f(2) = f(1)+2$$



$$f(1) = f(0)+2$$



$$f(0) = 3$$

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Rekurencja(ang. Recursion)

```
int f (int n)
{
    true
    if (n==0) return 3;
    else return f (n-1)+2;
}
```



$$f(3) = f(2)+2$$



$$f(2) = f(1)+2$$



$$f(1) = f(0)+2$$



$$f(0) = 3$$

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Rekurencja(ang. Recursion)

```
int f (int n)
{
    if (n==0) return 3;
    else return f (n-1)+2;
}
```



$$f(3) = f(2)+2$$



$$f(2) = f(1)+2$$



$$f(1) = 3 +2$$



$$f(0) = 3$$

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Rekurencja(ang. Recursion)

```
int f (int n)
{
    if (n==0) return 3;
    else return f (n-1)+2;
}
```



$$f(3) = f(2)+2$$



$$f(2) = f(1)+2$$



$$f(1) = 3 +2$$



$$f(0) = 3$$

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Rekurencja(ang. Recursion)

```
int f (int n)
{
    if (n==0) return 3;
    else return f (n-1)+2;
}
```



$$f(3) = f(2)+2$$



$$f(2) = 5 +2$$



$$f(1) = 3 +2$$



$$f(0) = 3$$

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Rekurencja(ang. Recursion)

```
int f (int n)
{
    if (n==0) return 3;
    else return f (n-1)+2;
}
```



$$f(3) = 7 + 2$$



$$f(2) = 5 + 2$$



$$f(1) = 3 + 2$$



$$f(0) = 3$$

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Rekurencja(ang. Recursion)

```
int f (int n)
{
    if (n==0) return 3;
    else return f (n-1)+2;
}
```



$$f(3) = 7 + 2$$



$$f(2) = 5 + 2$$



$$f(1) = 3 + 2$$



$$f(0) = 3$$

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Rekurencja(ang. Recursion)

```
int f (int n)
{
    if (n==0) return 3; przypadek podstawowy
    else return f (n-1)+2;
}
```



Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Rekurencja – suma liczb od 1 do N

```
public int iterationSum(int N){  
    int result = 0;  
  
    for(int i=1;i<N;++i){  
        result = result + i;  
    }  
  
    return result;  
}
```

```
public int recursionSum(int N){  
    if( N == 1 ) return 1;  
  
    return N + recursionSum(N-1);  
}
```

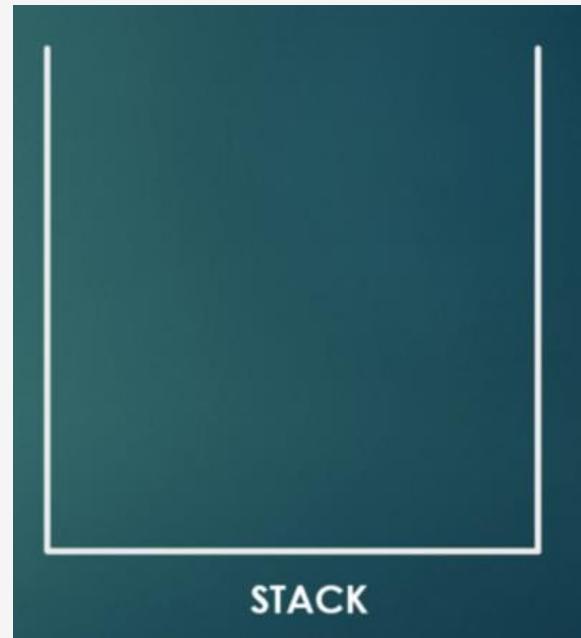
Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Rekurencja – suma liczb od 1 do 4

```
public static int recursionSum(int N) {  
    if (N == 1) return 1;  
  
    return N + recursionSum(N: N - 1);  
}
```



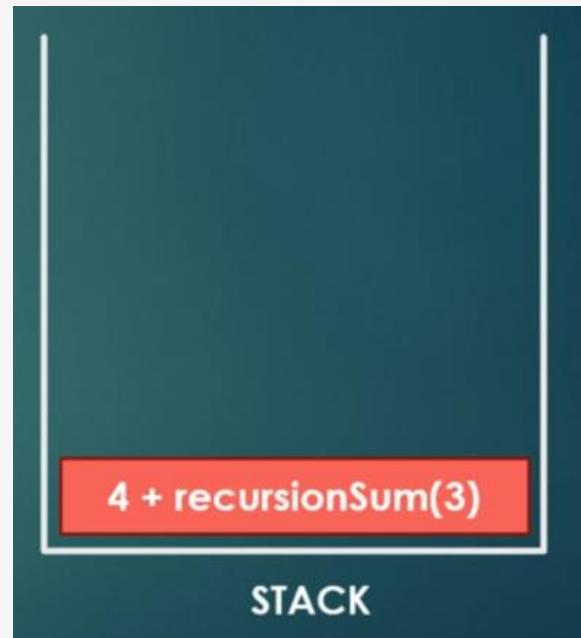
Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Rekurencja – suma liczb od 1 do 4

```
public static int recursionSum(int N) {  
    if (N == 1) return 1;  
  
    return N + recursionSum(N - 1);  
}
```



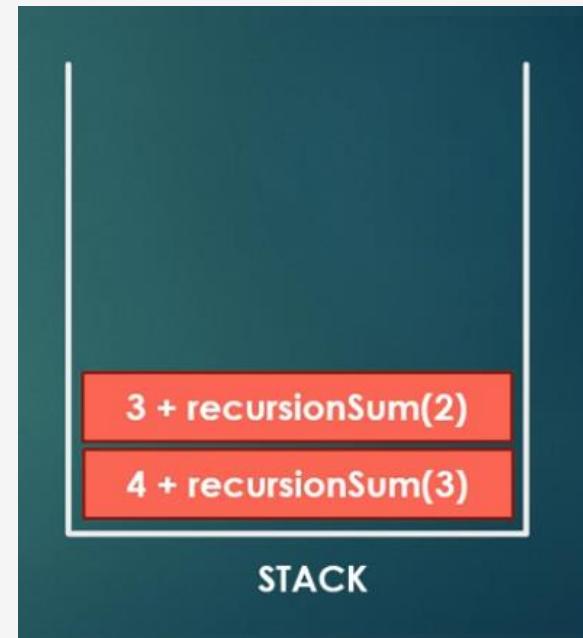
Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Rekurencja – suma liczb od 1 do 4

```
public static int recursionSum(int N) {  
    if (N == 1) return 1;  
  
    return N + recursionSum(N - 1);  
}
```



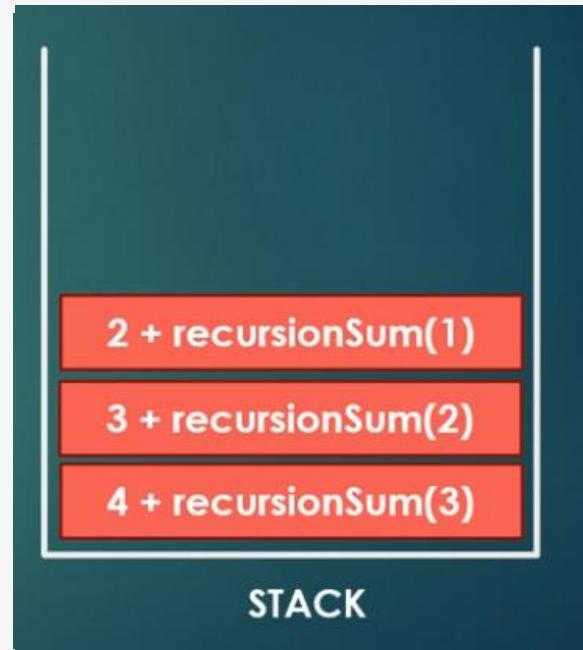
Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Rekurencja – suma liczb od 1 do 4

```
public static int recursionSum(int N) {  
    if (N == 1) return 1;  
  
    return N + recursionSum(N - 1);  
}
```



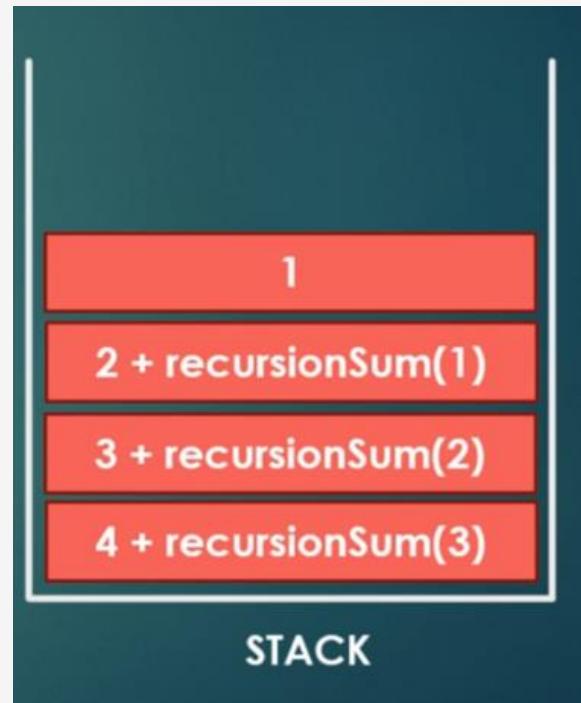
Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Rekurencja – suma liczb od 1 do 4

```
public static int recursionSum(int N) {  
    if (N == 1) return 1;  
  
    return N + recursionSum(N - 1);  
}
```



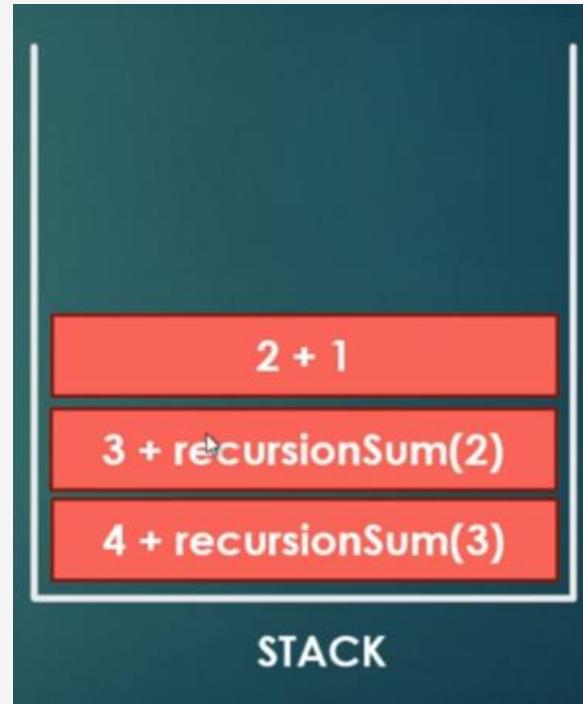
Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Rekurencja – suma liczb od 1 do 4

```
public static int recursionSum(int N) {  
    if (N == 1) return 1;  
  
    return N + recursionSum(N - 1);  
}
```



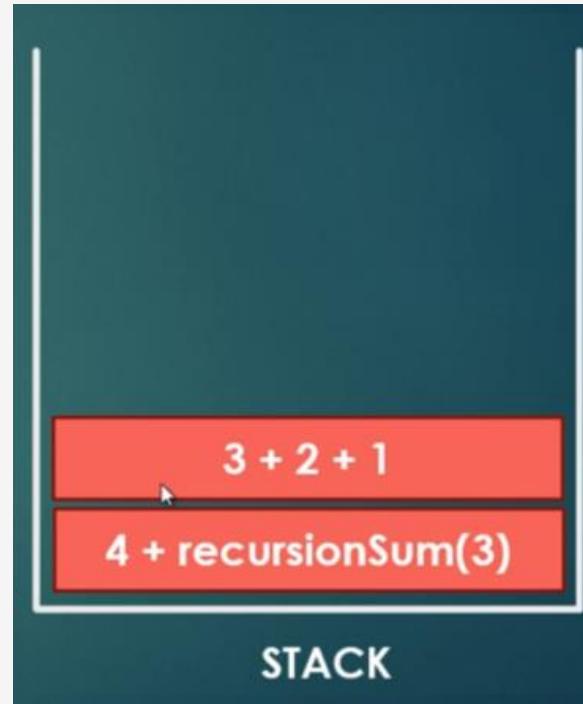
Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Rekurencja – suma liczb od 1 do 4

```
public static int recursionSum(int N) {  
    if (N == 1) return 1;  
  
    return N + recursionSum(N - 1);  
}
```



Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Rekurencja – suma liczb od 1 do 4

```
public static int recursionSum(int N) {  
    if (N == 1) return 1;  
  
    return N + recursionSum(N - 1);  
}
```



Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Rekurencja – suma liczb od 1 do 4

```
public static int recursionSum(int N) {  
    if (N == 1) return 1;  
  
    return N + recursionSum(N - 1);  
}
```

```
recursionSum(4)  
recursionSum(3)  
recursionSum(2)  
recursionSum(1)  
    return 1  
    return 2+1  
    return 3+2+1  
return 4+3+2+1
```

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Rekurencja vs Iteracja

- Porównując metodę iteracyjną i rekurencyjną, należy pamiętać, że w przypadku rekurencji musimy wykonać co najmniej 2x operacji.
- Czyli najpierw "*rozwijamy*" wywołania(dodajemy do stosu wywołań programu) aż osiągniemy warunek brzegowy.
- Następnie od ostatnio dodanego wywołania na stosie iterujemy do ostatniego wywołania.

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Rekurencja – lewostronna vs prawostronna

- Jeśli wywołanie rekurencyjne występuje na końcu metody, a przed wywołaniem ma miejsce wykonanie operacji, mamy doczynienia z **rekurencją prawostronną**
- W przypadku gdy zanim wywołamy metodę rekurencyjną, a na końcy wykonamy operację mamy doczynienia z **rekurencje lewostronna**
- Dla przykładu metody *head* i *tail* w klasie **Recursion.java**

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Rekurencja – ciąg fibonacciego

- Ciąg Fibonacciego – ciąg liczb naturalnych określony rekurencyjnie w sposób następujący:

$$F_n := \begin{cases} 0 & \text{dla } n = 0; \\ 1 & \text{dla } n = 1; \\ F_{n-1} + F_{n-2} & \text{dla } n > 1. \end{cases}$$

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Rekurencja – ciąg fibonacciego

Ciąg liczb naturalnych, Fibonacciego wygląda następująco:

- 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987 itd.

Ciąg zaczyna się od 1 oraz 1 . Każda kolejna liczba ciągu to suma dwóch poprzednich.

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Rekurencja – ciąg fibonacciego ZADANIE

W klasie `Recursion.java` zaimplementuj metodę *fib*, wykorzystując równianie dla ciągu fibonacciego:

$$F_n := \begin{cases} 0 & \text{dla } n = 0; \\ 1 & \text{dla } n = 1; \\ F_{n-1} + F_{n-2} & \text{dla } n > 1. \end{cases}$$

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Sortowanie szybkie (ang. Quick sort)

Rekurencyjny algorytm sortowania danych, stosujący metodę *dziel i zwyciężaj*.

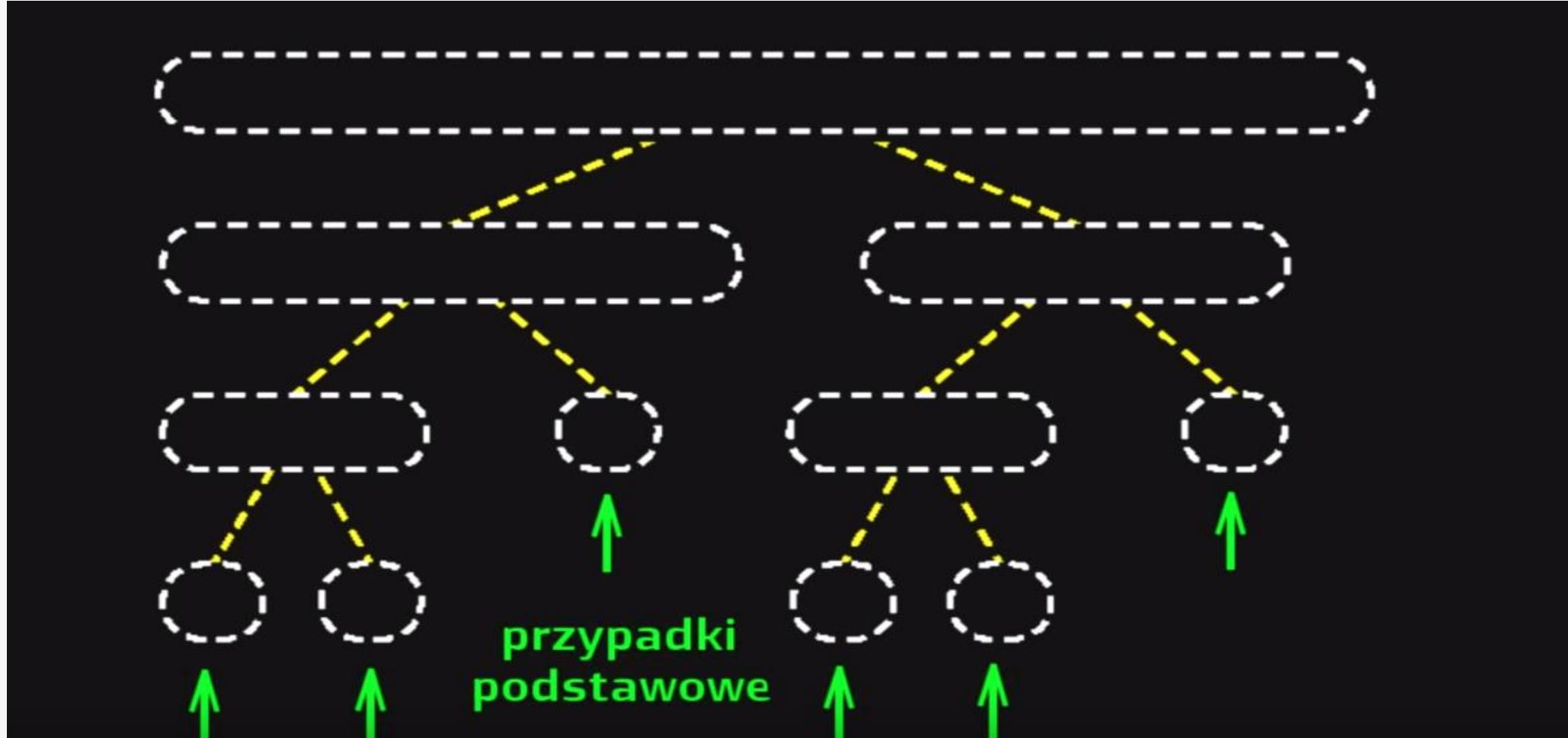
W przypadku quicksort zastosowano podejście, podziału dużej tablicy , na mniejsze - które łatwiej posortować.

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Sortowanie szybkie (ang. Quick sort)



Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Sortowanie szybkie (ang. Quick sort)

Tablica do posortowania:

24	11	42	65	93	54	14	82
----	----	----	----	----	----	----	----

0 1 2 3 4 5 6 7

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Sortowanie szybkie (ang. Quick sort)

Zanim przystąpimy do sortowania , należy wybrać oś (ang. Pivot), ustawiana jest :

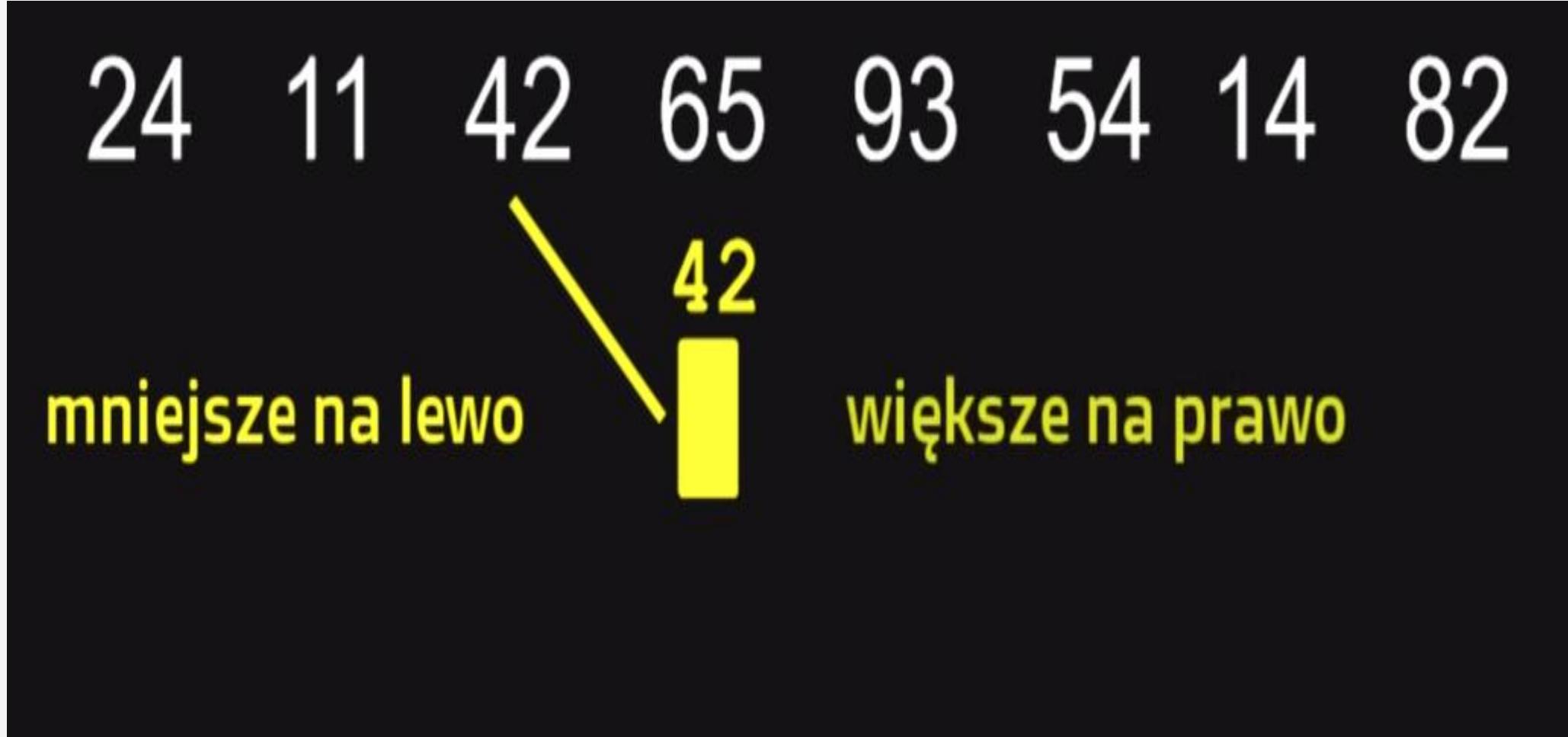
- w połowie tablicy,
- wyznaczana losowo,
- może to być także skrajny element tablicy

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Sortowanie szybkie (ang. Quick sort)

