

Podstawy programowania W11OPA-SI0072G

Wstęp do programowania W11FTE-SI0141WL

Wstęp do programowania W11IKW-SI0080WL

rok akademicki 2024/25
semestr letni

Wykład 7

Karol Tarnowski

karol.tarnowski@pwr.edu.pl

L-1 p. 221

Plan prezentacji

- Co to jest algorytm?
- Algorytmy sortowania:
 - Sortowanie przez wybór (selectsort)
 - Sortowanie bąbelkowe (bubblesort)
 - Sortowanie przez scalanie (mergesort)
 - Sortowanie szybkie (quicksort)

Co to jest algorytm?

- Algorytm jest przepisem opisującym krok po kroku rozwiązanie problemu lub osiągnięcie jakiegoś celu
- Algorytmika to dziedzina zajmująca się algorytmami i ich właściwościami (dowodzenie poprawności, złożoność obliczeniowa i pamięciowa)

Zapis algorytmów

- Algorytm może być zapisywany na różne sposoby: językiem naturalnym, pseudokodem, schematem blokowym, językiem programowania

Znajdowanie elementu maksymalnego

- Dane wejściowe:
 - n -elementowa lista liczb
- Dane wyjściowe:
 - wartość największej liczby na liście

Znajdowanie elementu maksymalnego

01_element_maksymalny.py X

```
1  #Program ilustruje działanie funkcji wbudowanej max()
2  #do znalezienia największego elementu na liście.
3
4  #Lista jest pomieszana funkcją shuffle z modułu random.
5
6  import random #import modułu - wykorzystywana funkcja shuffle
7
8  def main():
9      #utworzenie listy zawierającej liczby całkowite
10     l = list(range(8))
11     #pomieszanie kolejności liczb
12     random.shuffle(l)
13     print('Zawartość pomieszanej listy: ',l)
14     print('Element maksymalny znaleziony funkcją max():', max(l))
15
16 main()
17
```

Znajdowanie elementu maksymalnego

- Przypisz *maksimum* wartość początkowego elementu tablicy
- Dla kolejnych elementów tablicy:
 - Jeśli dany element jest większy od *maksimum*
 - Przypisz *maksimum* wartość danego elementu

Znajdowanie elementu maksymalnego

```
02_element_maksymalny.py X
1  #Program przedstawia funkcję my_max(),
2  #która znajduje największy element na liście.
3
4  #Lista jest pomieszana funkcją shuffle z modułu random.
5
6  import random #import modułu - wykorzystywana funkcja shuffle
7
8  def my_max(l):
9      m = l[0]
10     for item in l[1:]:
11         if item > m:
12             m = item
13     return m
14
15 def main():
16     #utworzenie listy zawierającej liczby całkowite
17     l = list(range(8))
18     #pomieszenie kolejności liczb
19     random.shuffle(l)
20     print('Zawartość pomieszanej listy: ',l)
21     print('Element maksymalny znaleziony funkcją my_max():', my_max(l))
22
23 main()
```

Znajdowanie elementu maksymalnego

```
02_element_maksymalny.py X
1  #Program przedstawia funkcję my_max(),
2  #która znajduje największy element na liście.
3
4  #Lista jest pomieszana funkcją shuffle z modułu random.
5
6  import random #import modułu - wyk
7
8  def my_max(l):
9      m = l[0]
10     for item in l[1:]:
11         if item > m:
12             m = item
13     return m
14
15 def main():
16     #utworzenie listy zawierającej
17     l = list(range(8))
18     #pomieszenie kolejności liczb
19     random.shuffle(l)
20     print('Zawartość pomieszanej listy: ', l)
21     print('Element maksymalny znaleziony funkcją my_max():', my_max(l))
22
23 main()
```

Znajdowanie elementu maksymalnego

```
03_element_maksymalny.py X
1  #Program przedstawia funkcję my_max(),
2  #która znajduje największy element na liście.
3  #Wykorzystuje indeks do przejścia przez elementy listy.
4
5  #Lista jest pomieszana funkcją shuffle z modułu random.
6
7  import random #import modułu - wykorzystywana funkcja shuffle
8
9  def my_max(l):
10     m = l[0]
11     n = len(l)
12     for i in range(1,n):
13         if l[i] > m:
14             m = l[i]
15
16     return m
17
18 def main():
19     #utworzenie listy zawierającej liczby całkowite
20     l = list(range(8))
21     #pomieszenie kolejności liczb
22     random.shuffle(l)
23     print('Zawartość pomieszanej listy: ',l)
24     print('Element maksymalny znaleziony funkcją my_max():', my_max(l))
25
26 main()
```