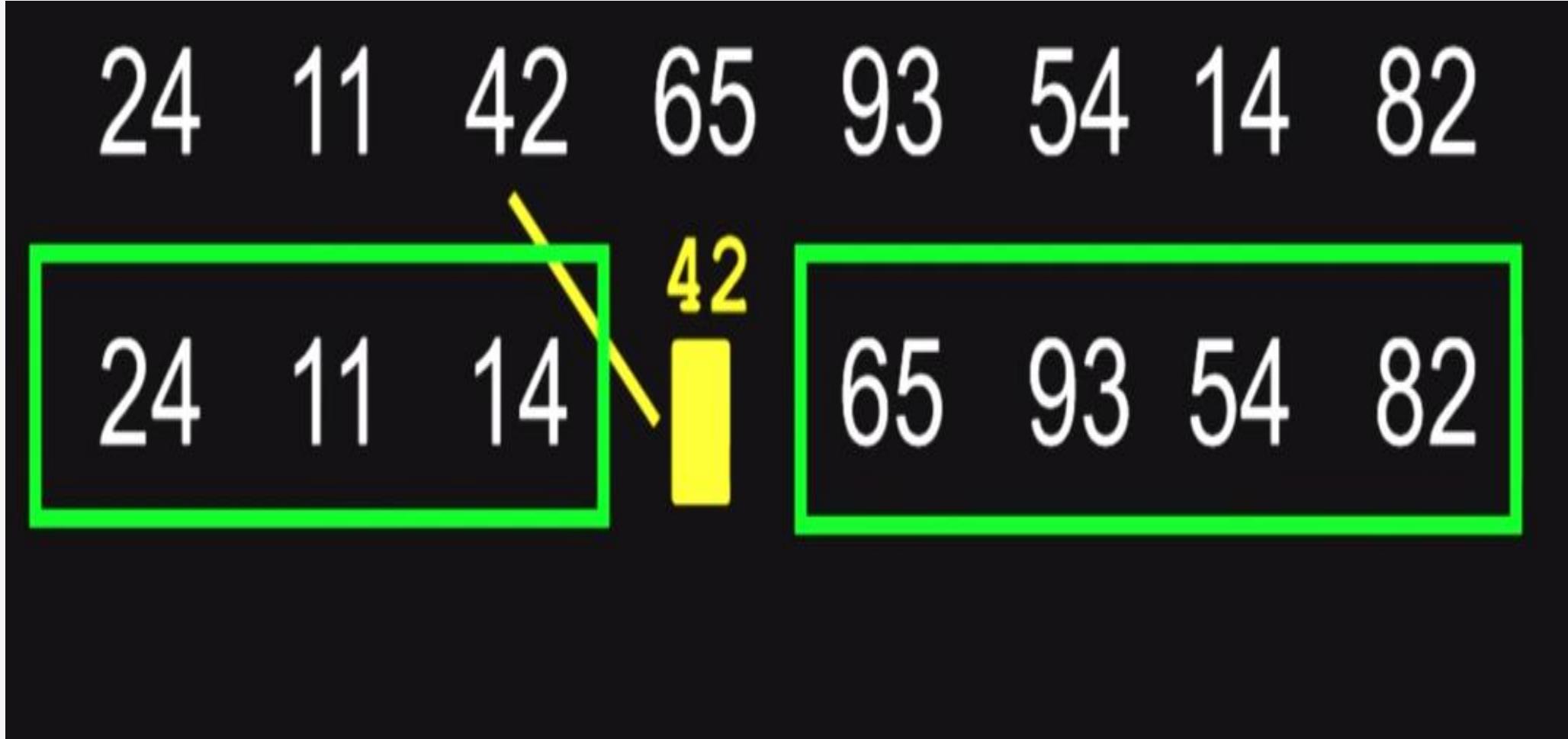


Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Sortowanie szybkie (ang. Quick sort)

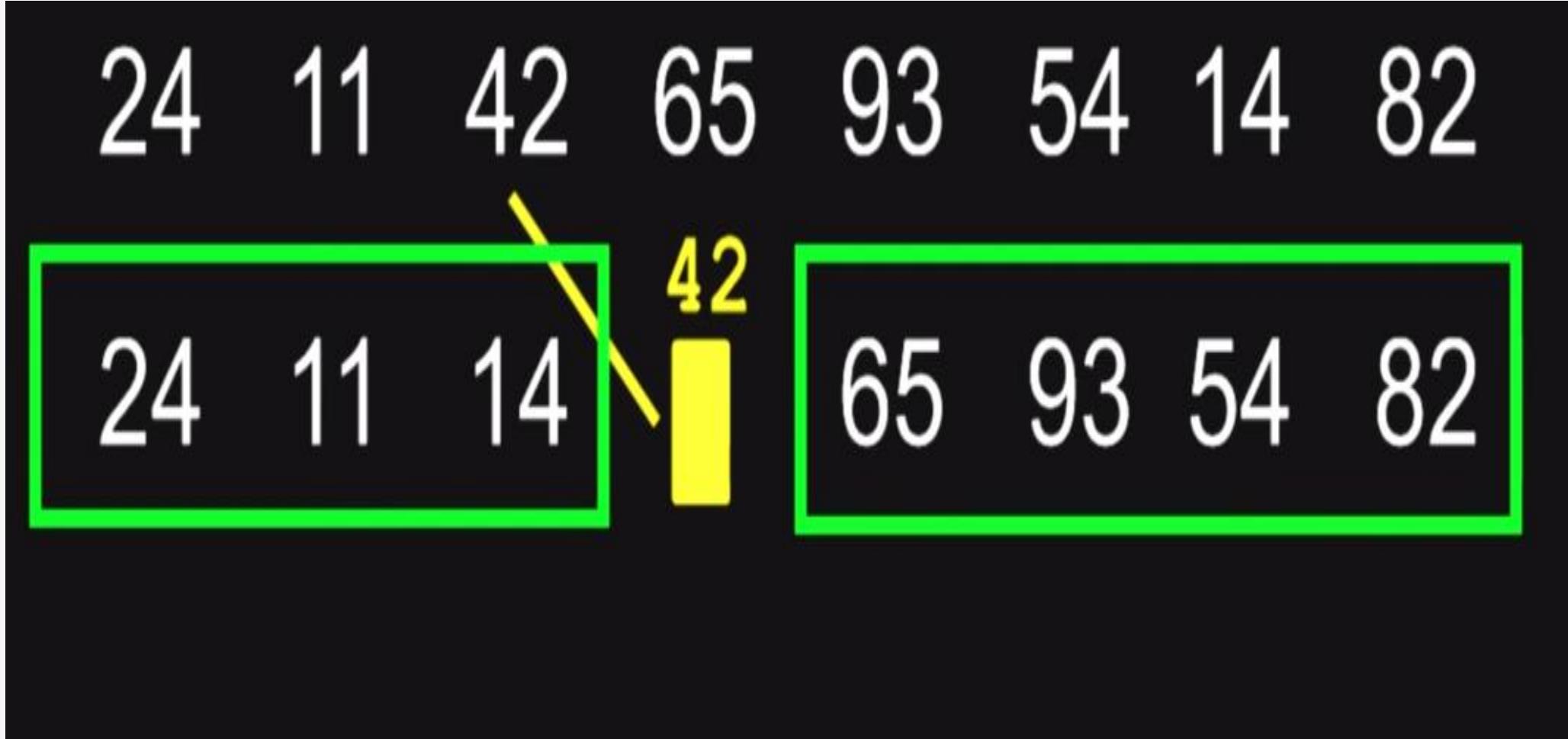


Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Sortowanie szybkie (ang. Quick sort)

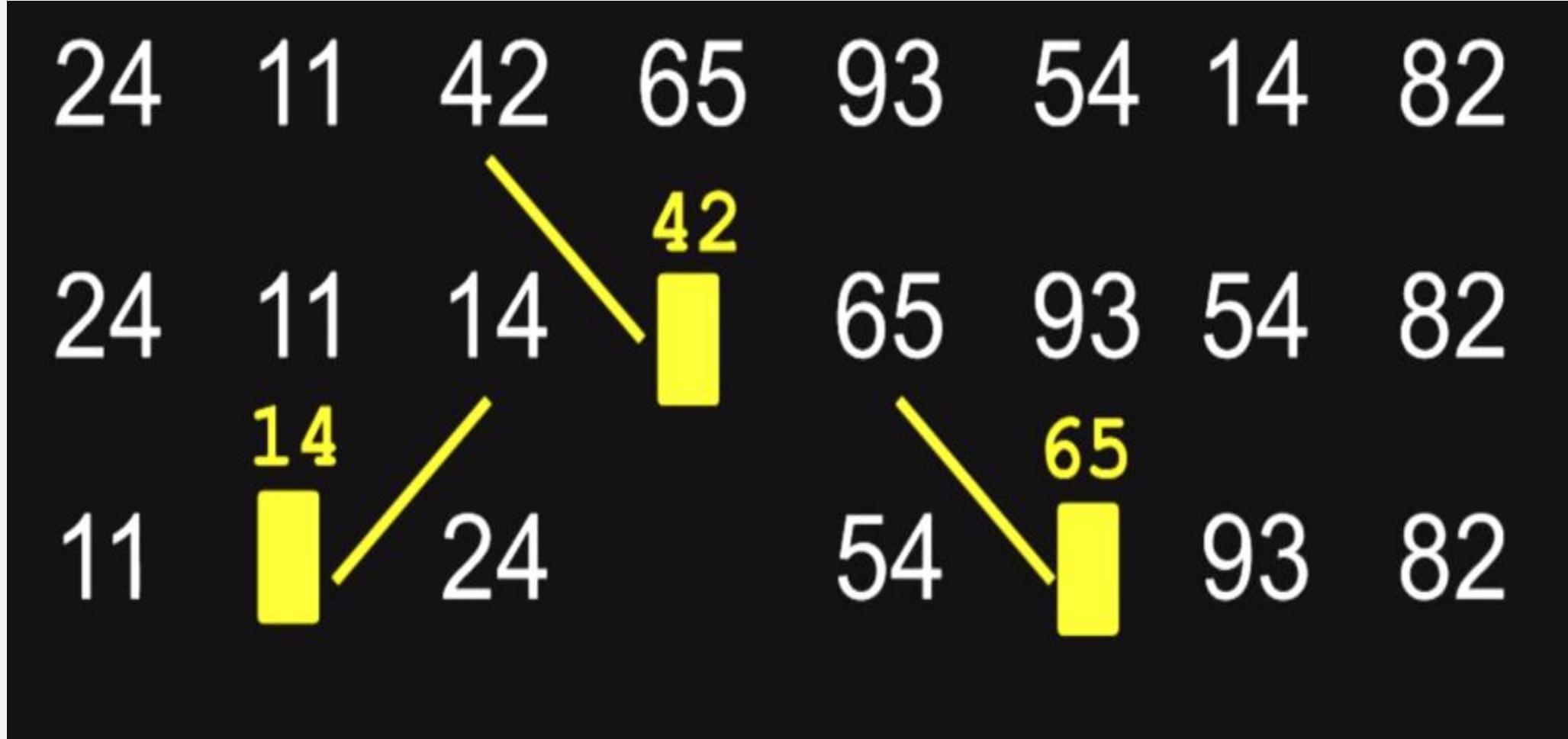


Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Sortowanie szybkie (ang. Quick sort)

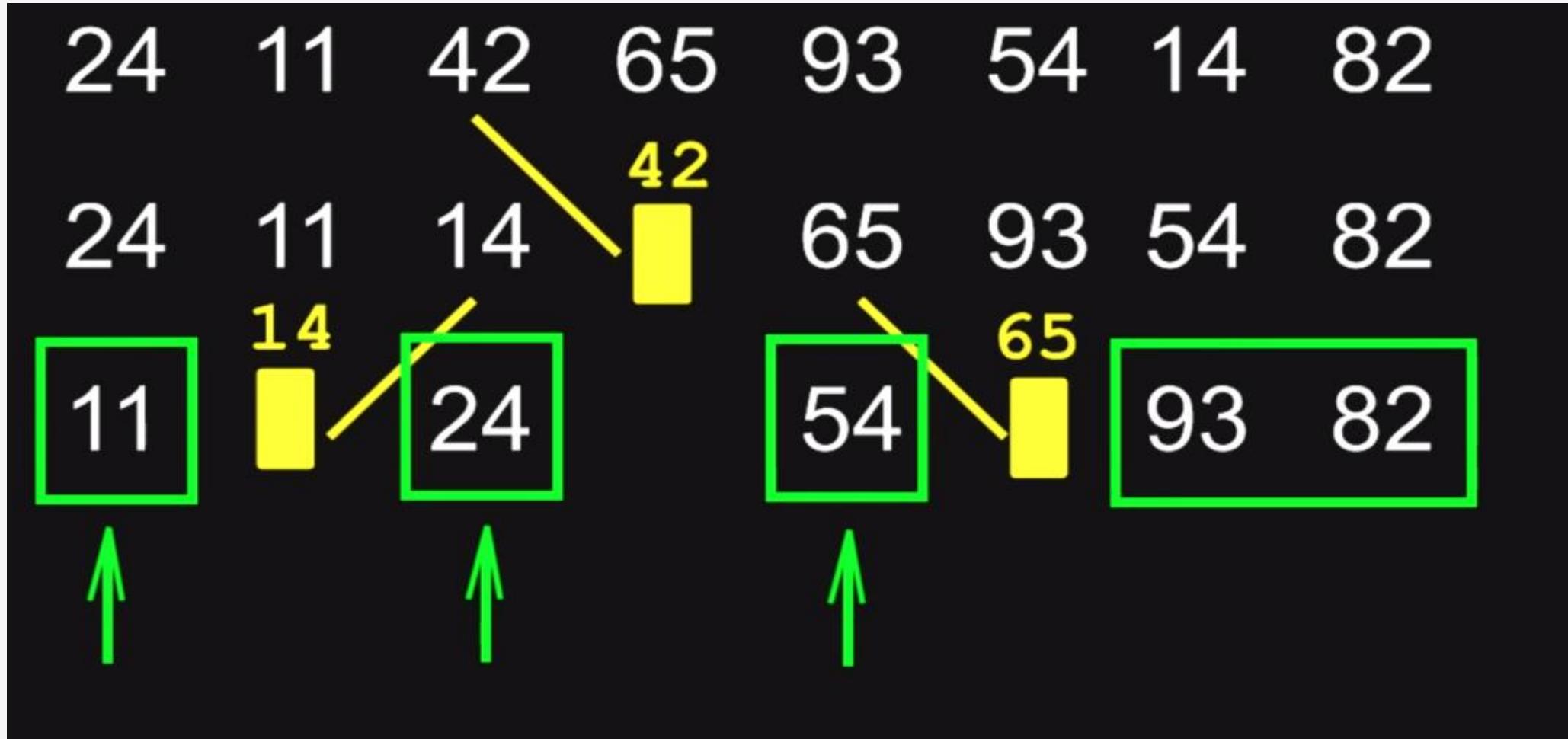


Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Sortowanie szybkie (ang. Quick sort)

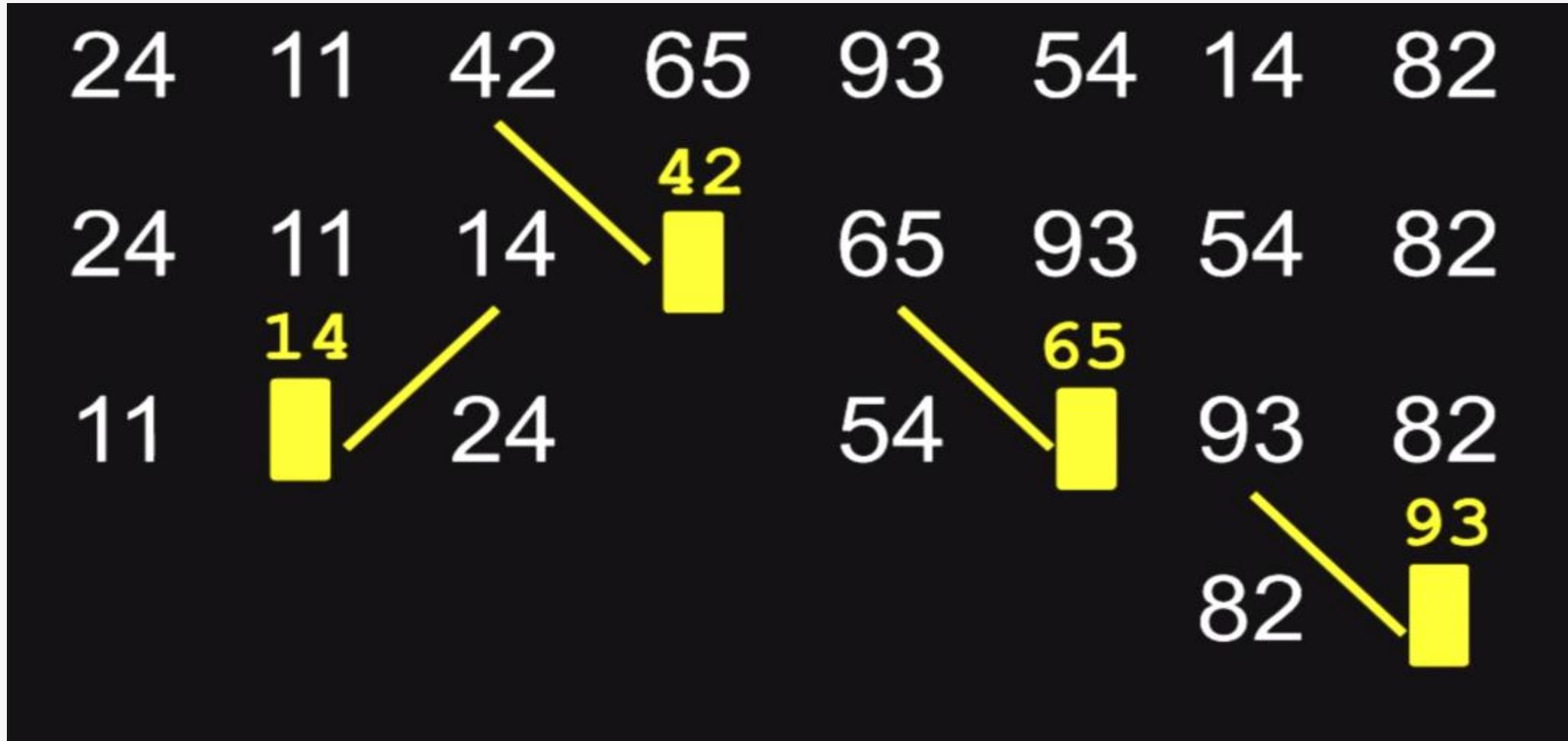


Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Sortowanie szybkie (ang. Quick sort)

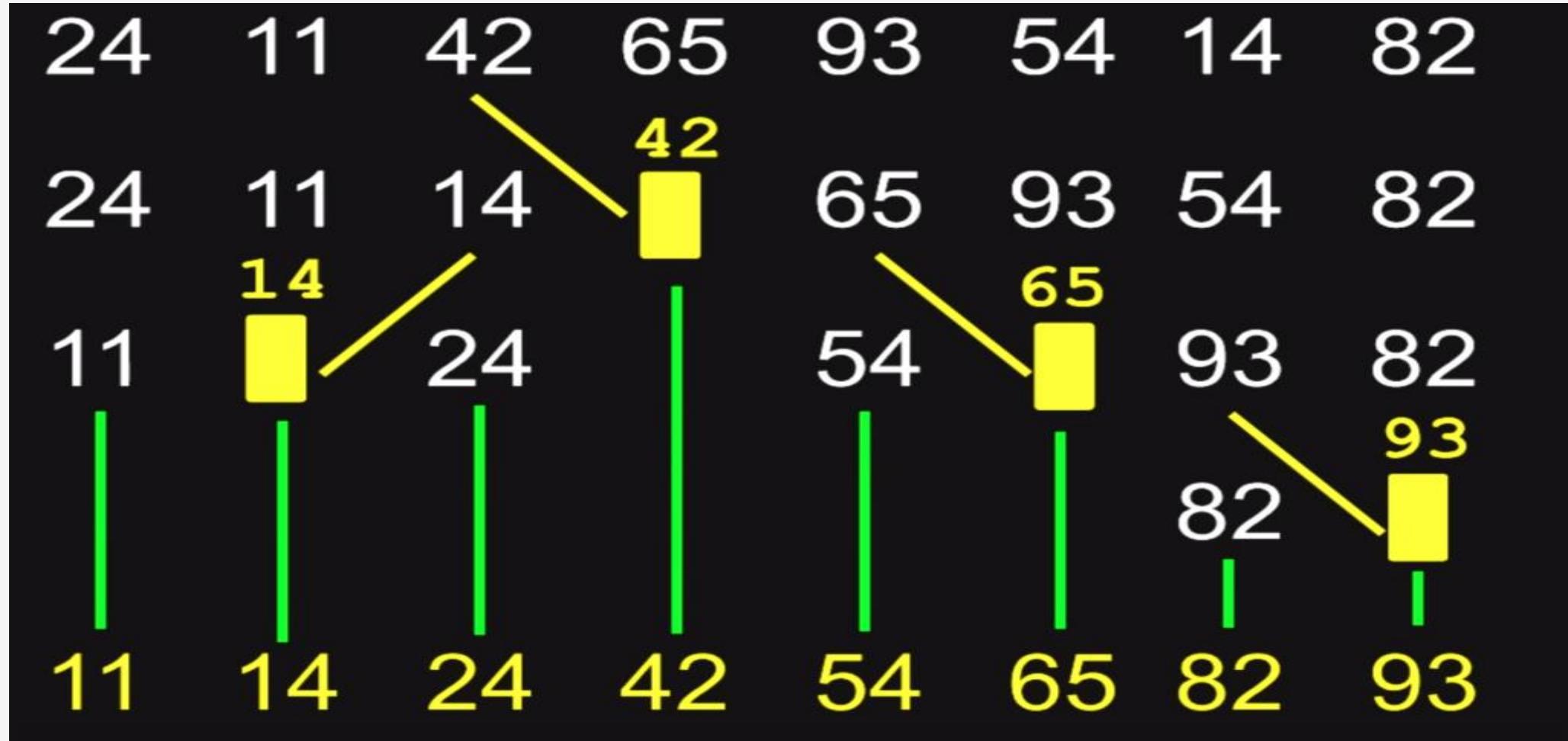


Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Sortowanie szybkie (ang. Quick sort)



Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Sortowanie szybkie – algorytm partycjonujący

Realizuje podział zadanego zbioru danych na podzbiory(partycje) według ustalonego kryterium (kolejności w przypadku sortowania)

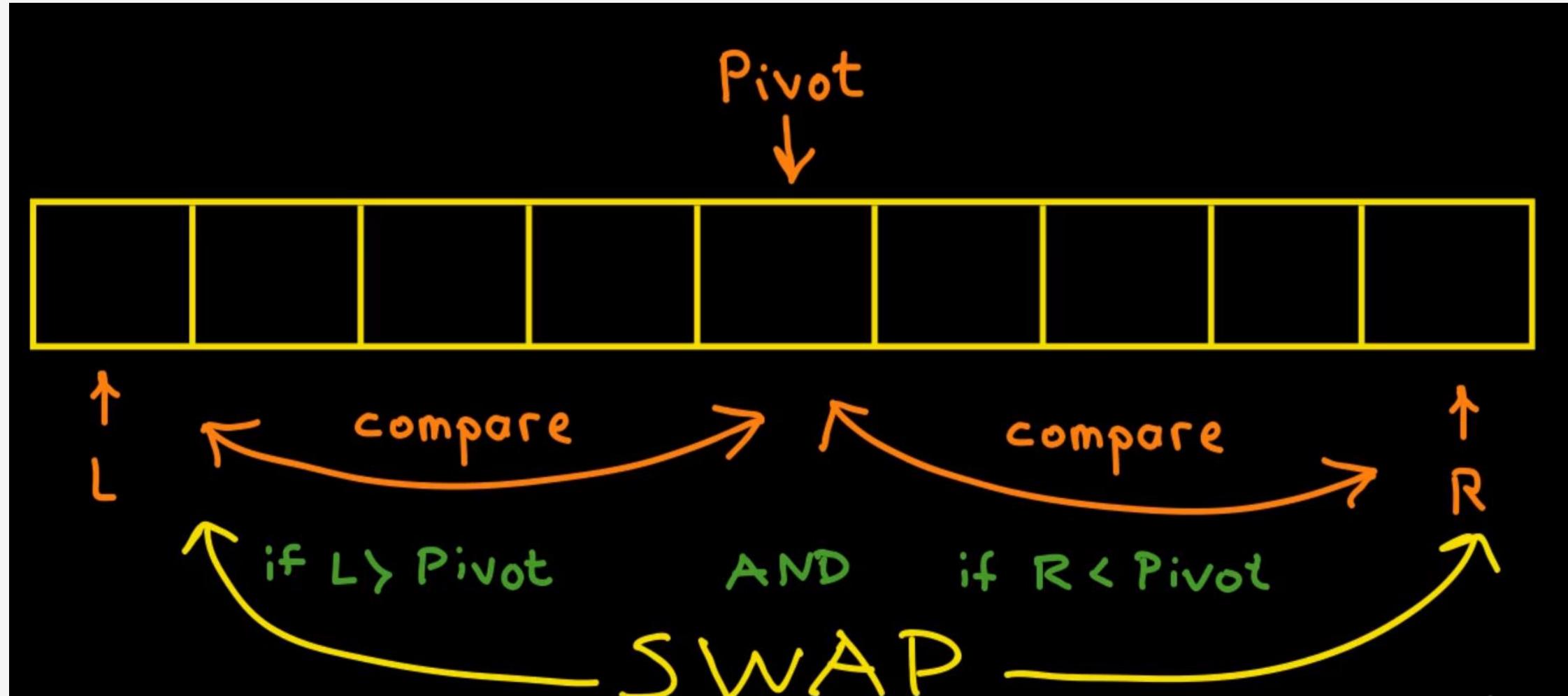
W sortowaniu quicksort o przynależności do danej partycji decyduje porównanie wartości elementu z wartością osi (pivot)

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Sortowanie szybkie (ang. Quick sort) - partycjonowanie

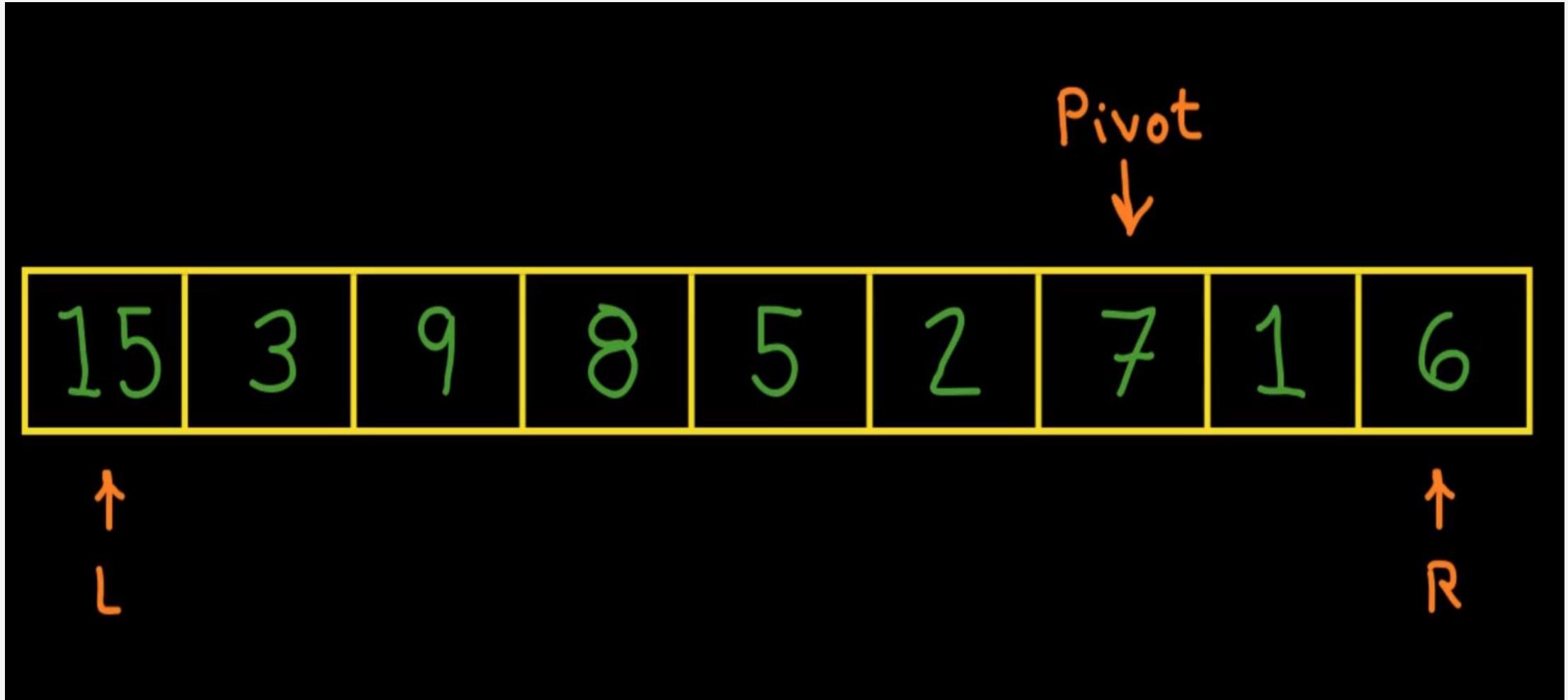


Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



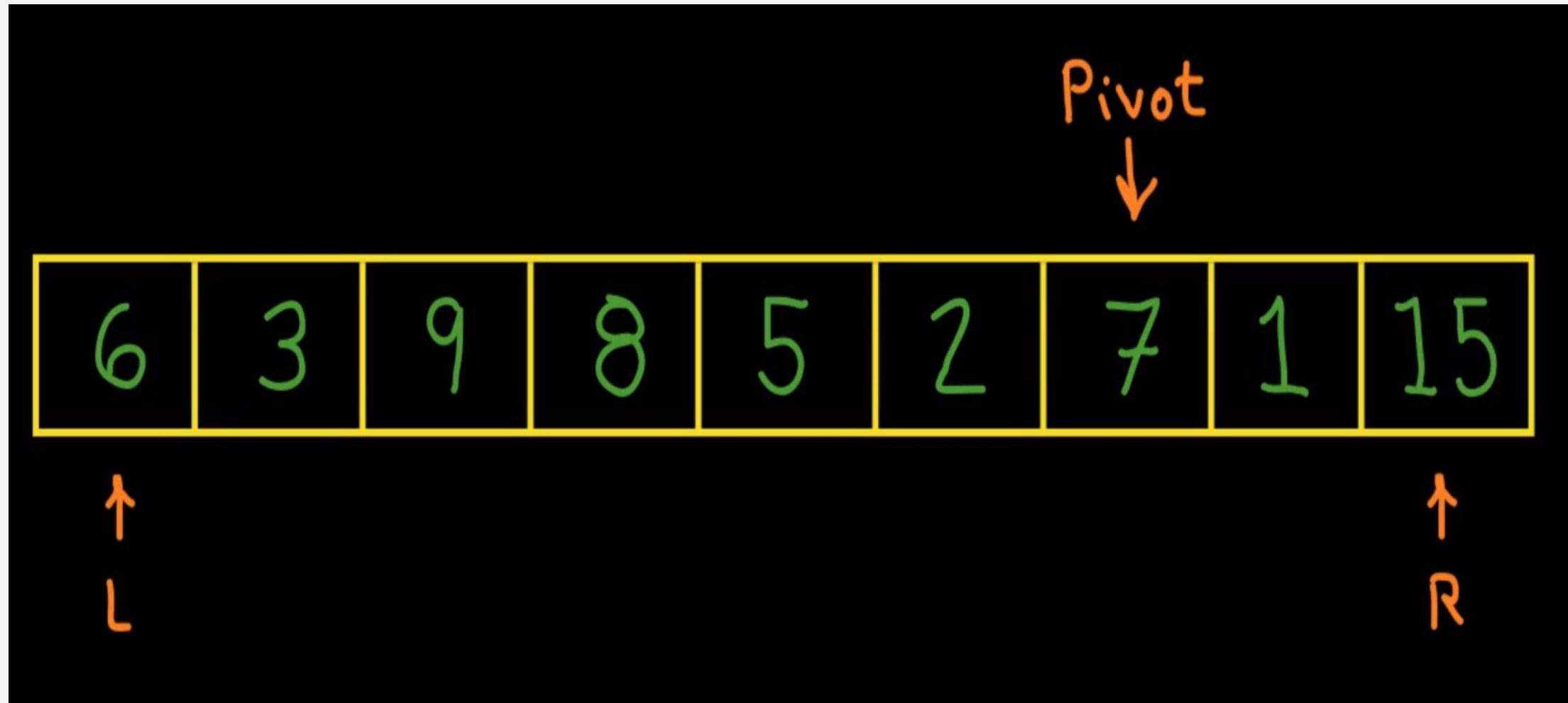
Sortowanie szybkie - partycjonowanie



Autor:

Prawa do korzystania z materiałów posiada Software Development Academy

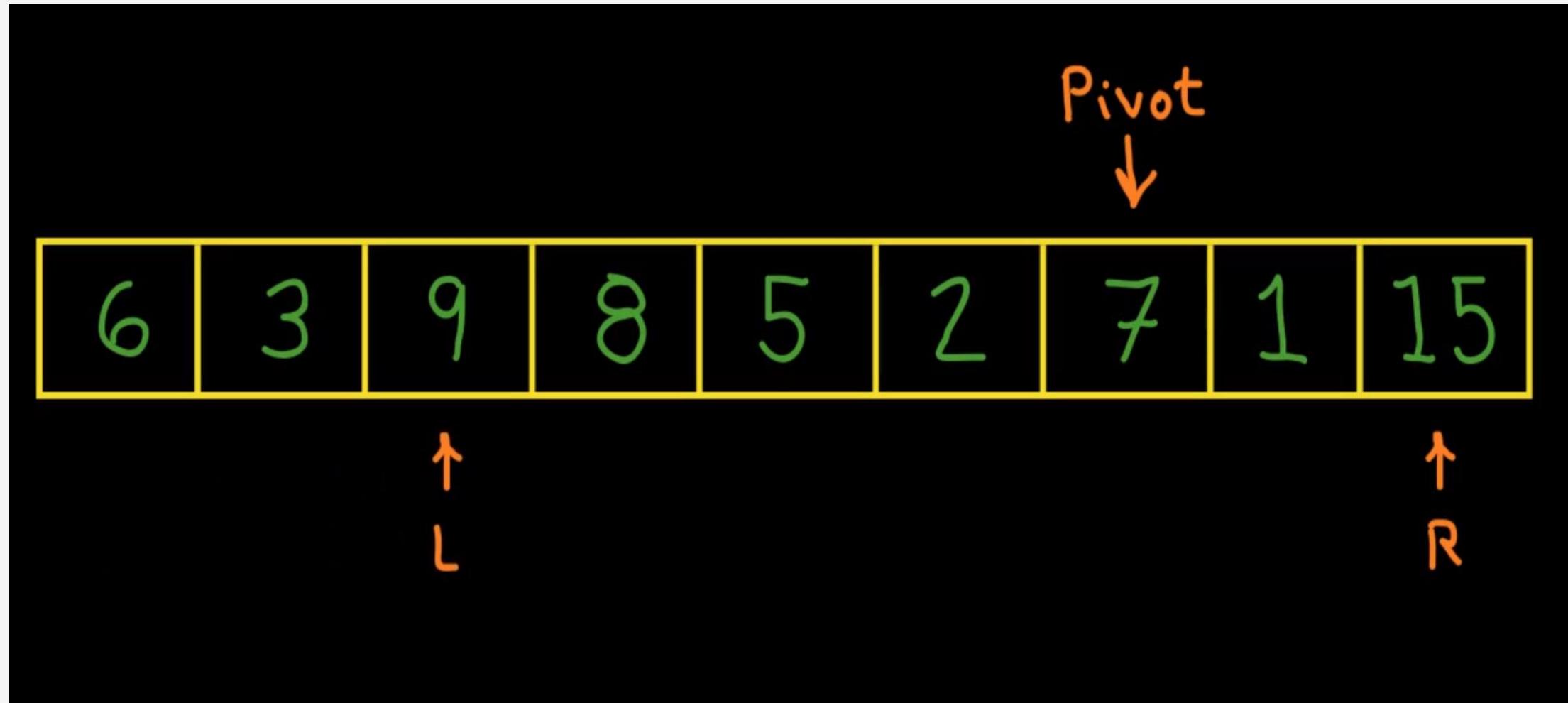
Sortowanie szybkie - partycjonowanie



Autor:

Prawa do korzystania z materiałów posiada Software Development Academy

Sortowanie szybkie - partycjonowanie

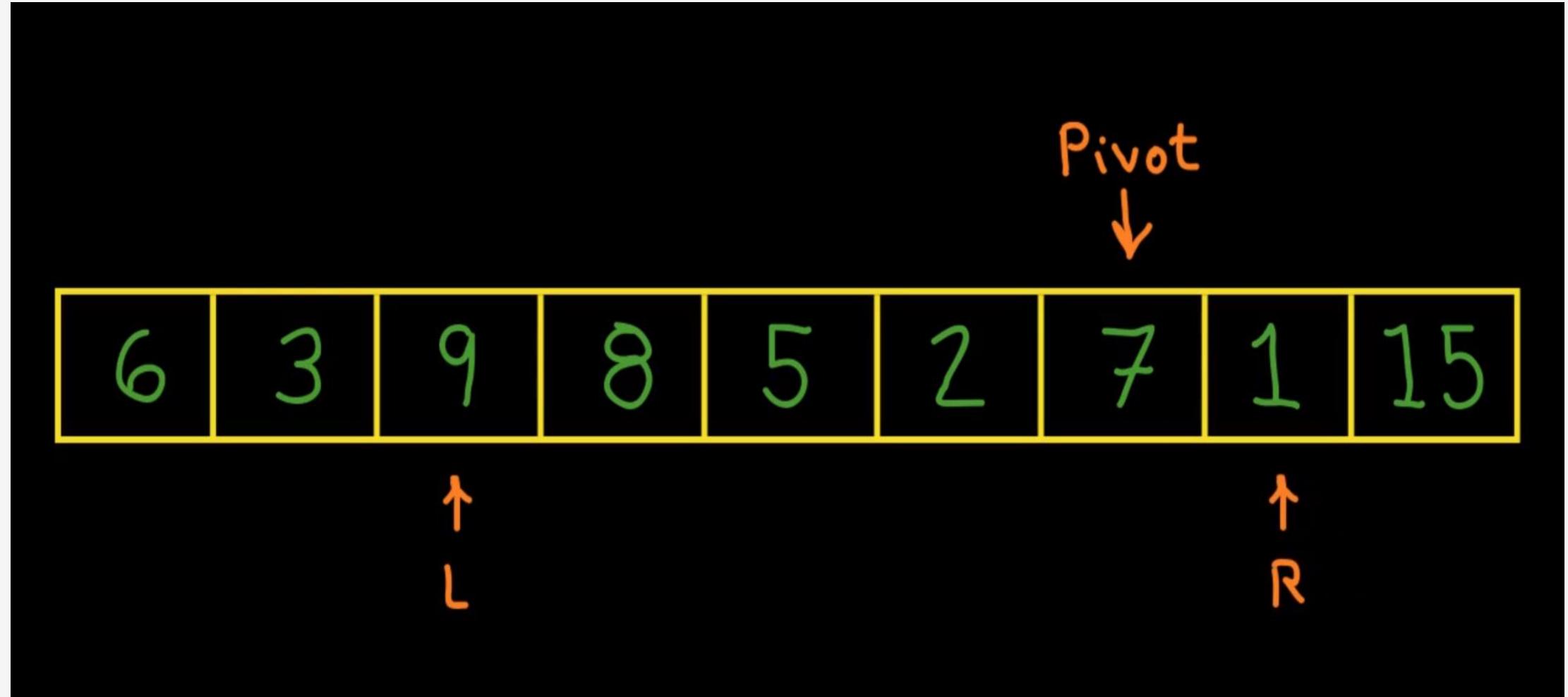


Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Sortowanie szybkie - partycjonowanie

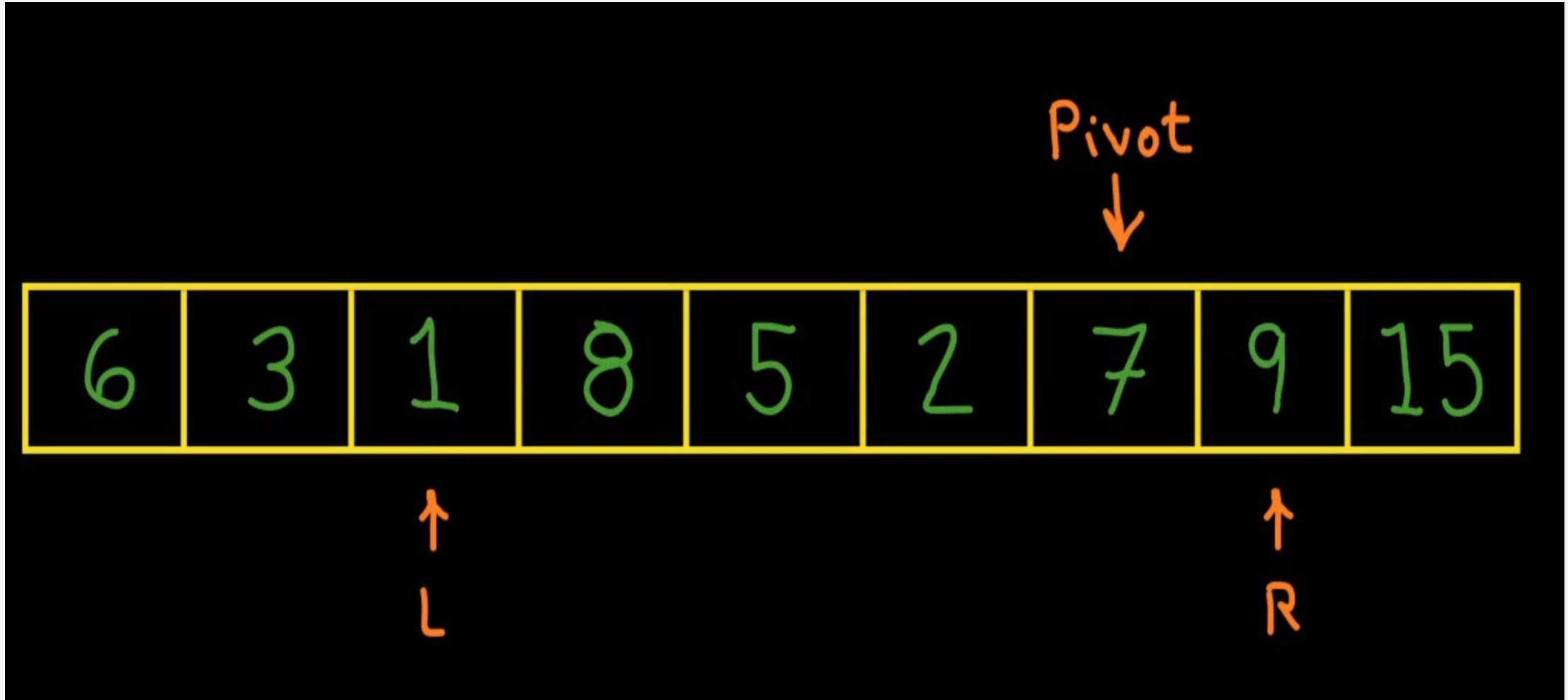


Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Sortowanie szybkie - partycjonowanie

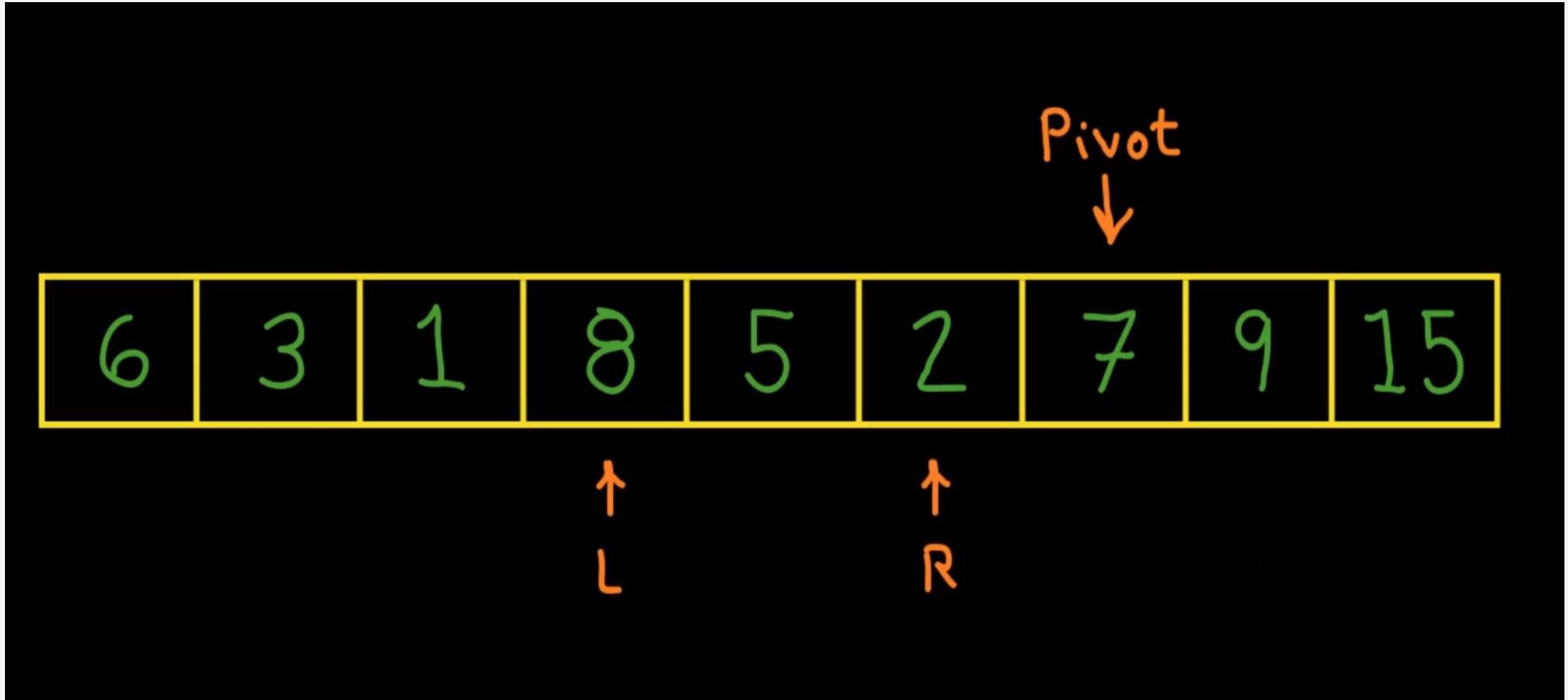


Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Sortowanie szybkie - partycjonowanie

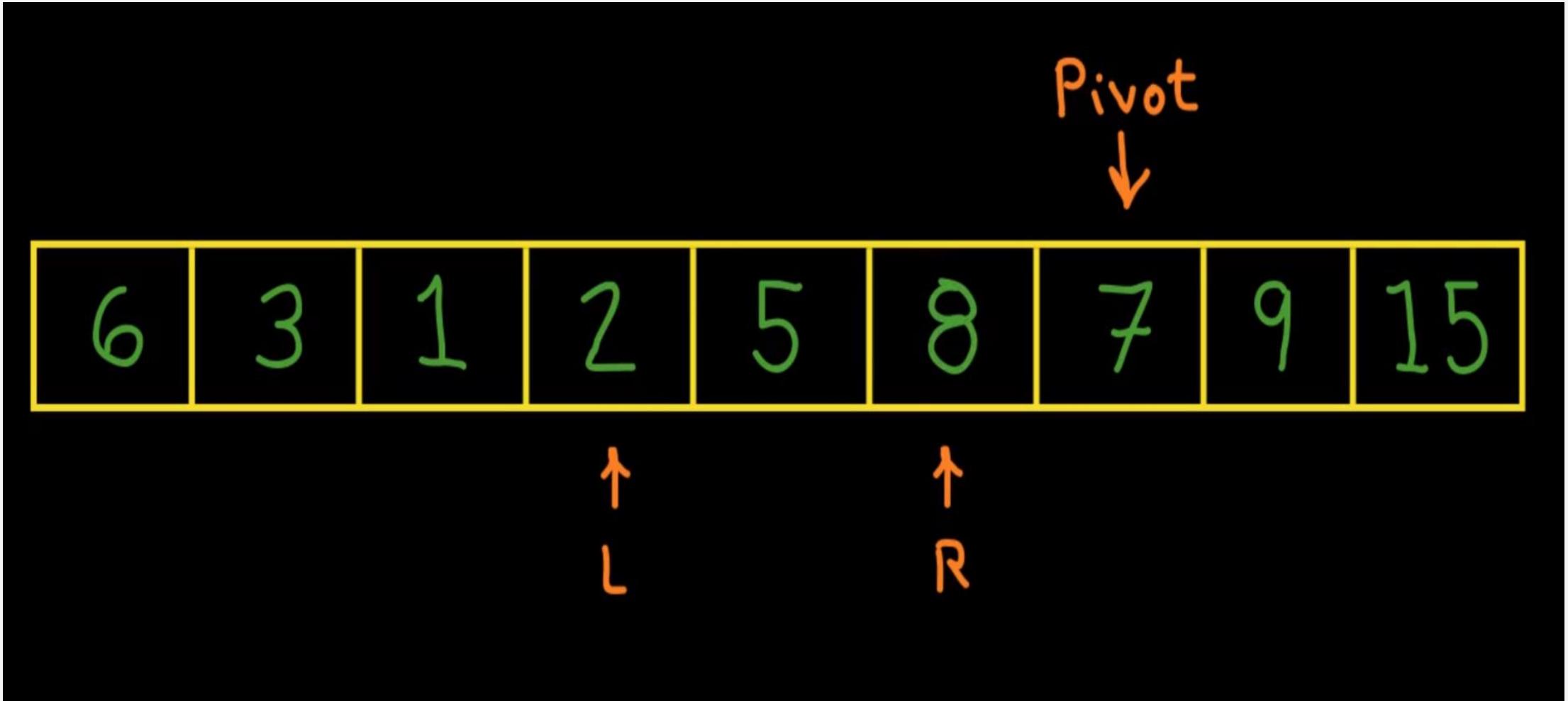


Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Sortowanie szybkie - partycjonowanie

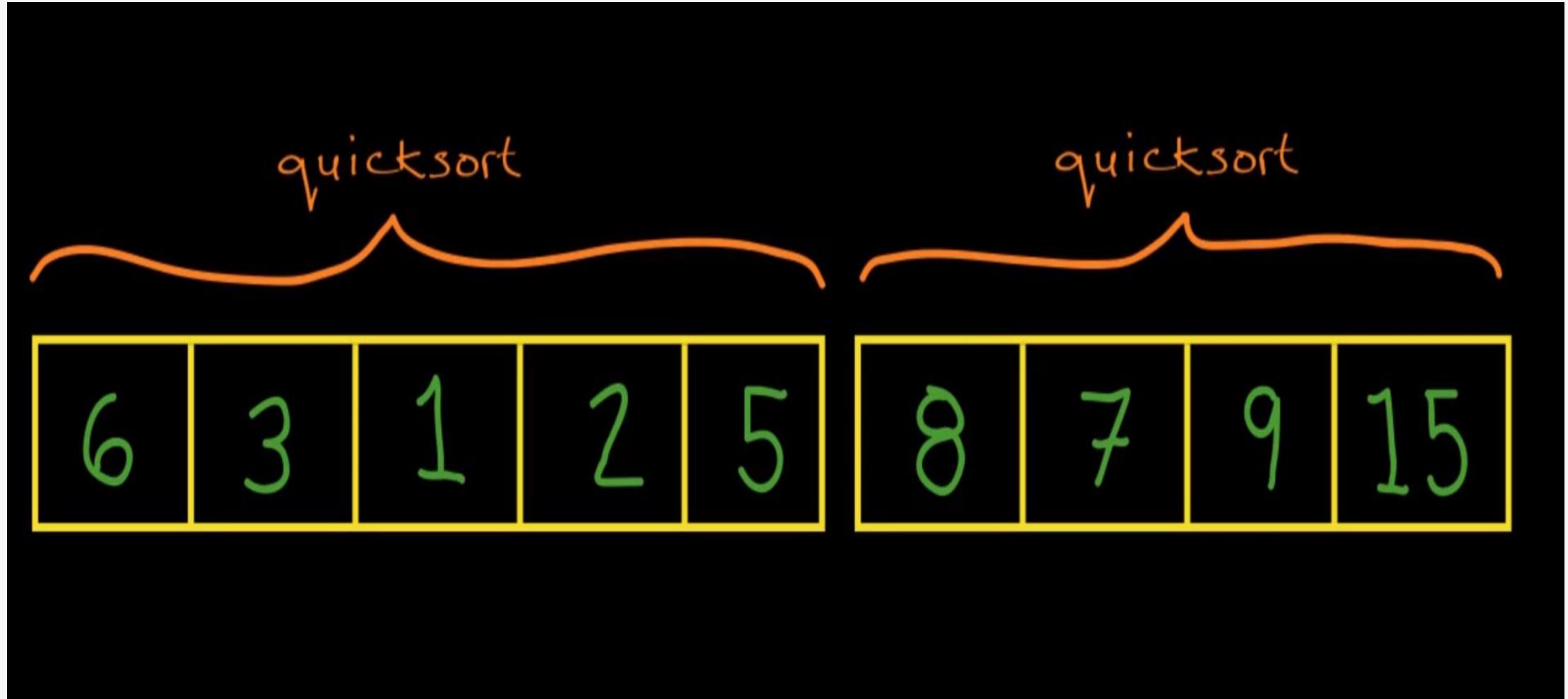


Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Sortowanie szybkie - partycjonowanie



Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Sortowanie szybkie (ang. Quick sort)

Średnia złożoność czasowa algorytmu:

$$O(n) = \log n * n$$

Występują skrajne przypadki (tablica posortowana w odwrotnej kolejności), wówczas złożoność wynosi:

$$O(n)=n * n$$

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Sortowanie przez scalanie (ang. Merge sort)

Rekurencyjny algorytm sortowania danych, stosujący metodę *dziel i zwyciężaj*.