Znajdowanie elementu maksymalnego

```
03_element_maksymalny.py X
1 #Program przedstawia funkcję my_max(),
2 #która znajduje największy element na liście.
3 #Wykorzystuje indeks do przejścia przez elementy listy.
4
5 #Lista jest pomieszana funkcją sh
6
7 import random #import modułu - wy
8
9 def my_max(l):
10     m = l[0]
11     n = len(l)
12     for i in range(1,n):
13         if l[i] > m:
14             m = l[i]
15
16     return m
17
18 def main():
19     #utworzenie listy zawierającej
20     l = list(range(8))
21     #pomieszenie kolejności liczb
22     random.shuffle(l)
23     print('Zawartość pomieszczonej listy: ',l)
24     print('Element maksymalny znaleziony funkcją my_max():', my_max(l))
25
26 main()
27
```

```
03_element_maksymalny.py X
9 def my_max(l):
10     m = l[0]
11     n = len(l)
12     for i in range(1,n):
13         if l[i] > m:
14             m = l[i]
15
16     return m
17
```

Zamiana dwóch elementów na liście

```
04_sort.py X
1  #Moduł 04_sort zawiera funkcję swap(), która pozwala zamienić
2  #miejscami dwa elementy na liście.
3
4  #Moduł zawiera również funkcję main(), która demonstruje działanie
5  #funkcji swap().
6
7  def swap(l,left,right):
8      # item      = l[left]
9      # l[left]   = l[right]
10     # l[right]  = item
11     l[left], l[right] = l[right], l[left]
12
13  def main():
14     l = list(range(4))
15     print(l)
16     swap(l,1,2)
17     print(l)
18
19  #Instrukcja warunkowa, która sprawdza, czy plik był
20  #uruchomiony jako skrypt, czy załadowany jako moduł.
21  if __name__ == "__main__":
22     main()
23
```

Sortowanie bąbelkowe

- Jeżeli ciąg nie jest uporządkowany, to istnieją dwa sąsiednie elementy, które są w złej kolejności

Sortowanie bąbelkowe

3	5	1	2	8	4	7	6
---	---	---	---	---	---	---	---

3	1	5	2	8	4	7	6
---	---	---	---	---	---	---	---

3	1	2	5	8	4	7	6
---	---	---	---	---	---	---	---

3	1	2	5	4	8	7	6
---	---	---	---	---	---	---	---

3	1	2	5	4	7	8	6
---	---	---	---	---	---	---	---

3	1	2	5	4	7	6	8
---	---	---	---	---	---	---	---

Sortowanie bąbelkowe

3	1	2	5	4	7	6	8
---	---	---	---	---	---	---	---

1	3	2	5	4	7	6	8
---	---	---	---	---	---	---	---

1	2	3	5	4	7	6	8
---	---	---	---	---	---	---	---

1	2	3	4	5	7	6	8
---	---	---	---	---	---	---	---

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

Sortowanie bąbelkowe

- Wykonaj $n-1$ przebiegów przez tablicę
 - Wykonaj $n-i-1$ porównań sąsiednich elementów (gdzie i to numer iteracji liczony od 0)
 - Jeśli elementy są nie po kolei to zamień je miejscami

Sortowanie bąbelkowe

```
sort.py X
1  #Moduł sort.py zawiera funkcje implementujące
2  #różne algorytmy sortowania i funkcje pomocnicze.
3
4  import random #wykorzystywana funkcja shuffle()
5
6  def bubblesort(l):
7      n = len(l)
8      for i in range(n-1):
9          for j in range(n-i-1):
10             if l[j+1] < l[j]:
11                 swap(l,j,j+1)
12
```

Sortowanie przez wybór

- Wyszukujemy pozycję najmniejszego elementu i przestawiamy go na właściwą pozycję
- Następnie kontynuujemy dla pozostałej części tablicy

Sortowanie przez wybór

3	5	1	2	8	4	7	6
---	---	---	---	---	---	---	---

1	5	3	2	8	4	7	6
---	---	---	---	---	---	---	---

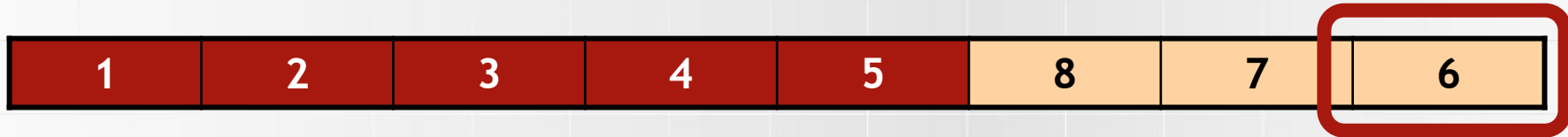
1	2	3	5	8	4	7	6
---	---	---	---	---	---	---	---