Wyróżnić można 3 podstawowe kroki:

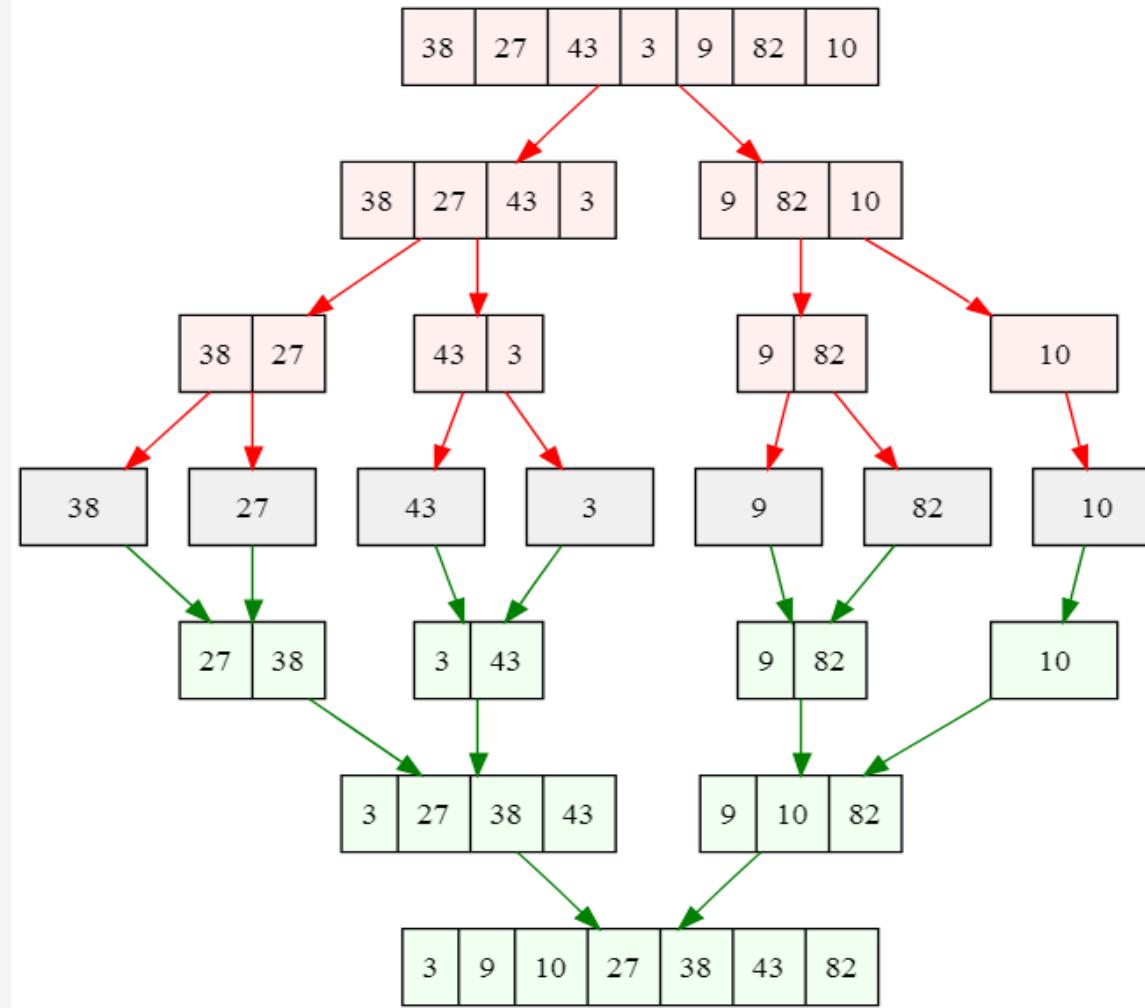
- Podziel zestaw danych na dwie równe części
- Zastosuj sortowanie przez scalanie dla każdej oddzielnie, chyba że pozostał już tylko jeden element
- Połącz posortowane podciągi w jeden ciąg posortowany

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Sortowanie przez scalanie (ang. Merge sort)

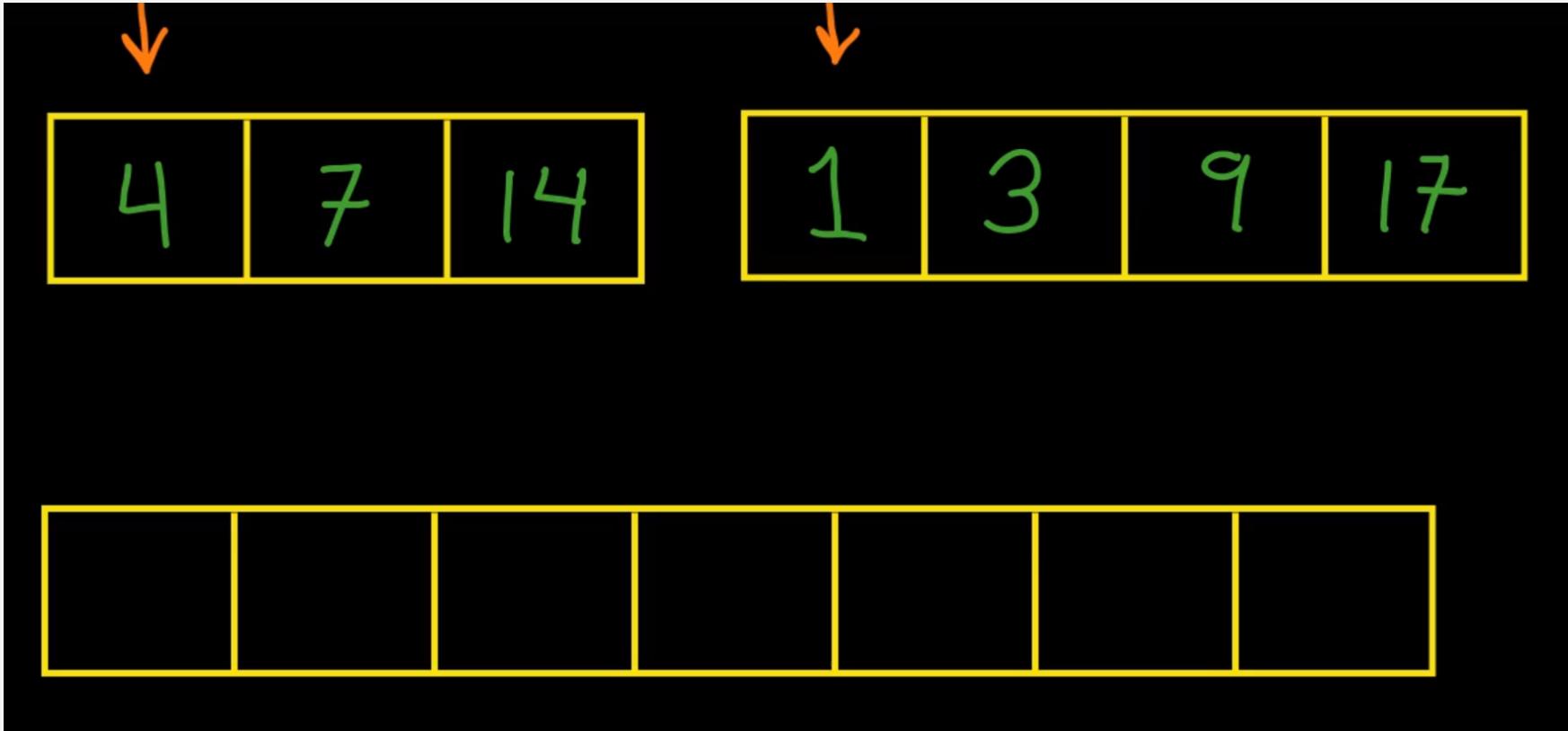


Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Sortowanie przez scalanie (ang. Merge sort)

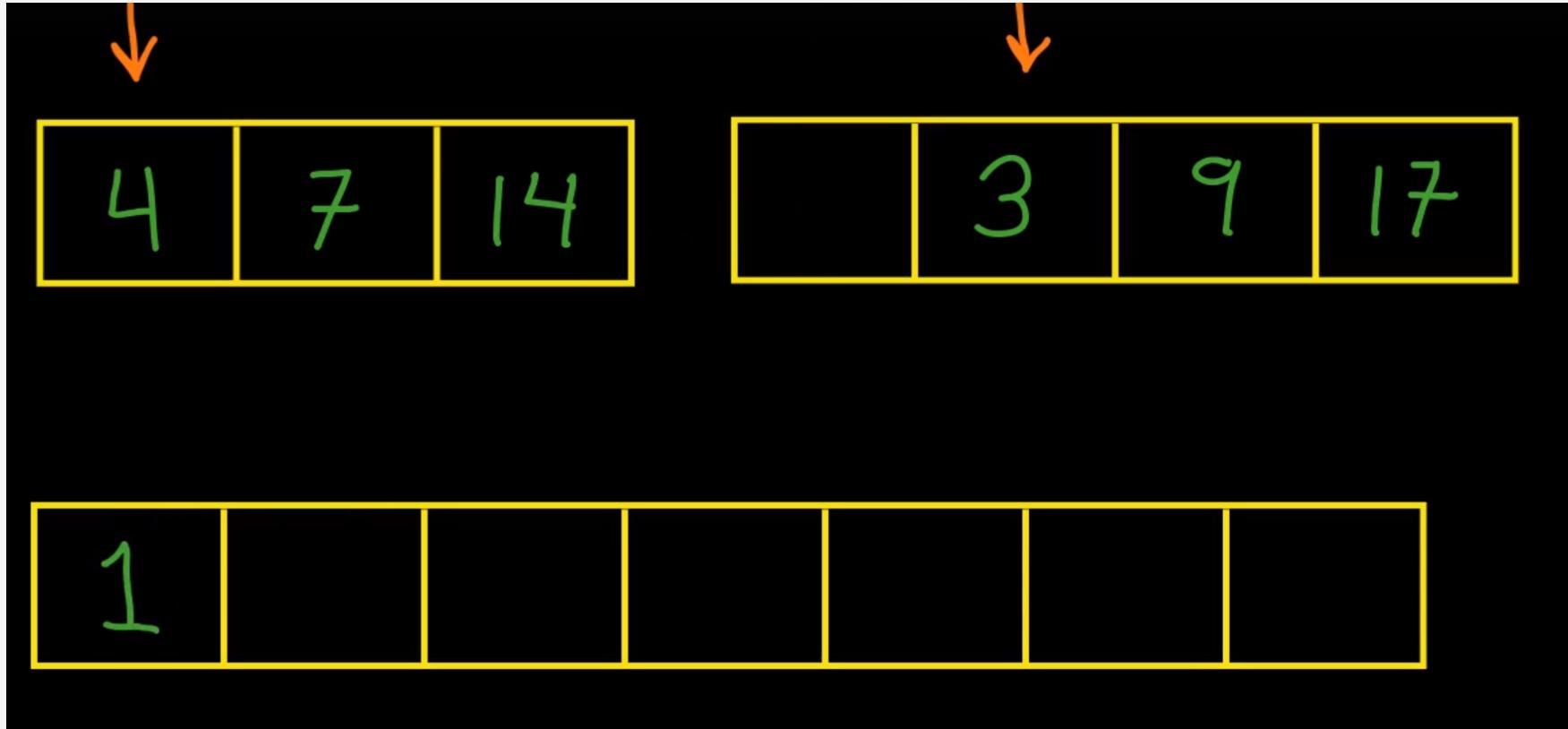


Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Sortowanie przez scalanie (ang. Merge sort)

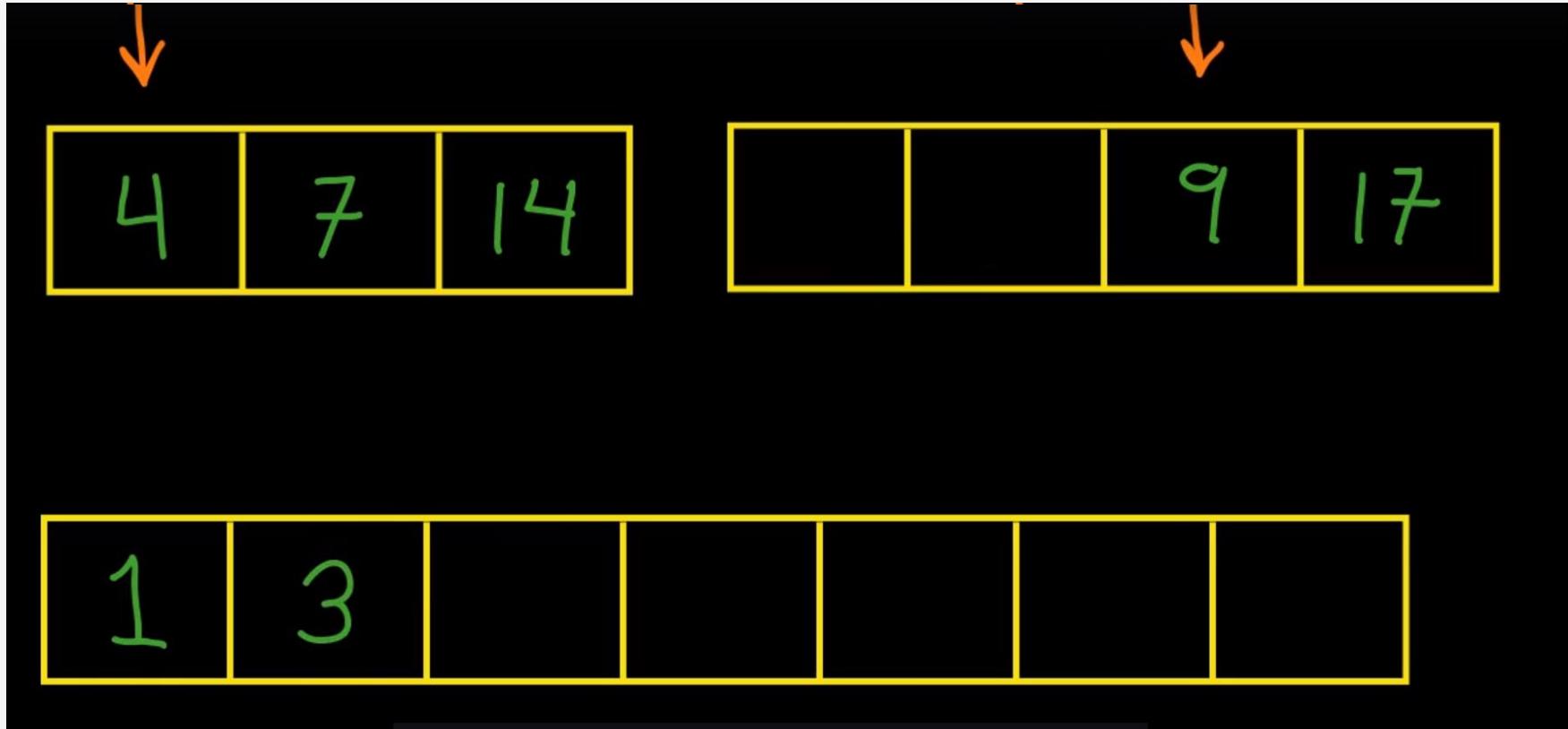


Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Sortowanie przez scalanie (ang. Merge sort)

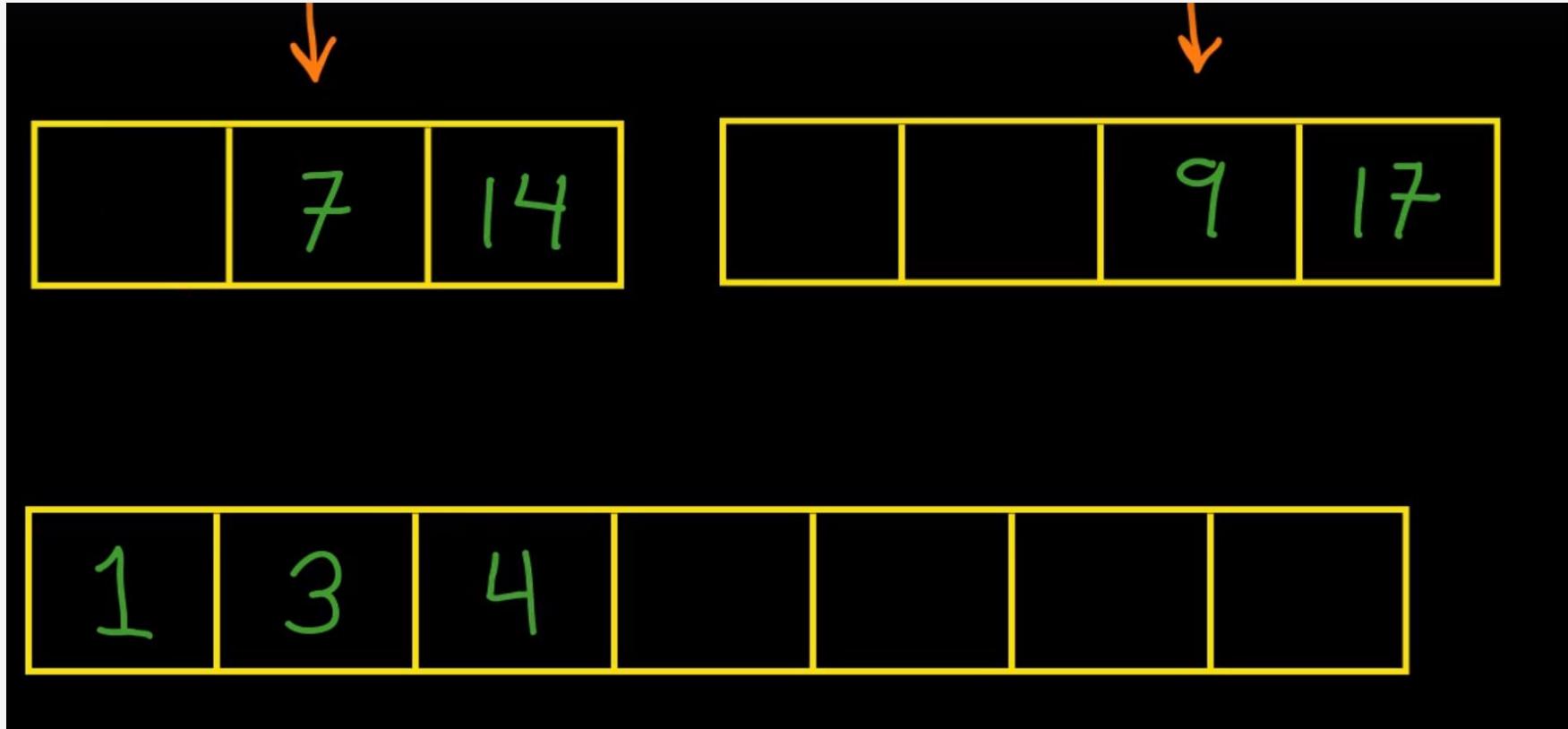


Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Sortowanie przez scalanie (ang. Merge sort)

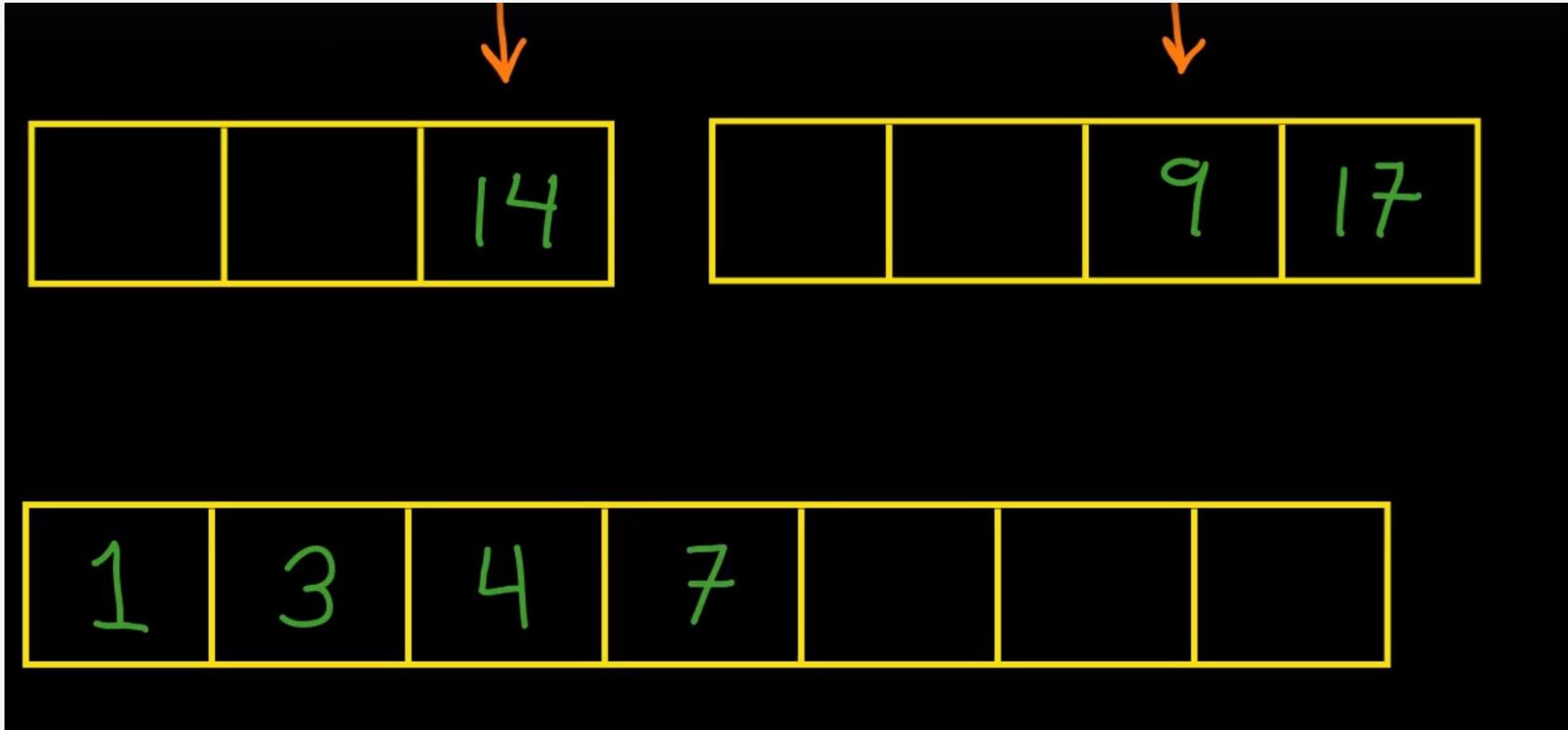


Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Sortowanie przez scalanie (ang. Merge sort)

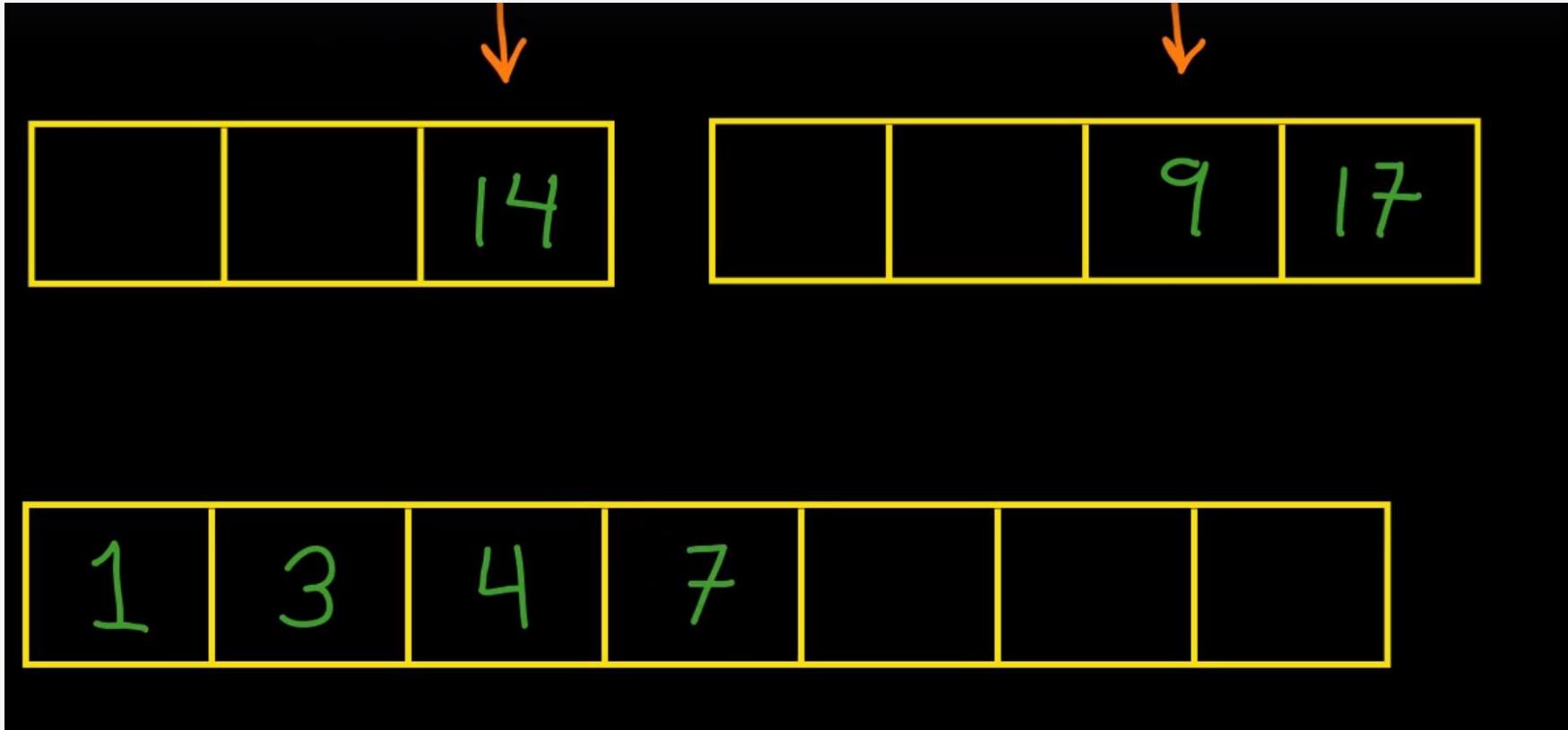


Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Sortowanie przez scalanie (ang. Merge sort)

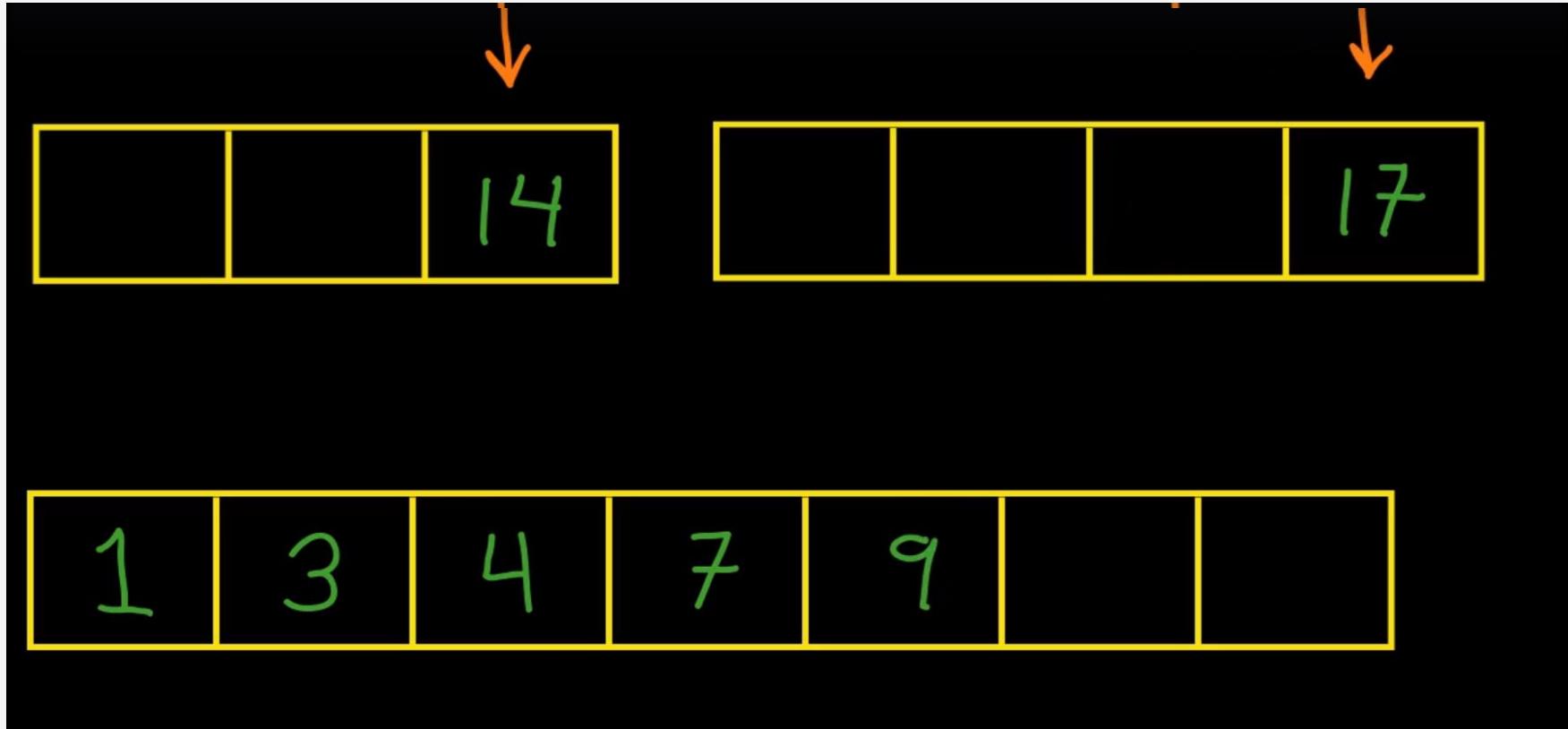


Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Sortowanie przez scalanie (ang. Merge sort)

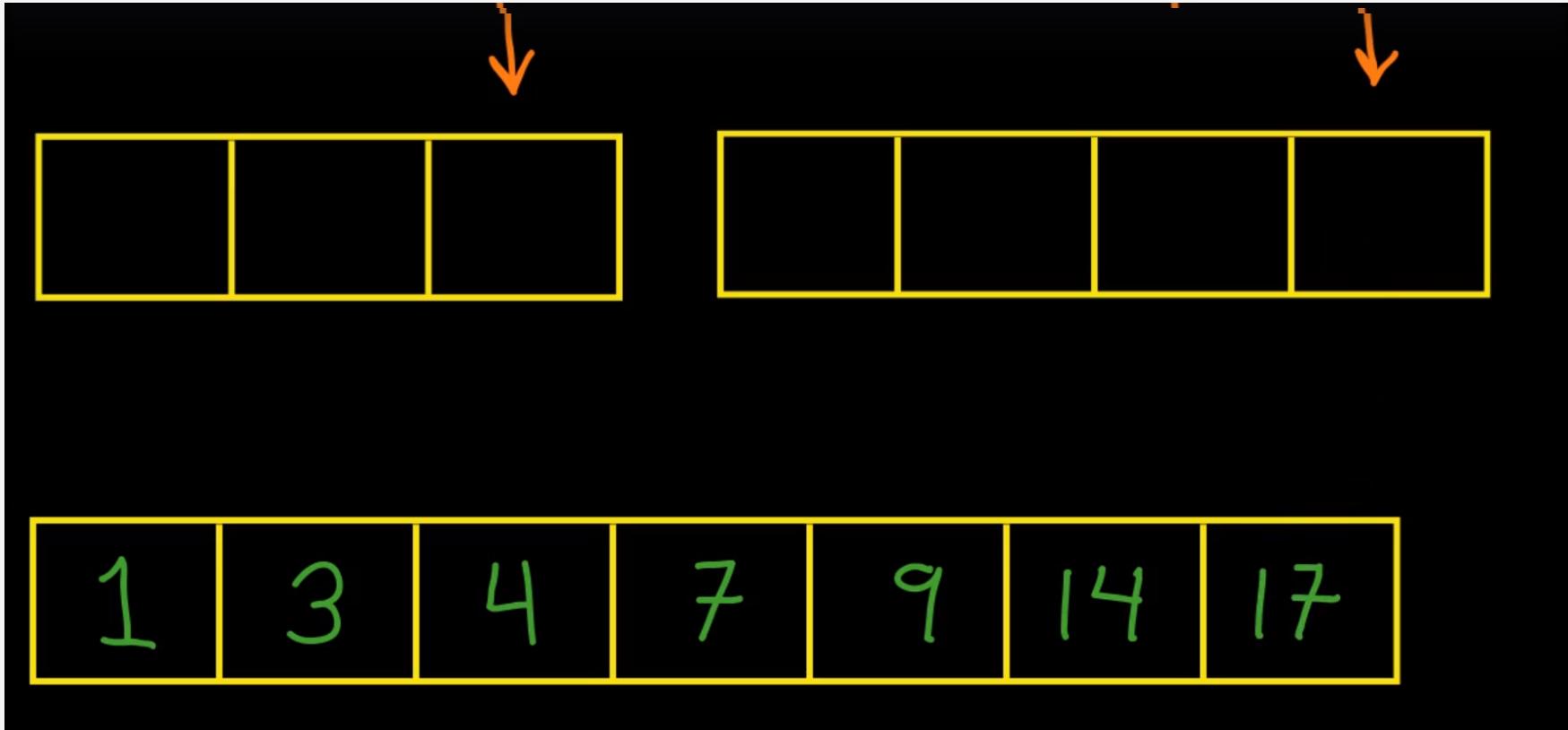


Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Sortowanie przez scalanie (ang. Merge sort)



Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Sortowanie przez scalanie charakterystyka

Złożoność algorytmu wynosi:

- $O(n) = n * \log n$

Jest to sortowanie *stabilne* (czyli złożoność jest taka sama dla przypadków średnich/optymistycznych/pesymistycznych)

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy

Rekurencja – ZADANIE DOMOWE



Zapoznaj się z algorytmem wyznaczania największego wspólnego dzielnika dwóch liczb (algorytm euklidesa):

<https://www.geeksforgeeks.org/euclidean-algorithms-basic-and-extended/>

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Drzewa binarne – super szybki wyszukiwanie danych

Binarne drzewo poszukiwań ([ang.](#) *Binary Search Tree (BST)*) – dynamiczna struktura danych będąca drzewem binarnym, w którym lewe poddrzewo każdego węzła zawiera wyłącznie elementy o kluczach mniejszych niż klucz węzła a prawe poddrzewo zawiera wyłącznie elementy o kluczach nie mniejszych niż klucz węzła. Węzły, oprócz klucza, przechowują wskaźniki na swojego lewego i prawego syna oraz na swojego ojca.

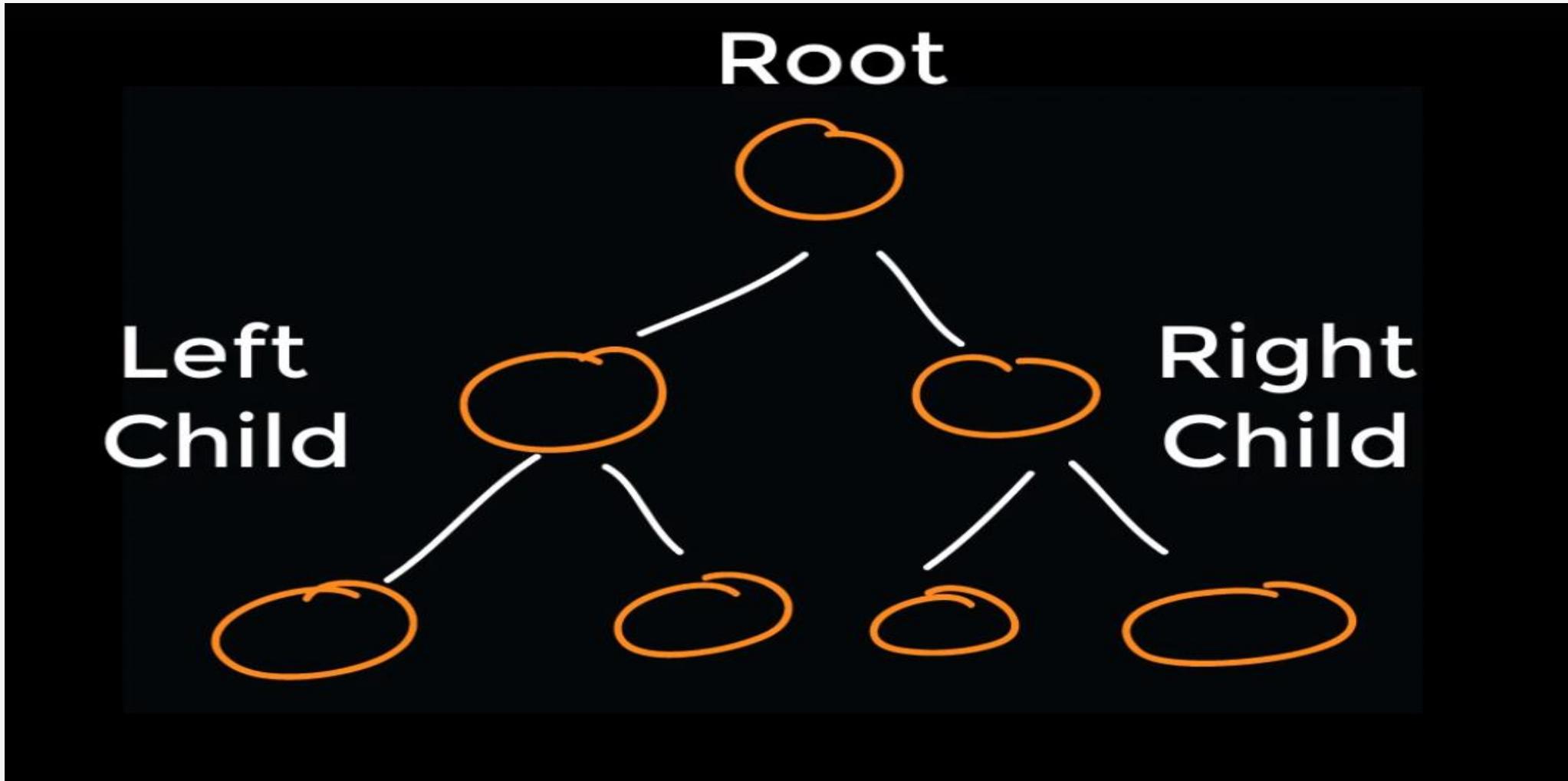
https://pl.wikipedia.org/wiki/Binarne_drzewo_poszukiwa%C5%84

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Drzewa binarne – struktura

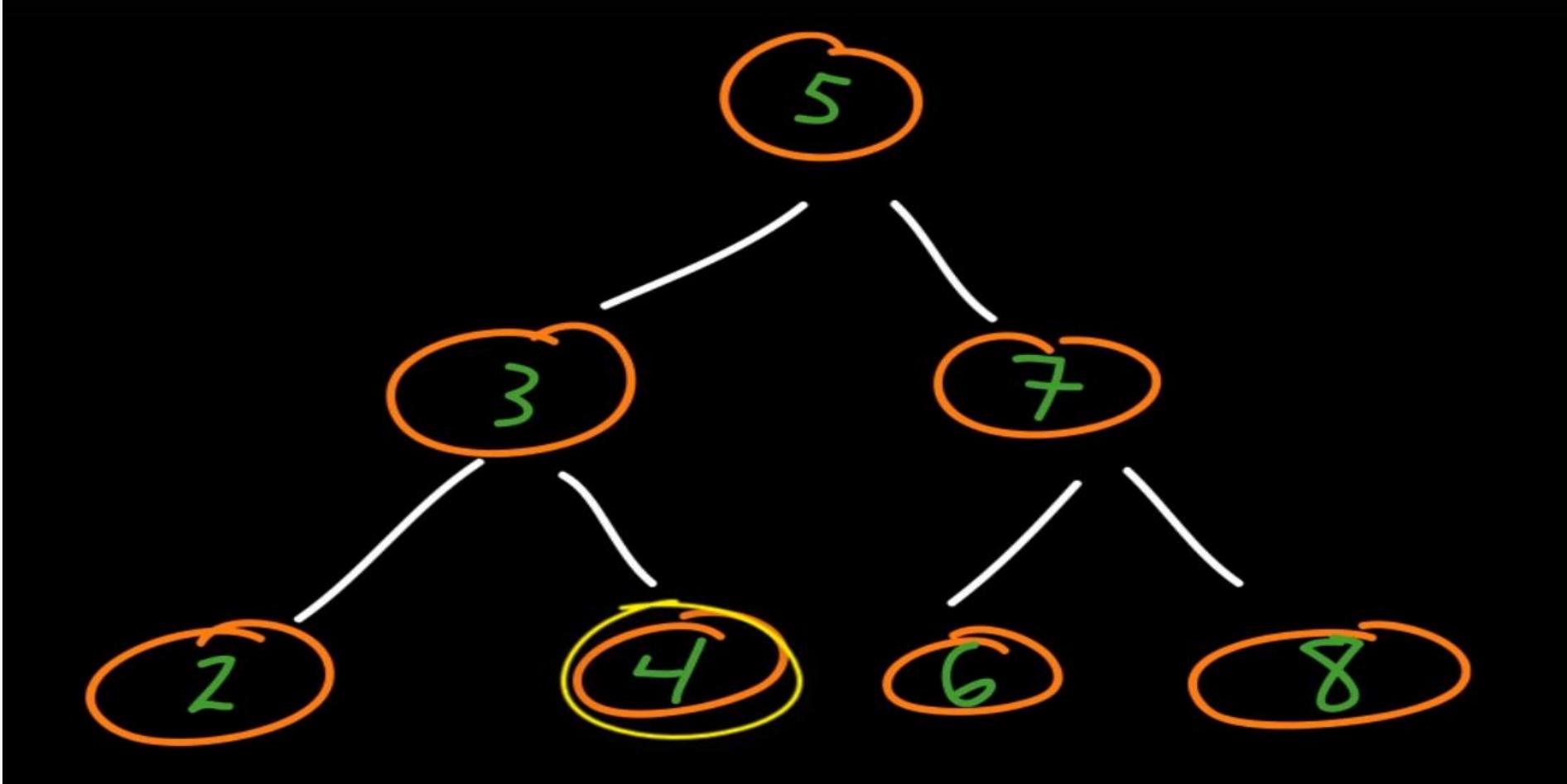


Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Drzewa binarne – jak działa wyszukiwanie O(logn)

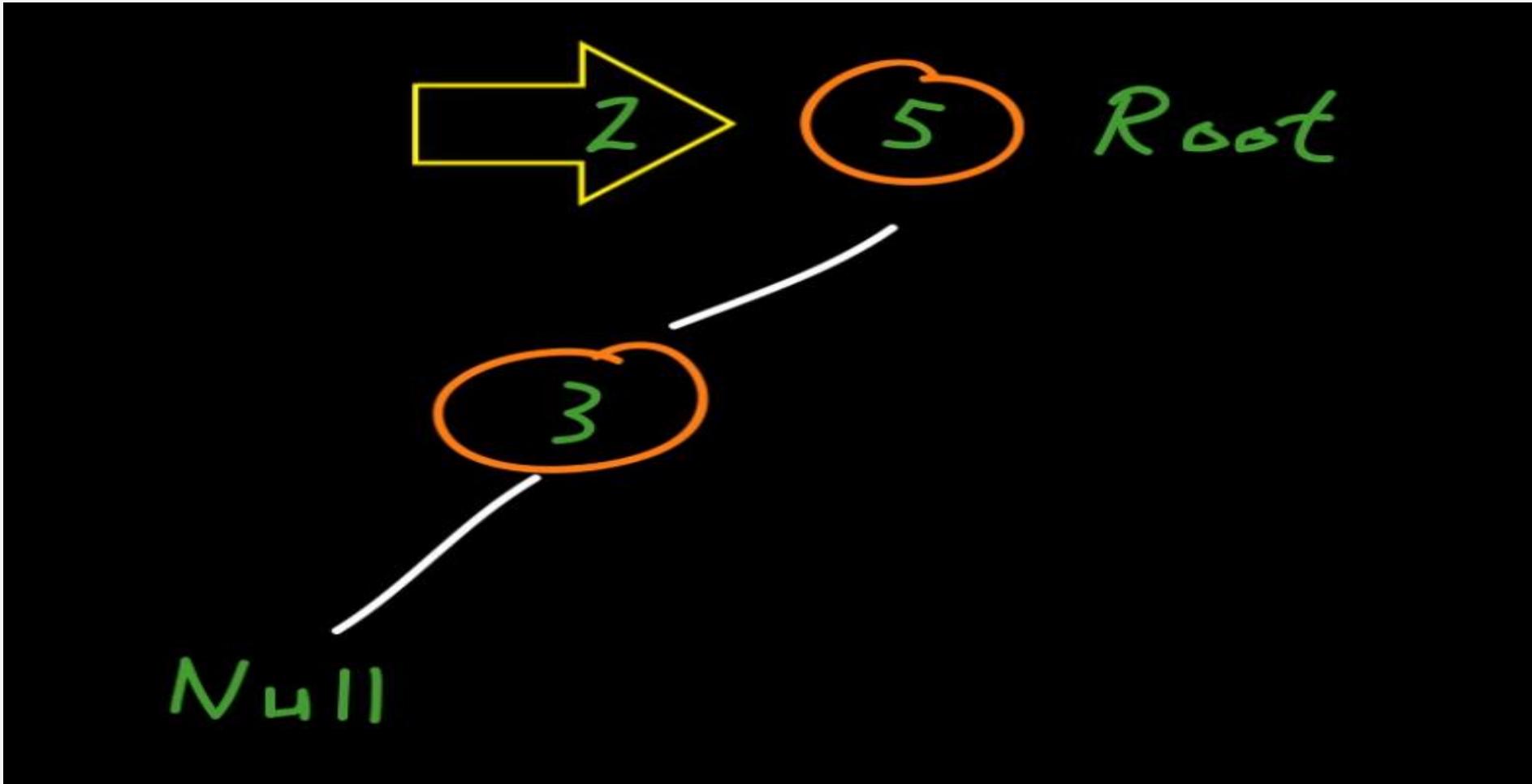


Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Drzewa binarne – jak działa dodawanie

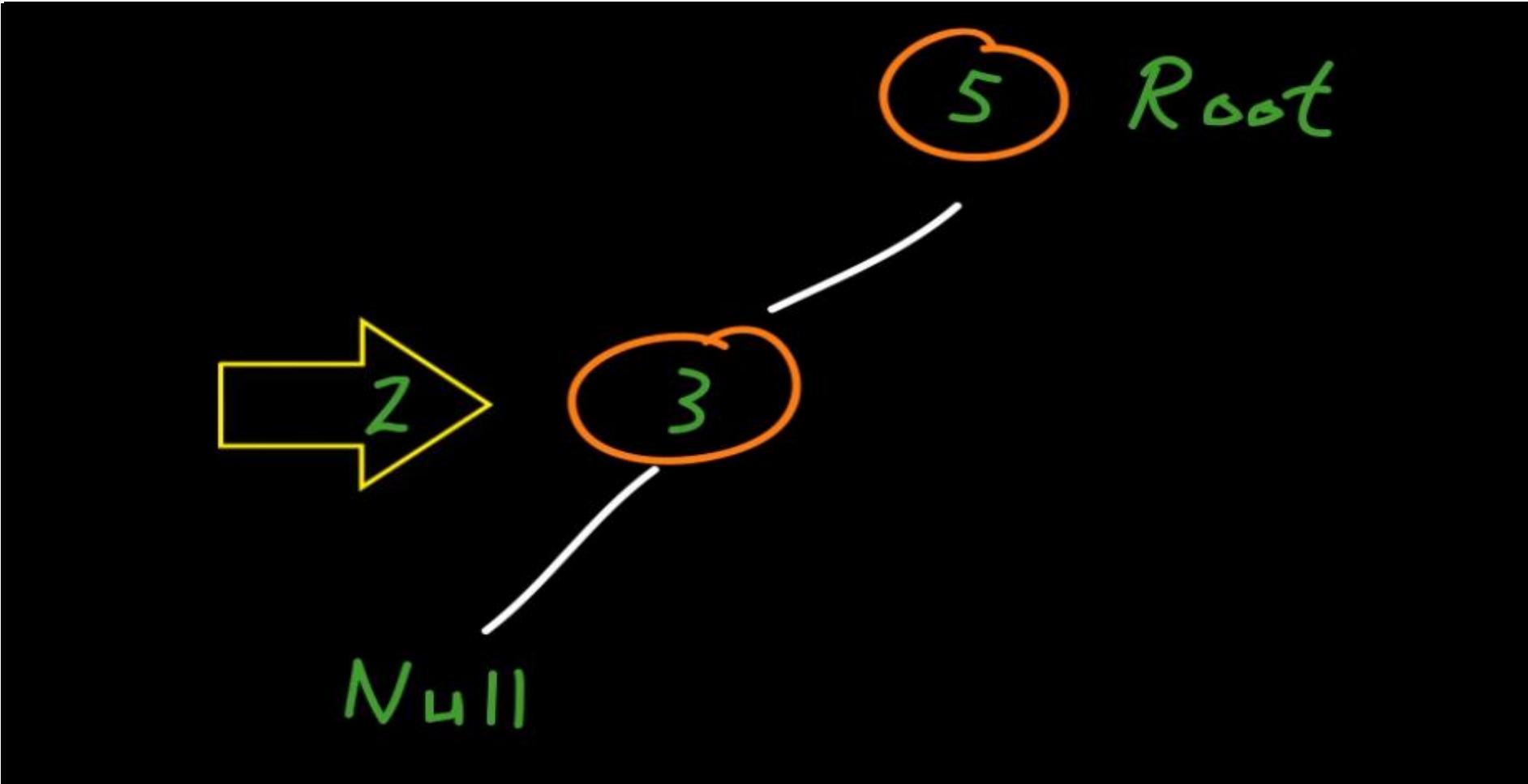


Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Drzewa binarne – jak działa dodawanie

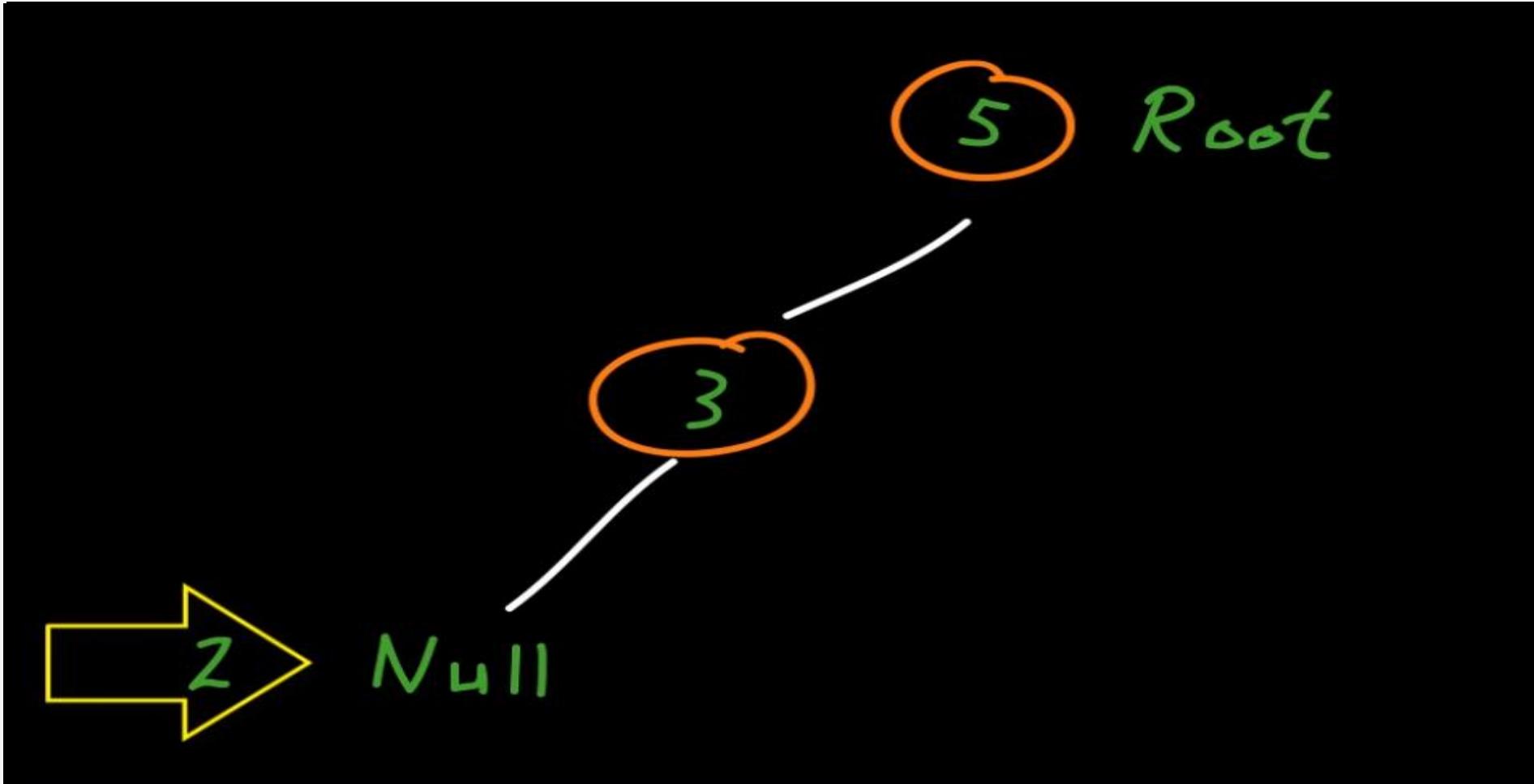


Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Drzewa binarne – jak działa dodawanie

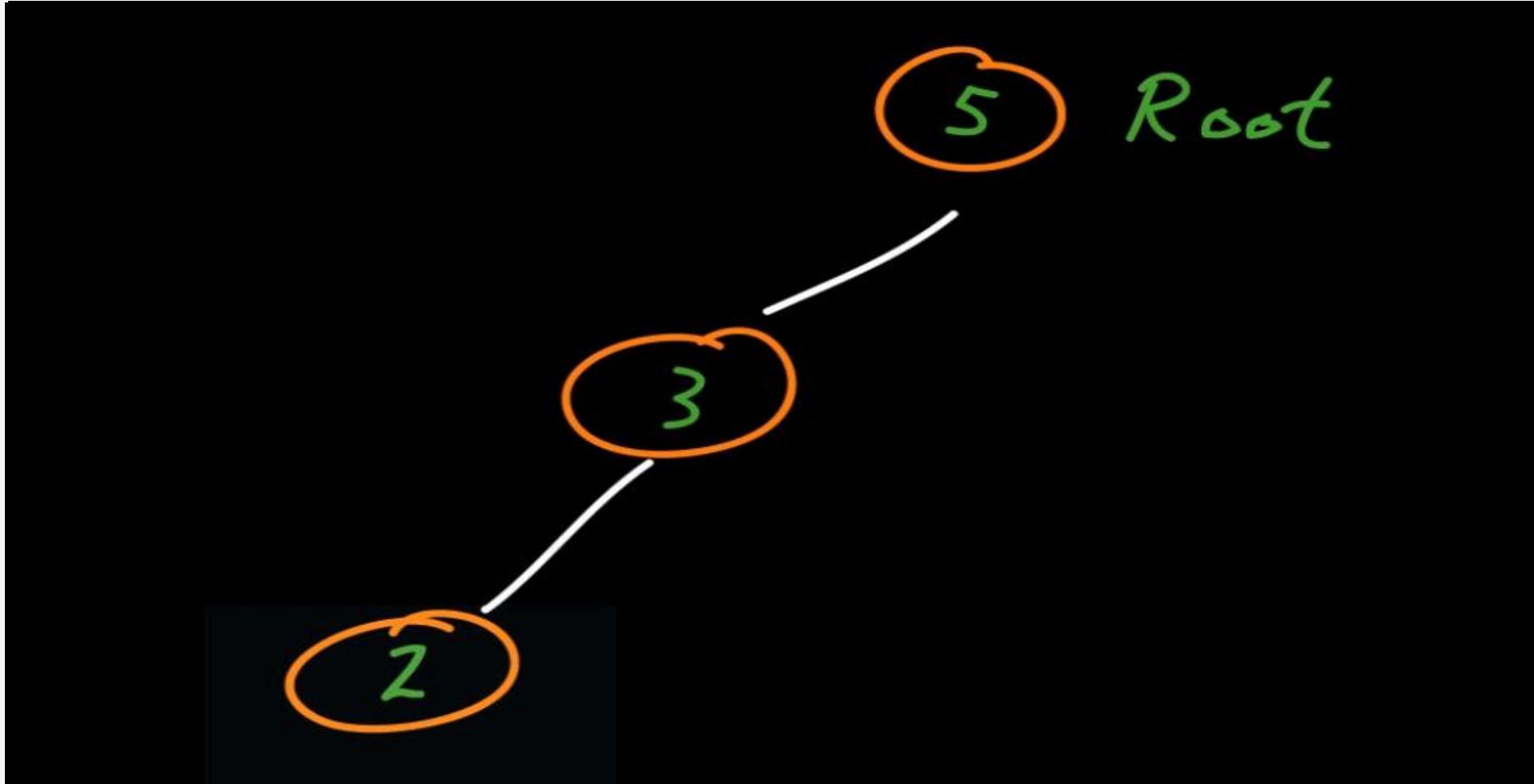


Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Drzewa binarne – jak działa dodawanie



Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Drzewa binarne – jak działa usuwanie

Podczas usuwania musimy rozpatrzyć 3 przypadki kiedy:

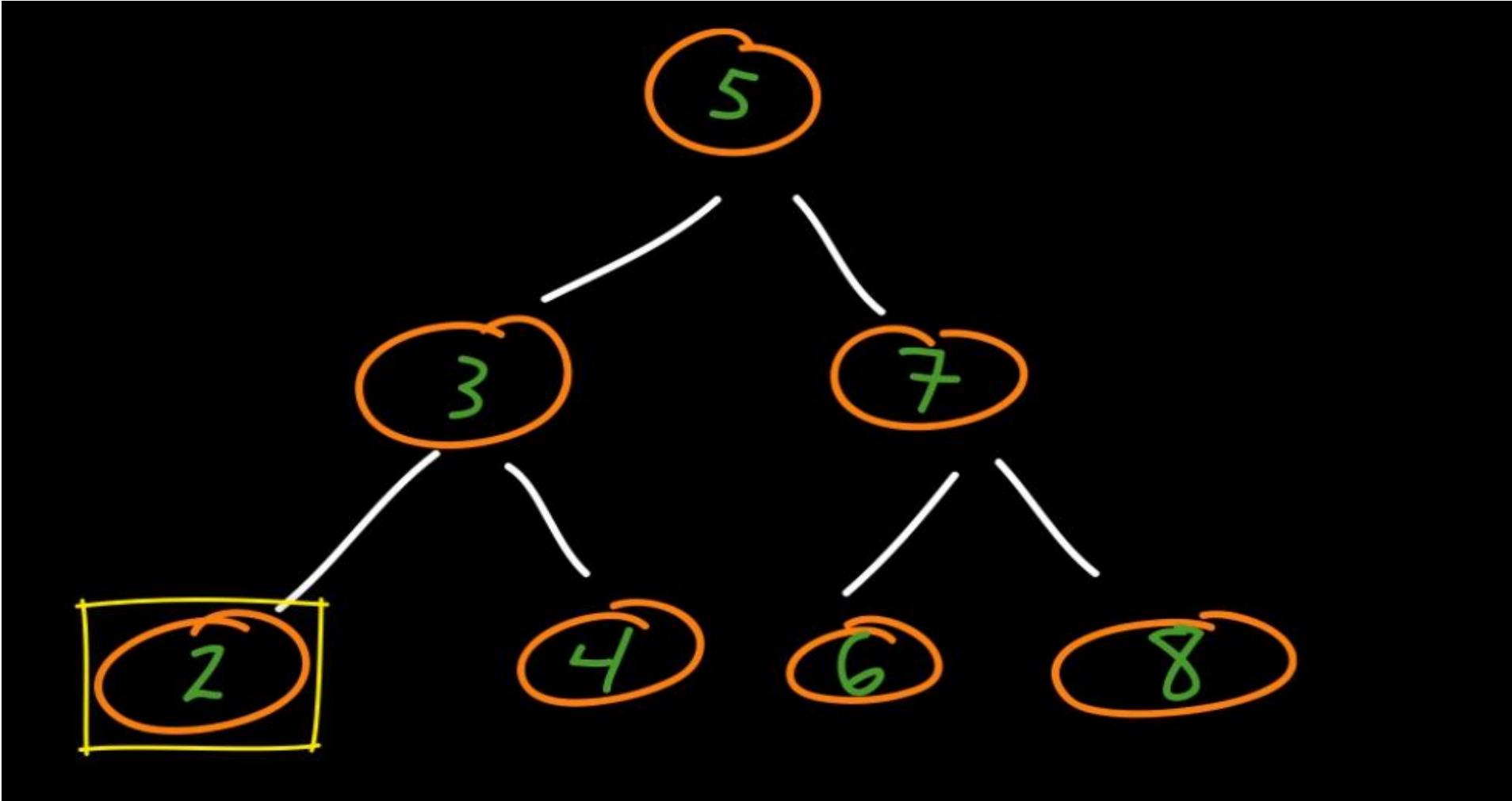
- usuwany element nie ma dzieci
- usuwany element ma jednego potomka
- usuwany element ma 2 dzieci

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Drzewa binarne – usuwanie – brak potomków

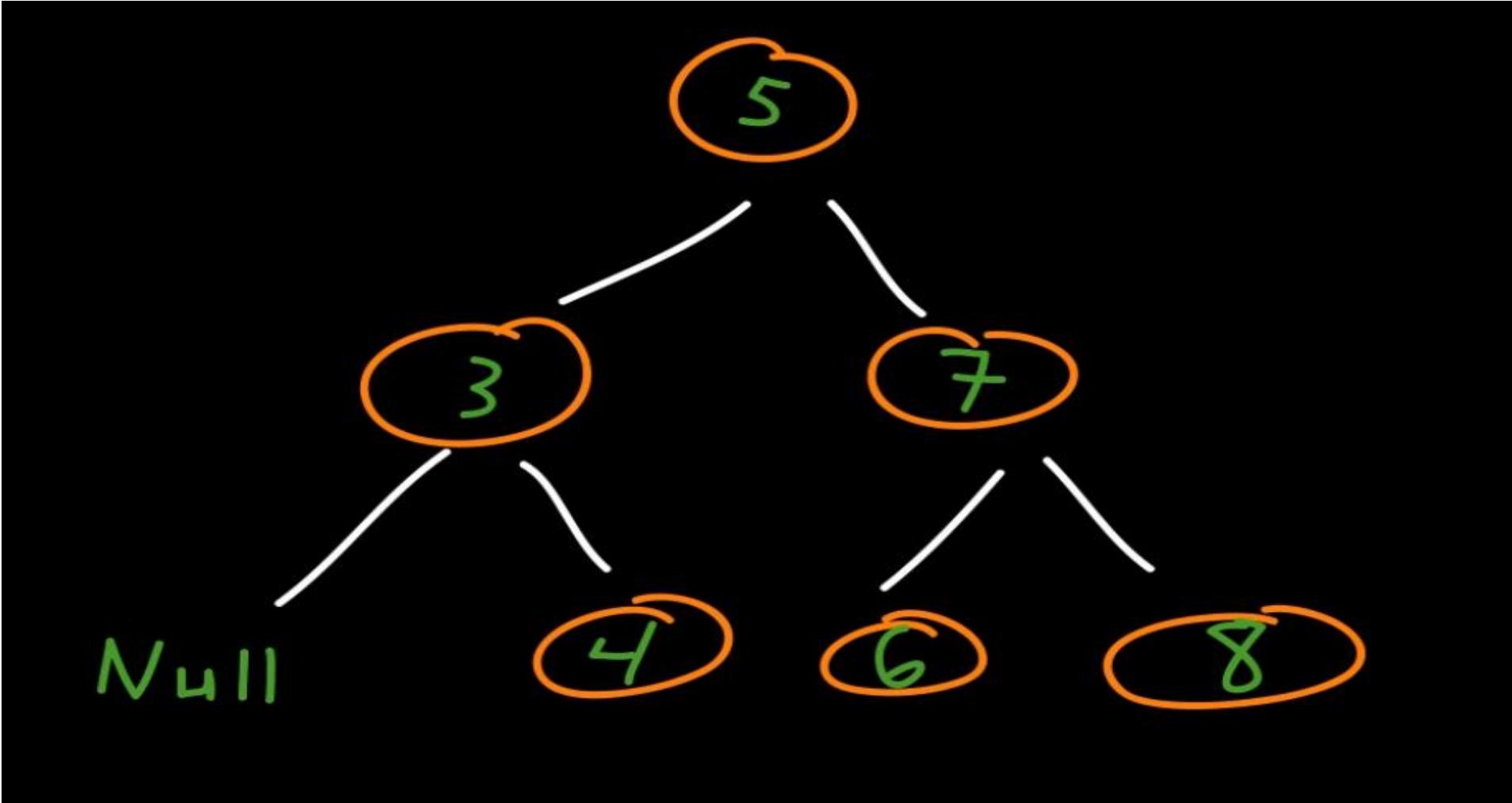


Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Drzewa binarne – usuwanie – brak potomków

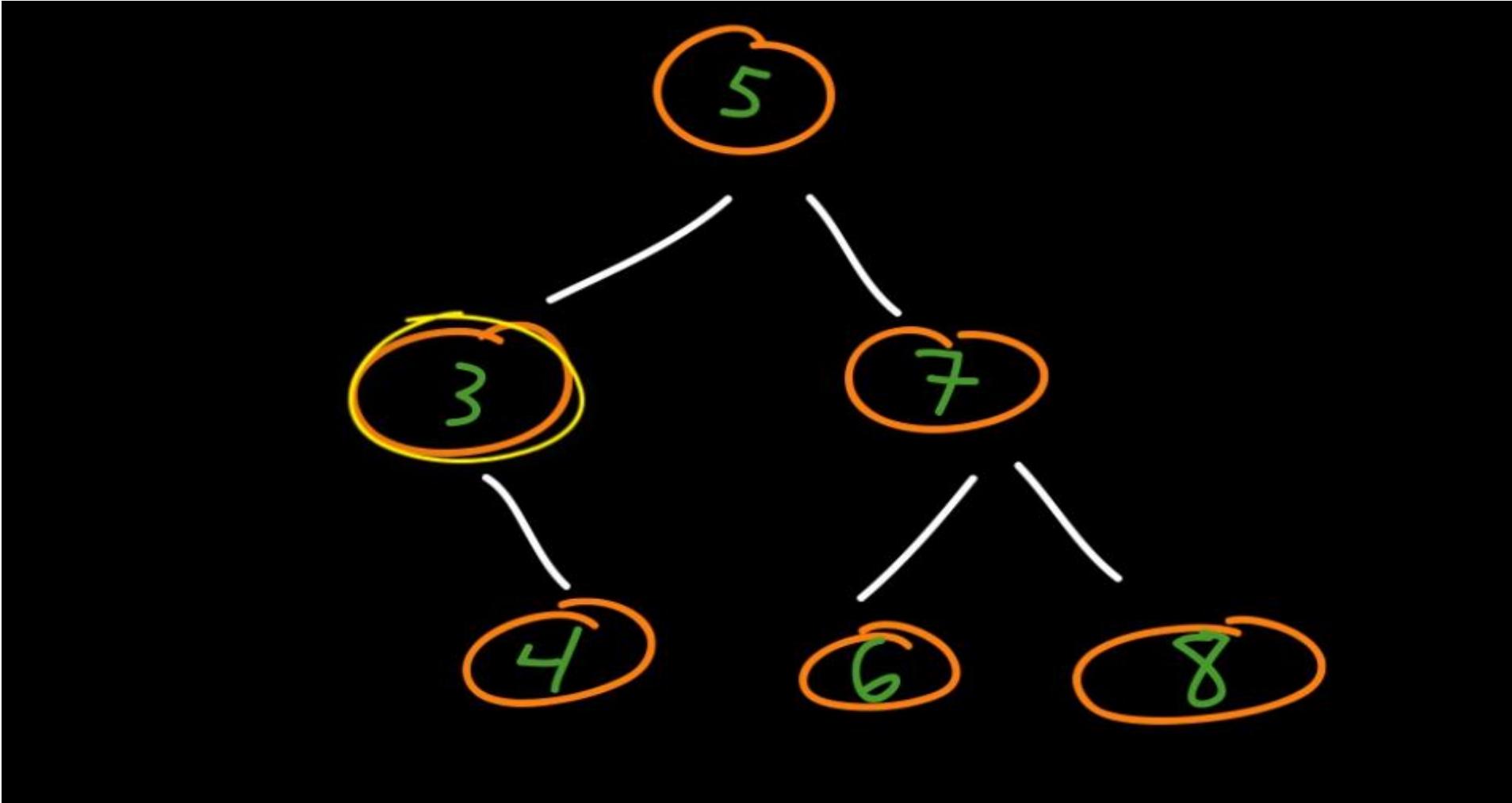


Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Drzewa binarne – usuwanie – 1 dziecko

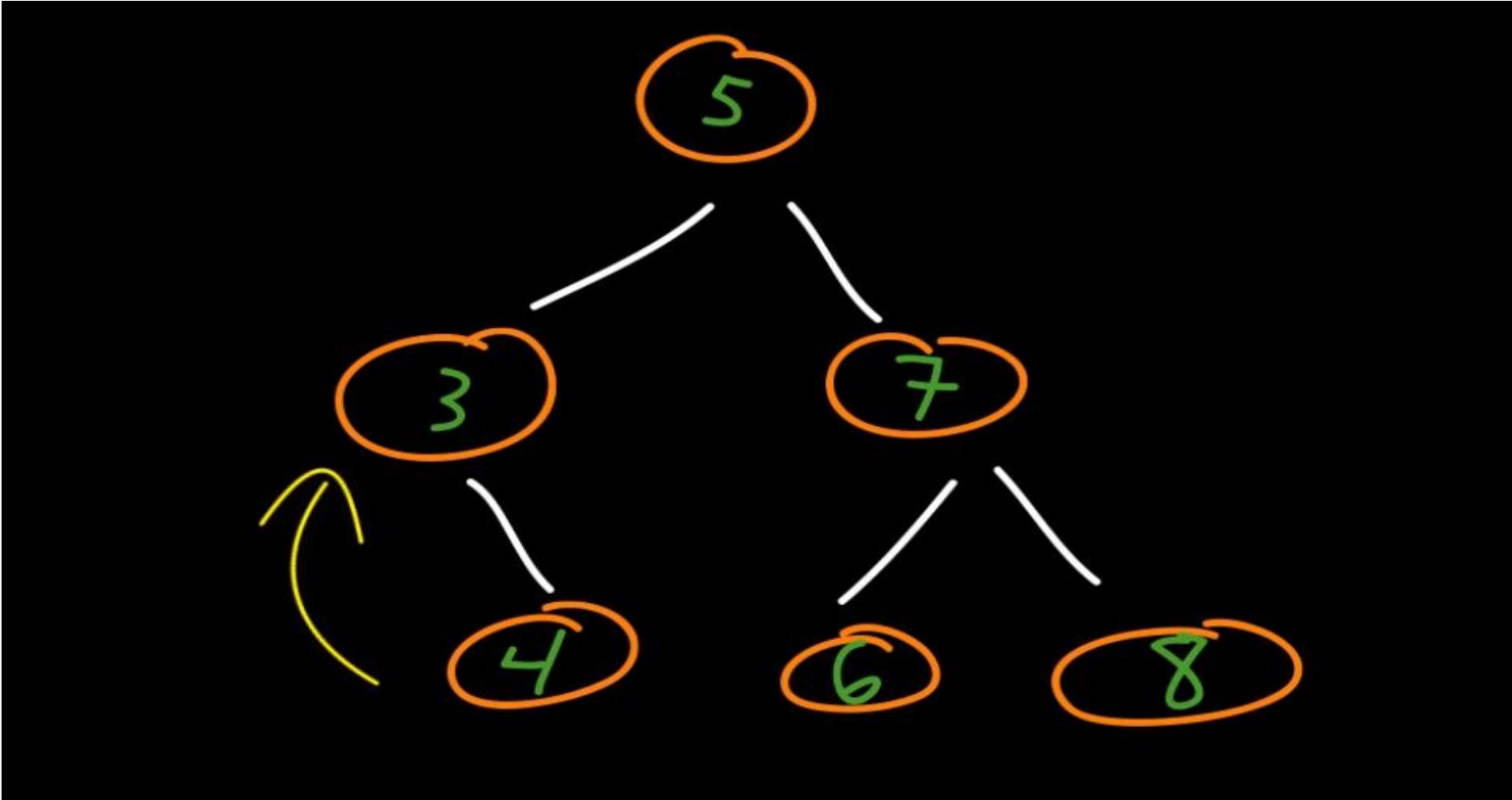


Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Drzewa binarne – usuwanie – 1 dziecko

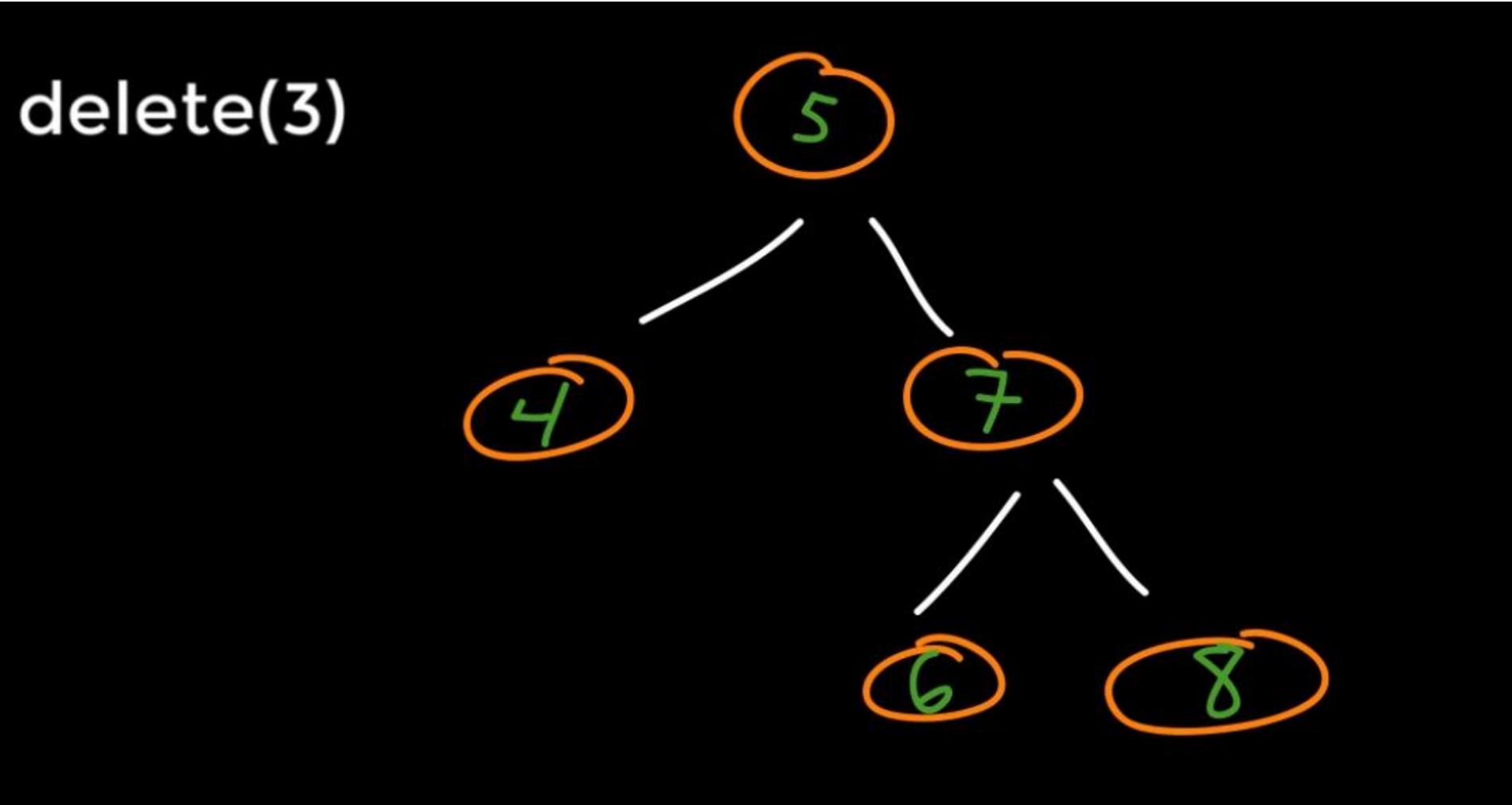


Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Drzewa binarne – usuwanie – 1 dziecko

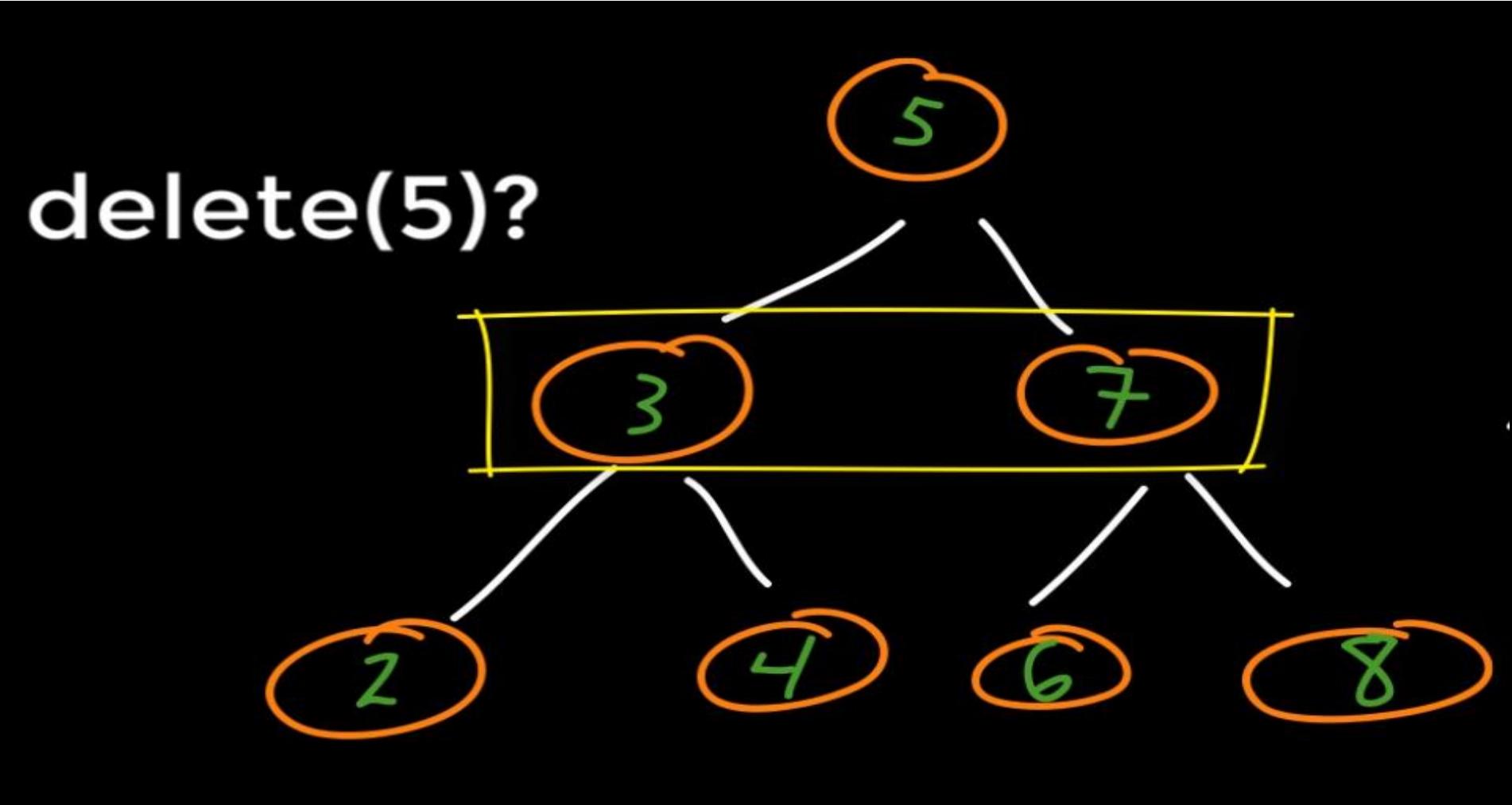


Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Drzewa binarne – usuwanie – 2 dzieci

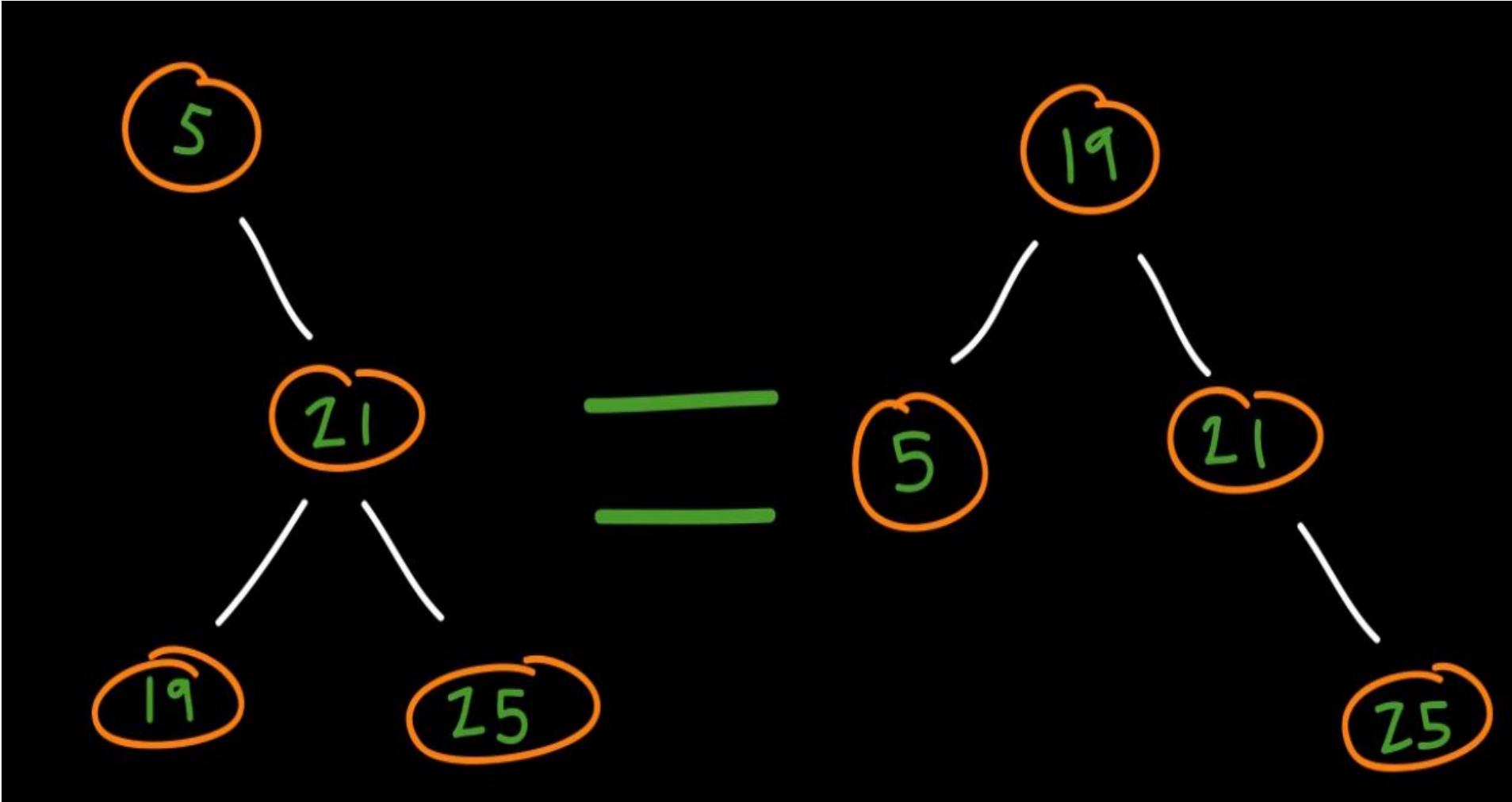


Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



BST – usuwanie – 1 drzewo -> kilka reprezentacji

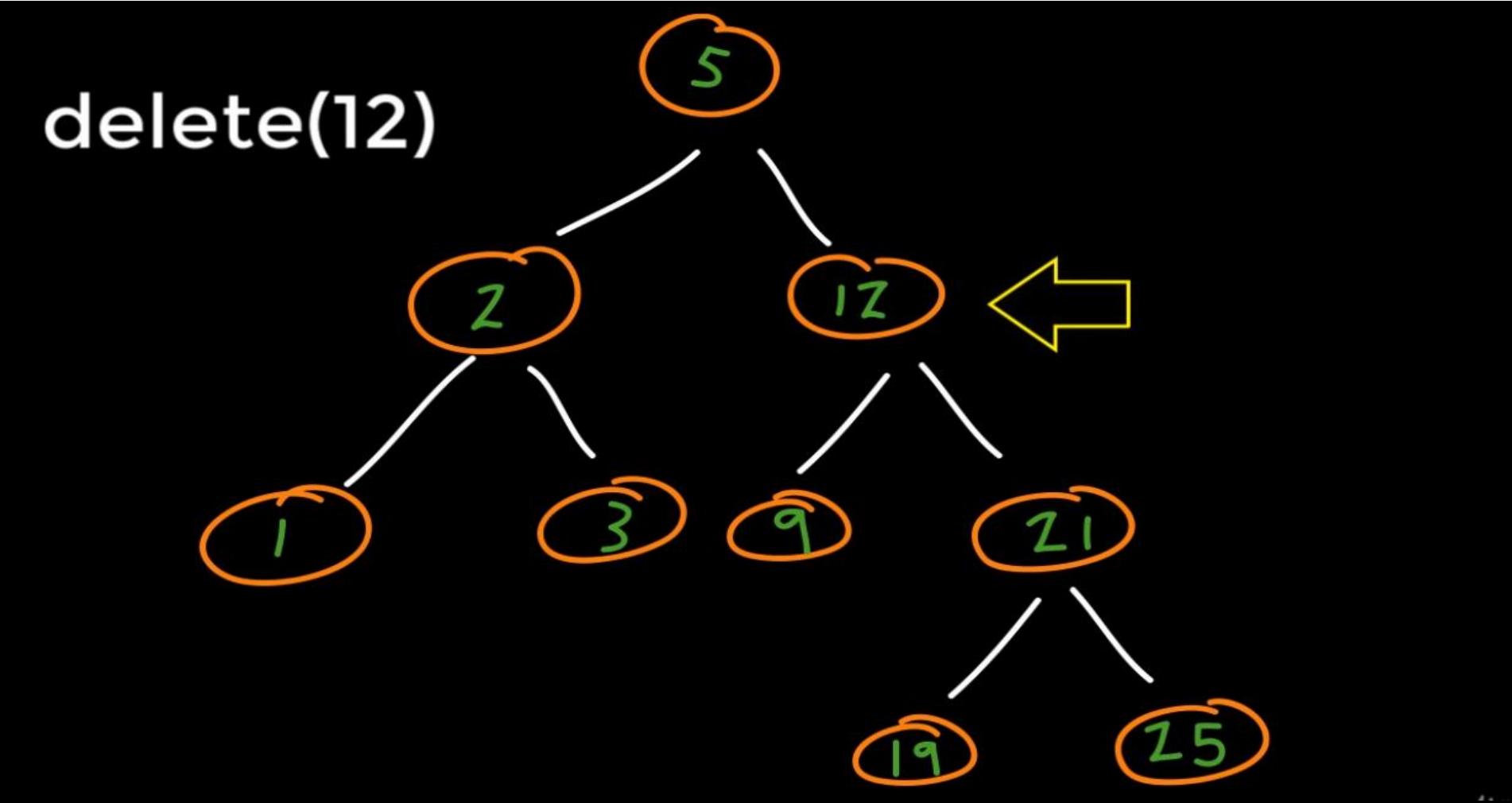


Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Drzewa binarne – usuwanie – 2 dzieci

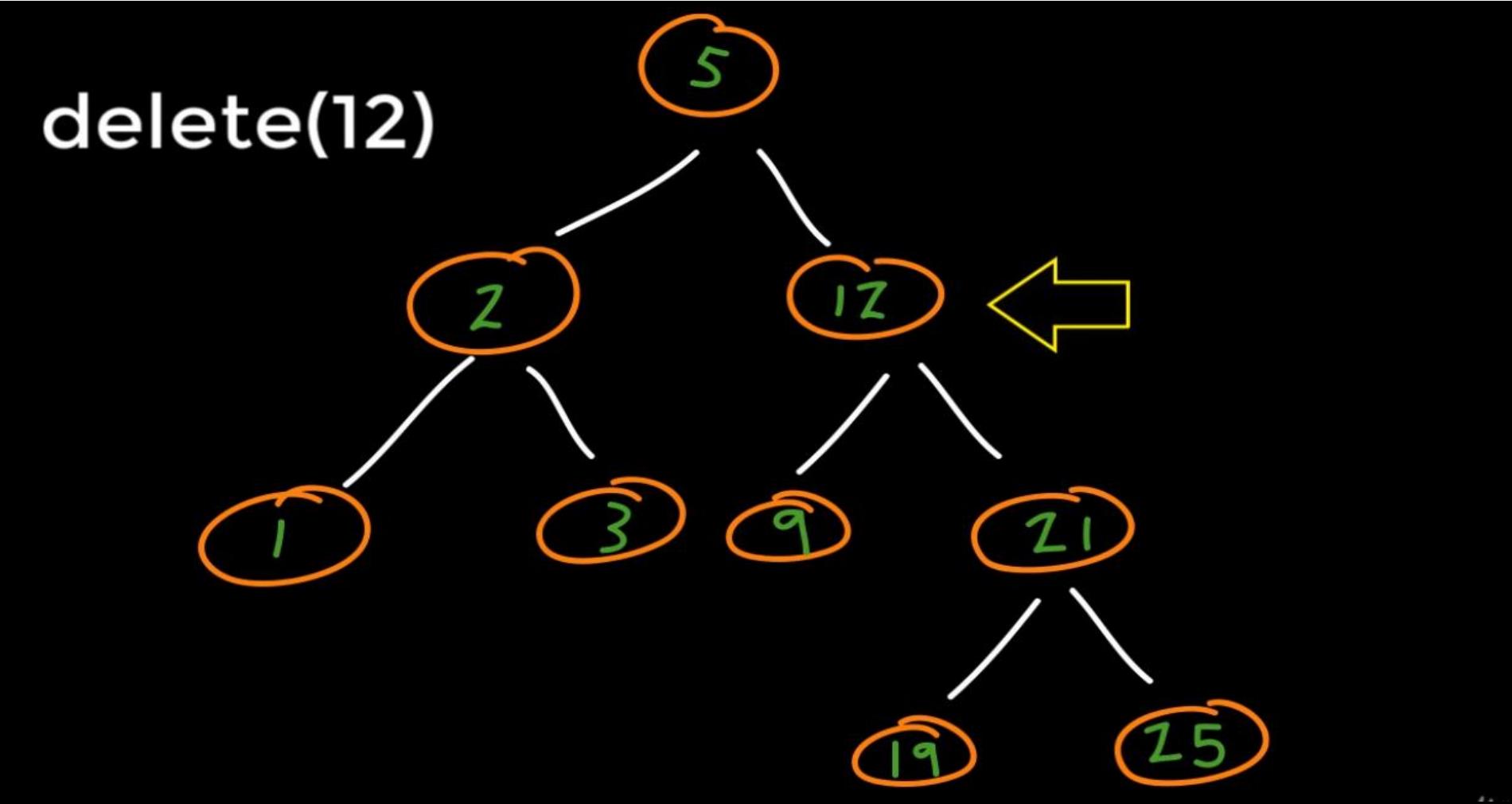


Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



BST – 2 dzieci – usuwamy 12



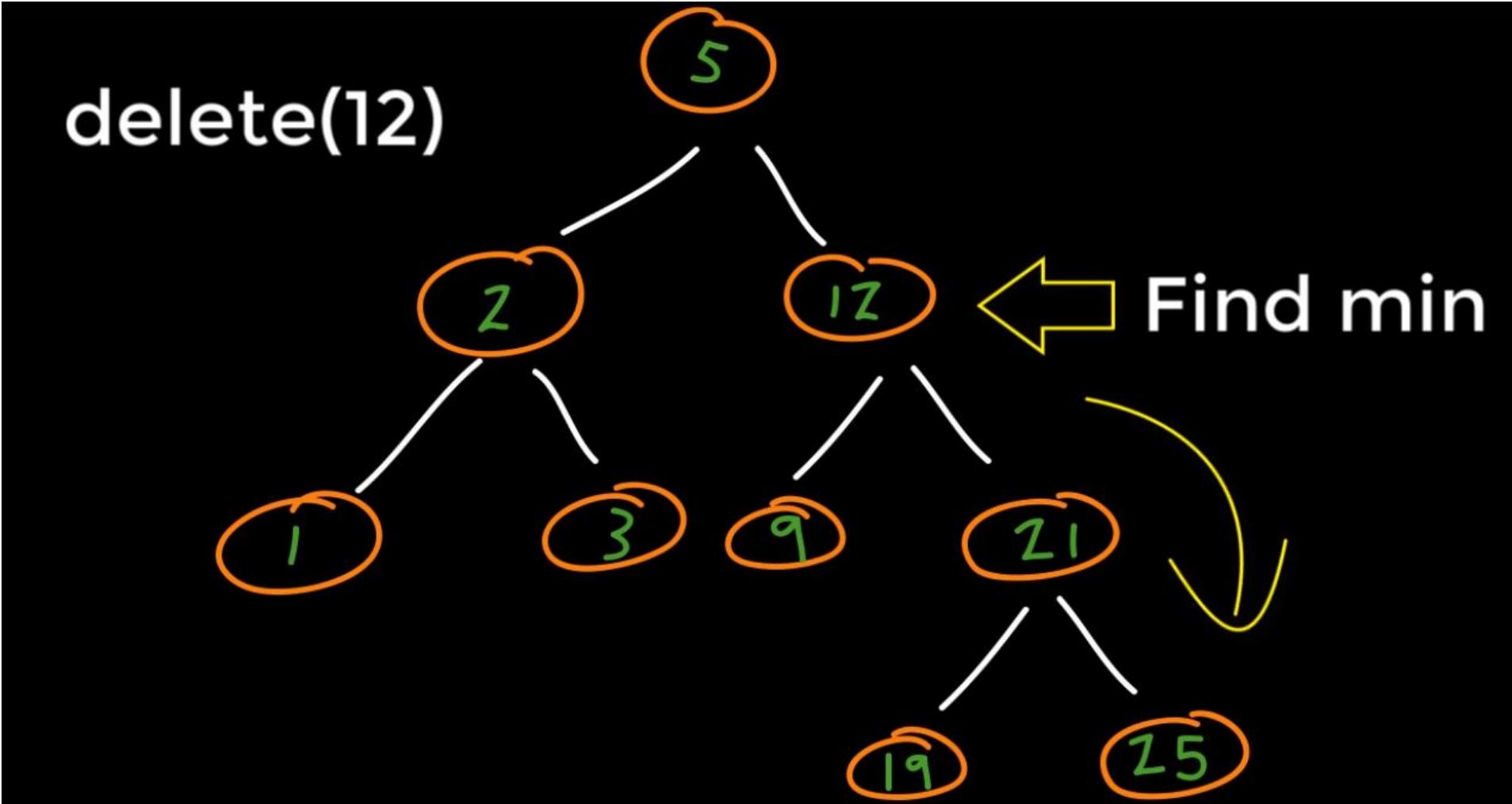
Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



BST – 2 dzieci – usuwamy 12

Znaleźć minimum po prawej stronie drzewa



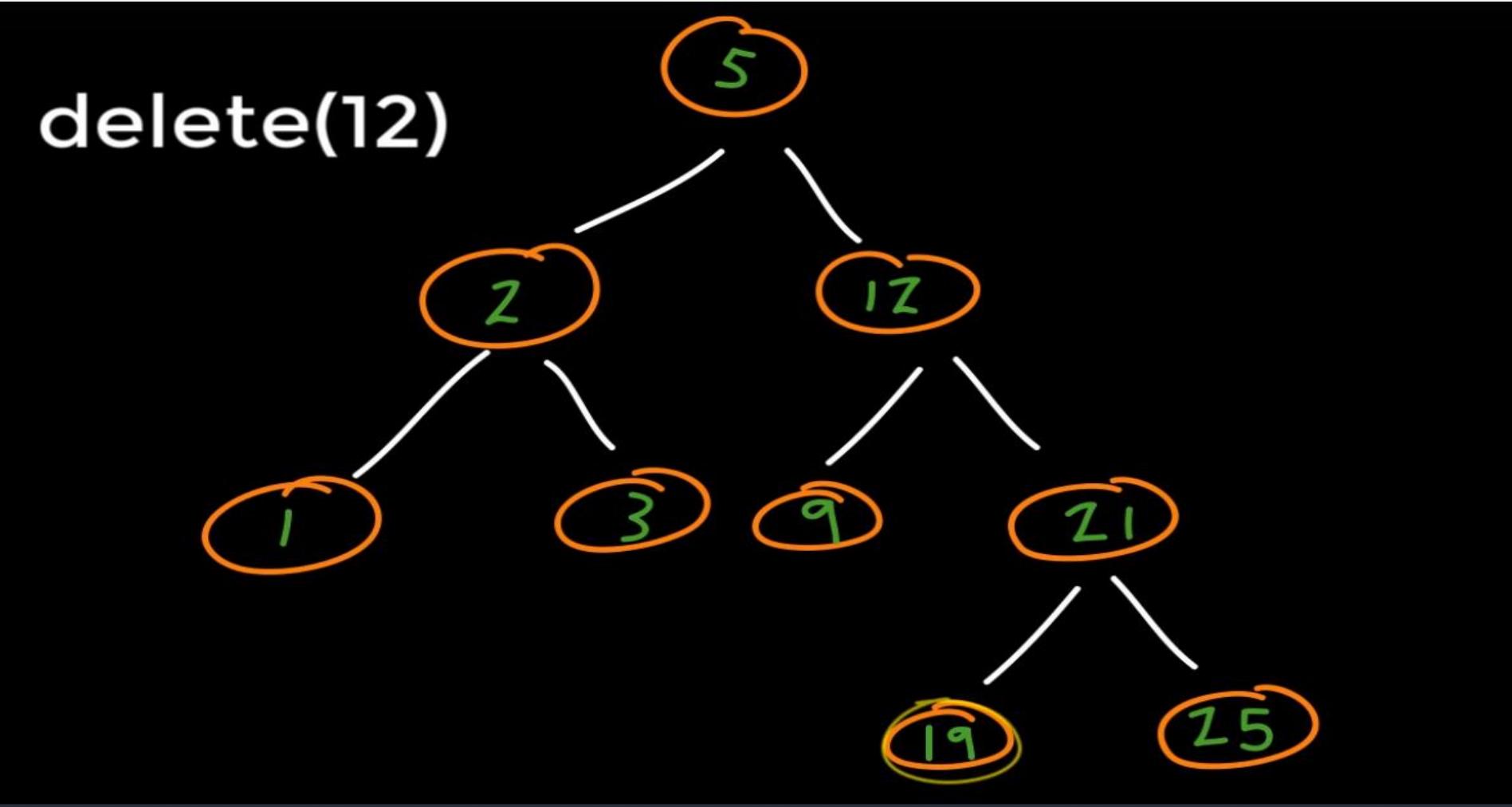
Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



BST – 2 dzieci – usuwamy 12

Minimum po prawej stronie to 19



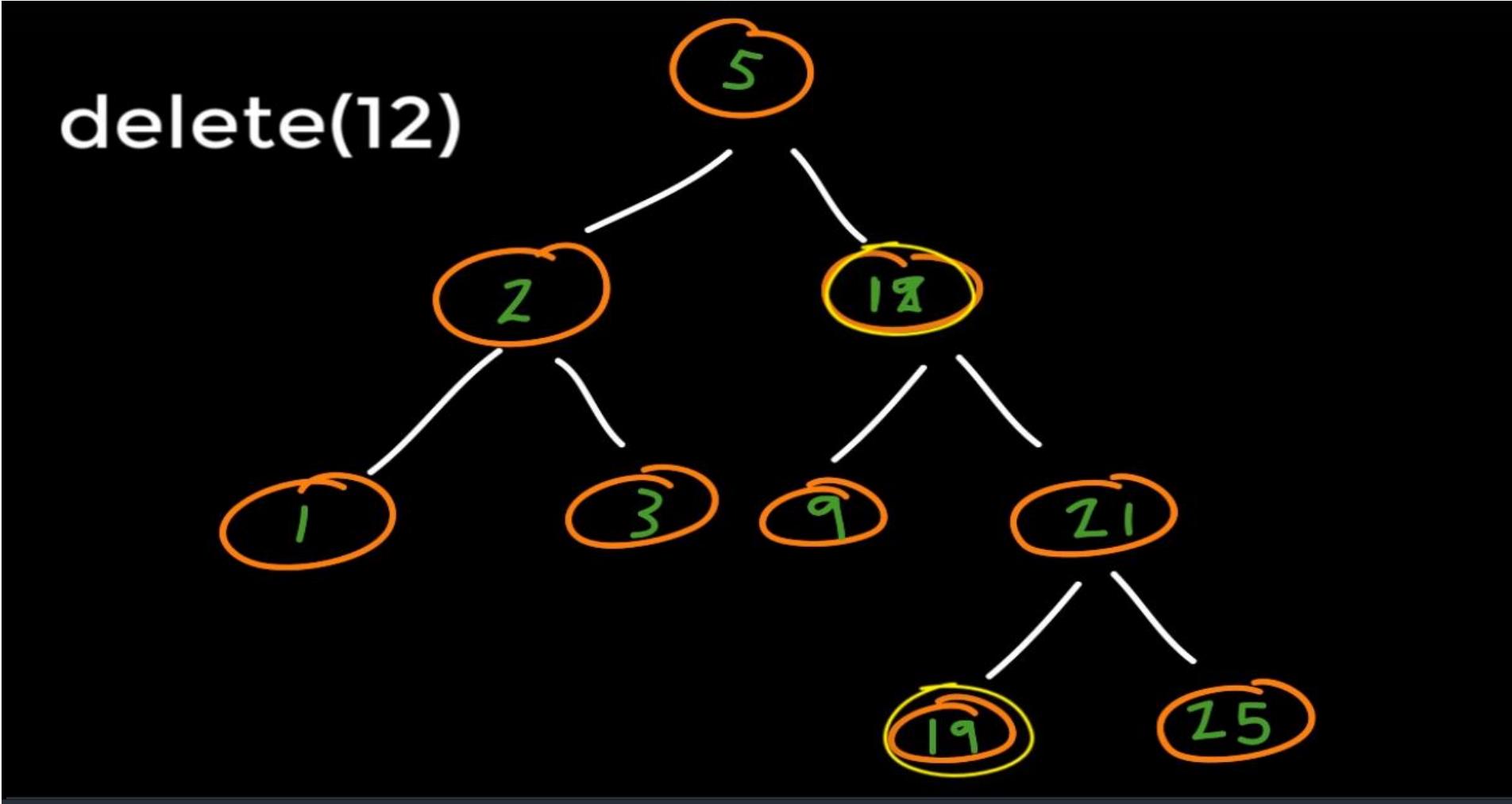
Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Drzewa binarne – usuwanie – 2 dzieci

Na usuwanym nodzie ustawiamy wartość 19



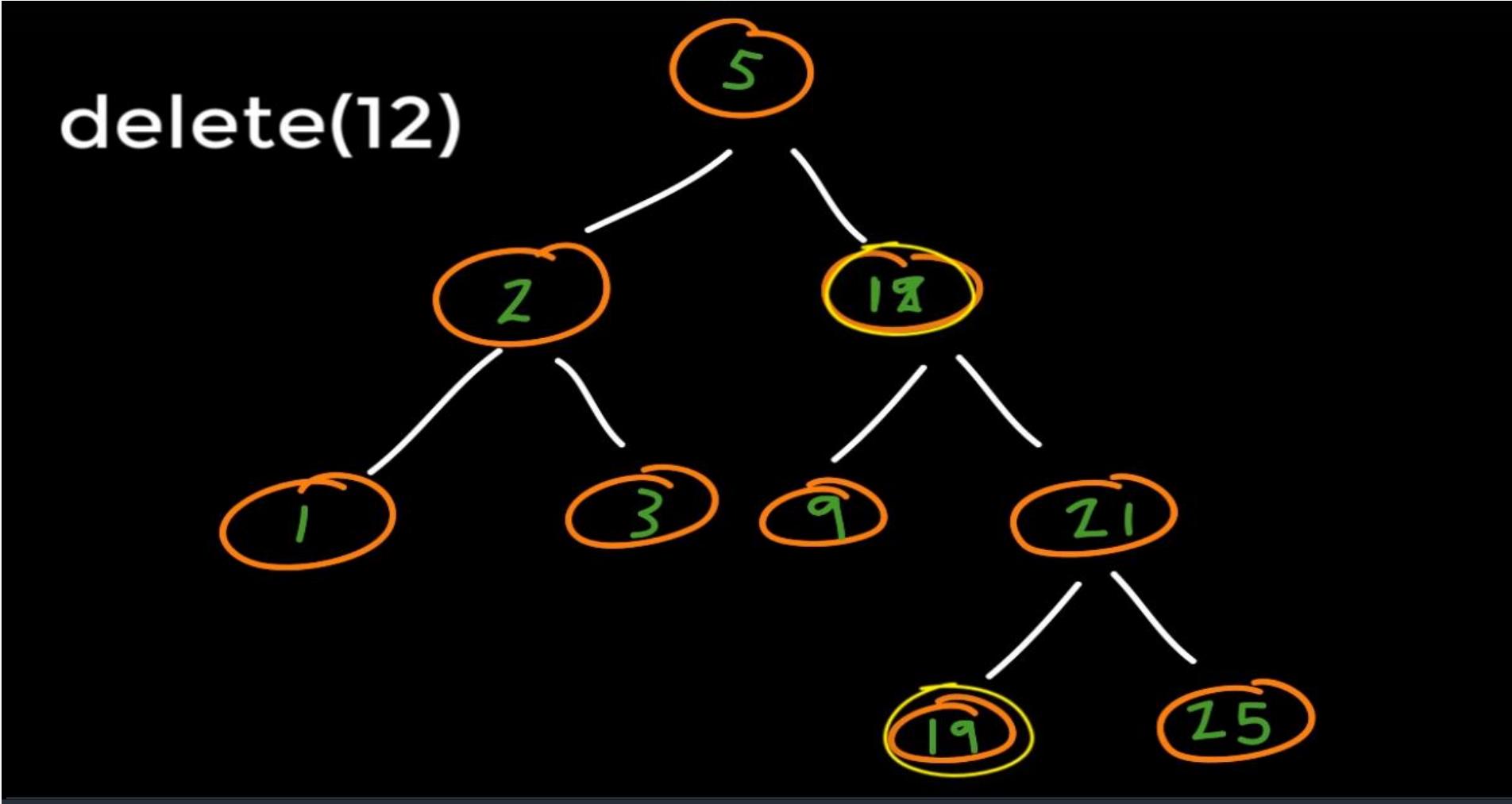
Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Drzewa binarne – usuwanie – 2 dzieci

Na usuwanym nodzie ustawiamy wartość 19



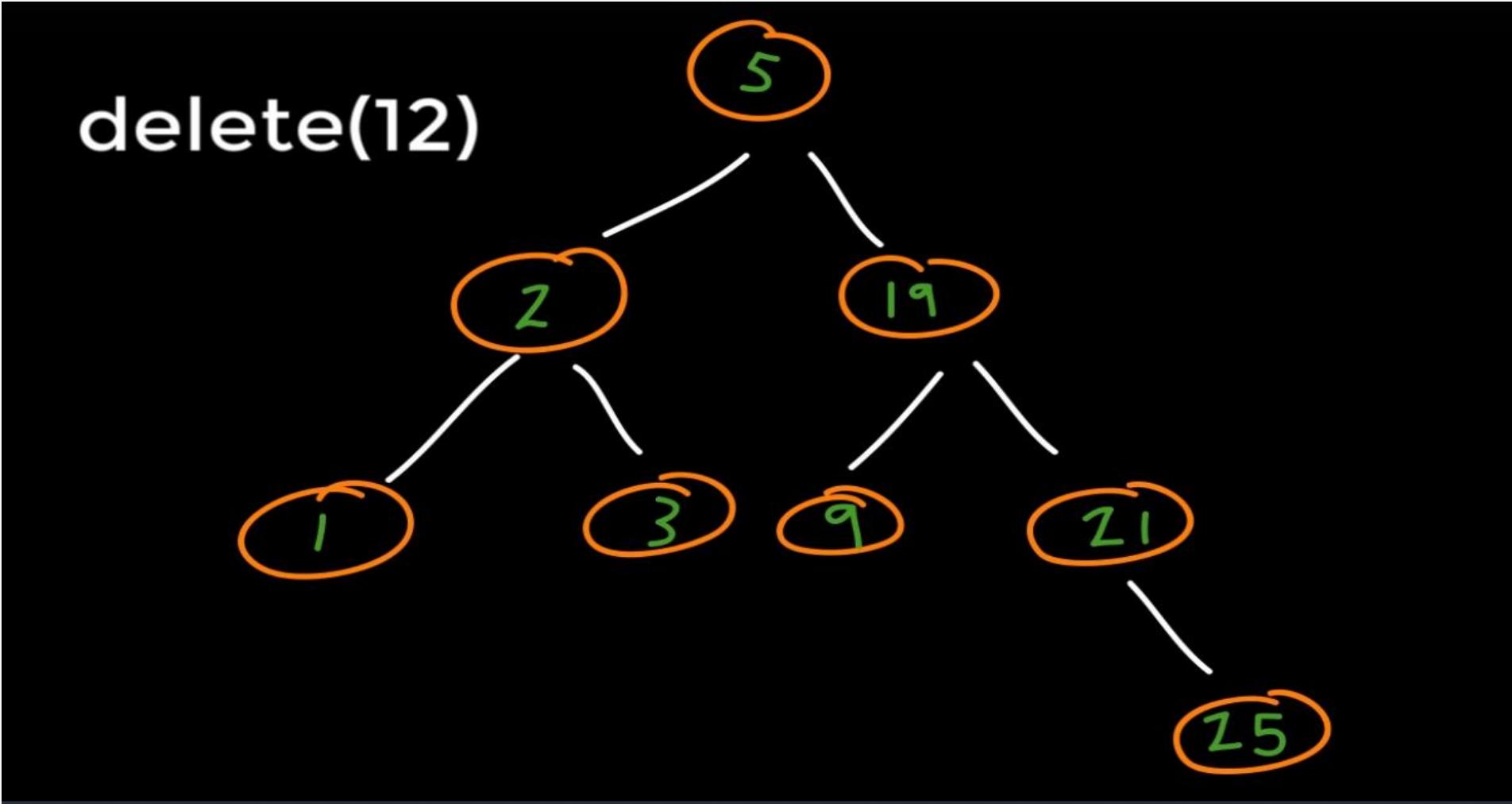
Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Drzewa binarne – usuwanie – 2 dzieci

Po usunięciu drzewo wygląda :



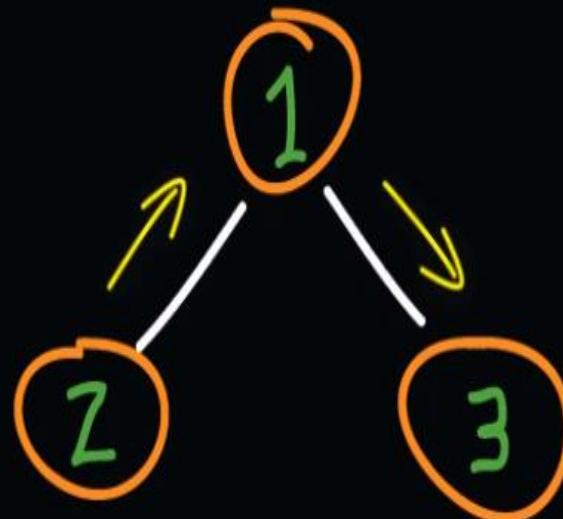
Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Metody iteracji po drzewie – przeszukiwanie w głąb

Inorder (L, Root, R) : 213



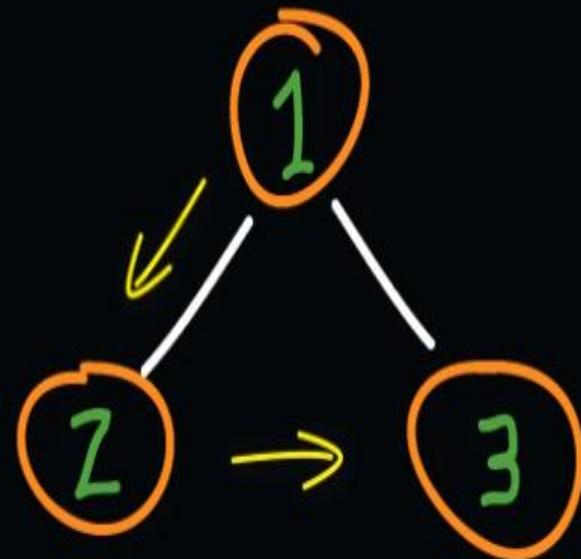
Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Metody iteracji po drzewie – przeszukiwanie w głąb

Preorder (Root, L, R) : 1 2 3

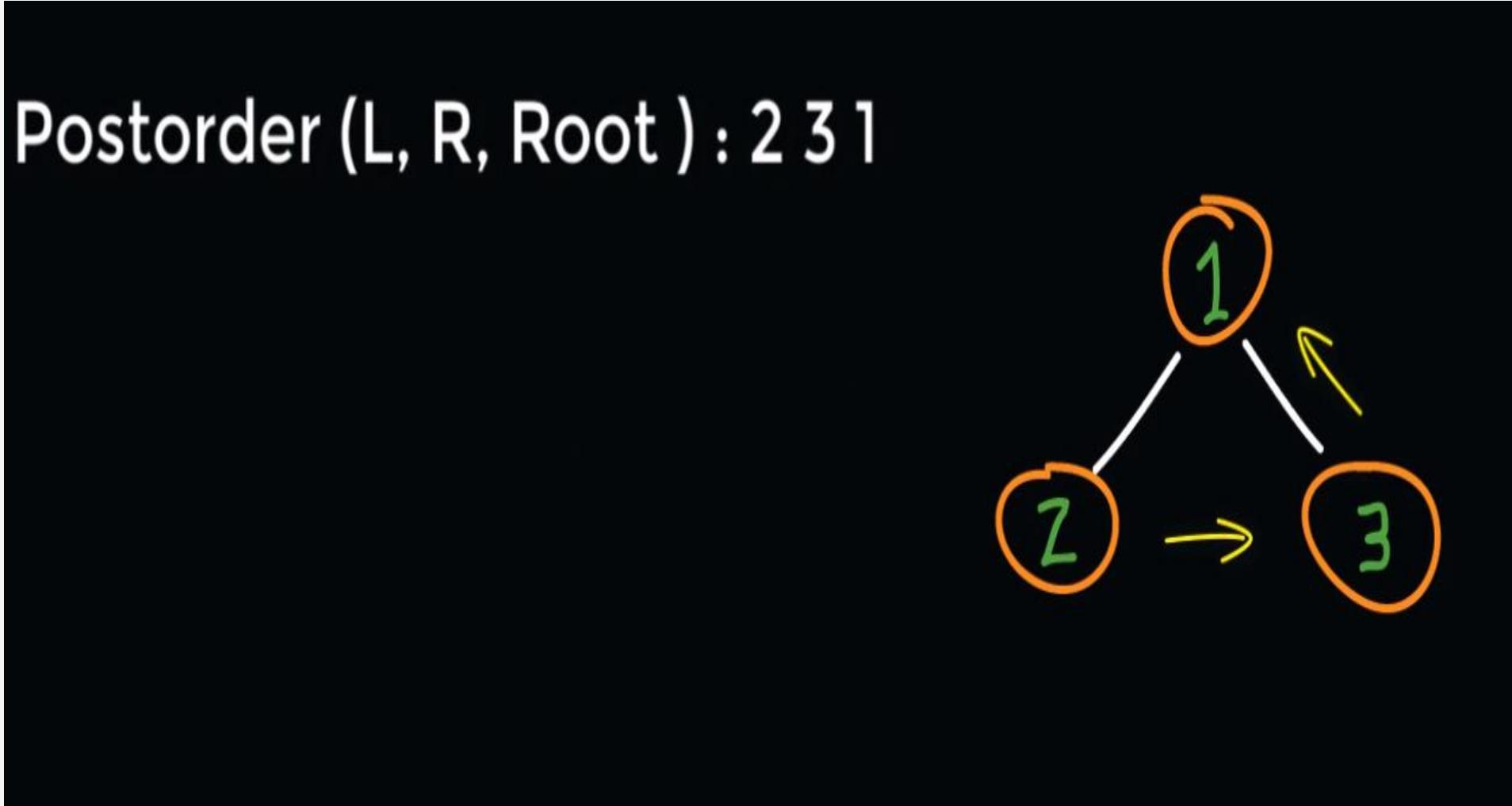


Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Metody iteracji po drzewie – przeszukiwanie w głąb



Autor:

Prawa do korzystania z materiałów posiada Software Development Academy

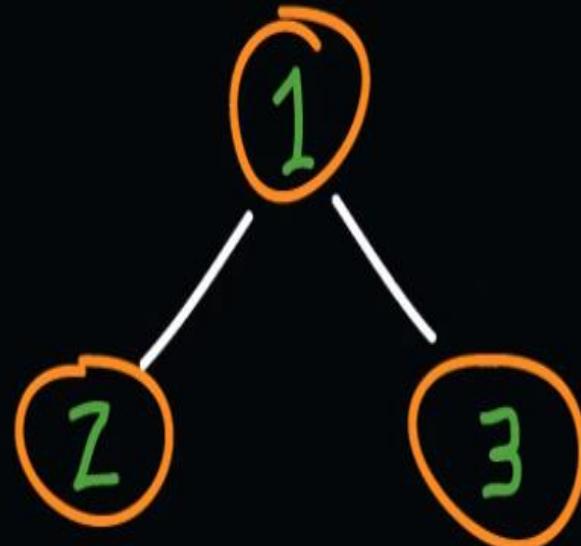


Metody iteracji po drzewie – przeszukiwanie w głąb

Inorder (L, Root, R) : 2 1 3

Preorder (Root, L, R) : 1 2 3

Postorder (L, R, Root) : 2 3 1



Autor:

Prawa do korzystania z materiałów posiada Software Development Academy



Binary Search Tree Runtime

$O(\log n)$

Find / Insert / Delete



Przydatne strony

https://eduinf.waw.pl/inf/alg/001_search/0086.php - o algorytmach i strukturach danych

<http://www.algorytm.org/> - duży zbiór gotowych implementacji prostych algorytmów

<http://www.hackerearth.com/practice> - świetna strona , zawiera opis poszczególnych algorytmów, w języku angielskim

<http://www.mathcs.emory.edu/~cheung/Courses/171/Syllabus/9-BinTree/BST-delete2.html> - bardzo dobrze opisane operacja na drzewach binarnych

https://www.youtube.com/watch?v=jNi_X5bvmQ0 - o rekurencji w nieco zabawny sposób

Autor:

Prawa do korzystania z materiałów posiada Software Development Academy