1	2	3	5	8	4	7	6
---	---	---	---	---	---	---	---

1	2	3	4	8	5	7	6
---	---	---	---	---	---	---	---

1	2	3	4	5	8	7	6
---	---	---	---	---	---	---	---

Sortowanie przez wybór



Sortowanie przez scalanie

- Podziel tablicę na dwie równe części
- Zastosuj sortowanie przez scalanie do każdej z nich oddzielnie
- Połącz posortowane podciągi w jeden ciąg posortowany

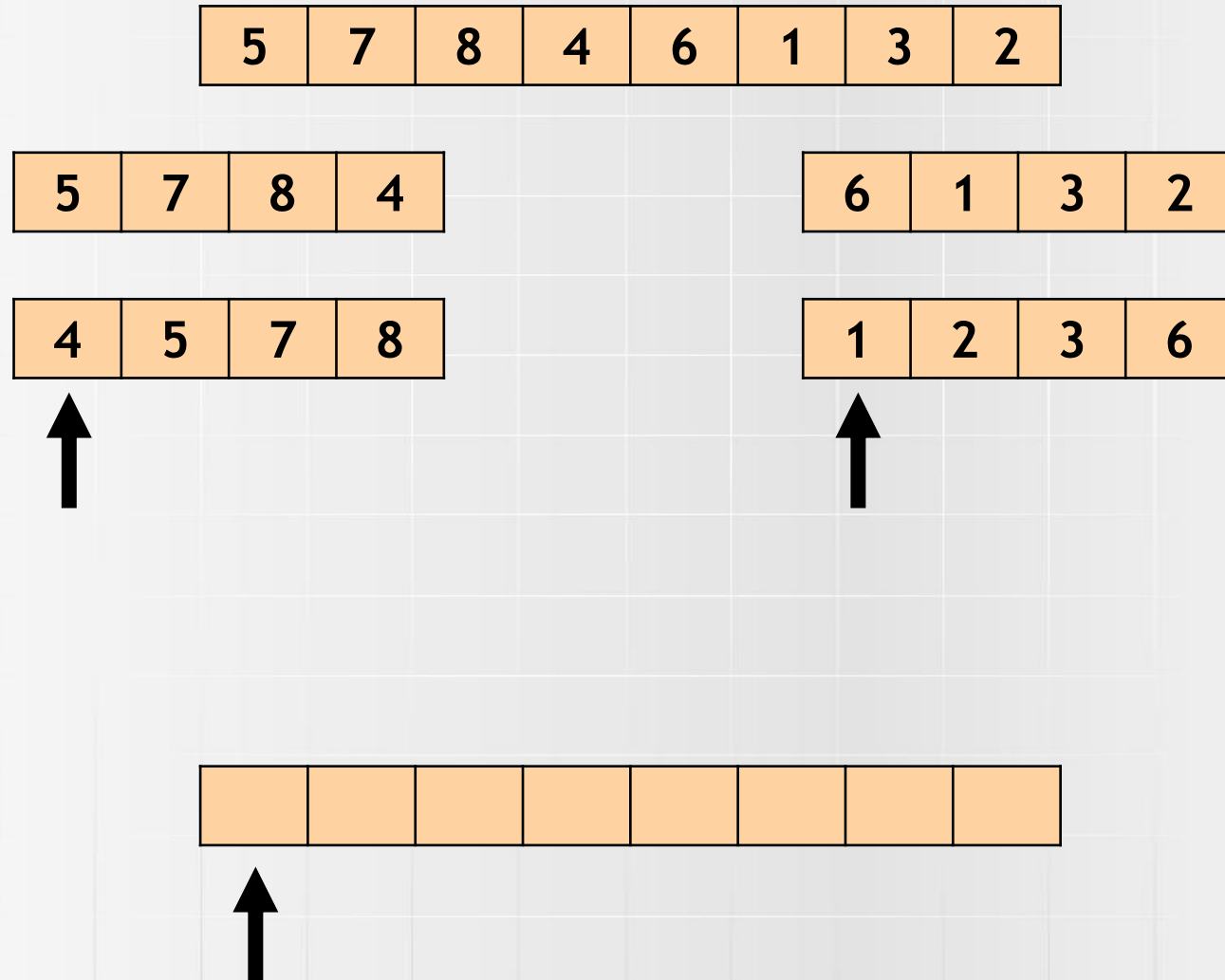
Sortowanie przez scalanie

Scalanie

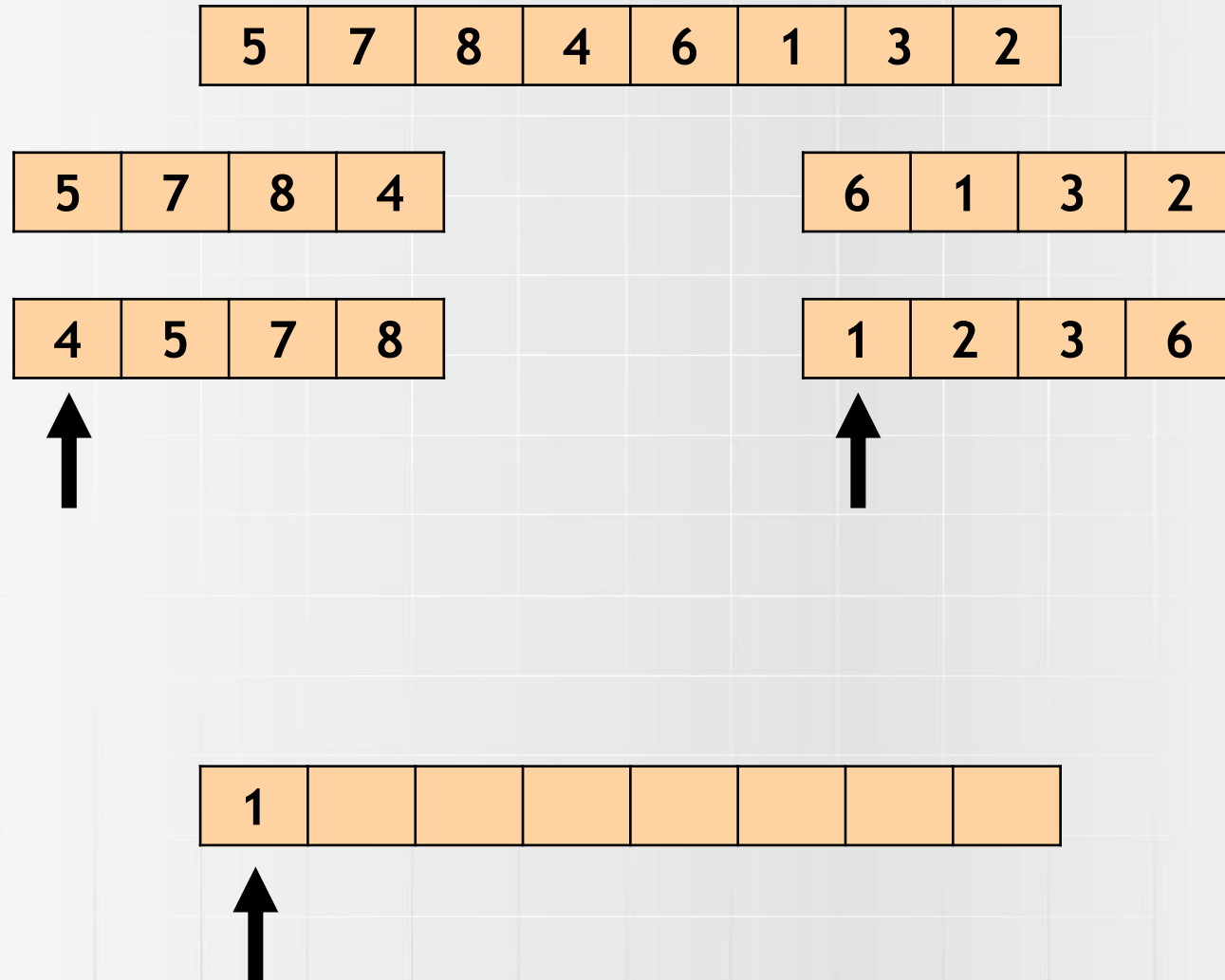
Procedura scalania dwóch ciągów $a[0..n-1]$ i $b[0..m-1]$ do ciągu $c[0..n+m-1]$

1. Ustaw indeksy na początek ciągów: $i=0$, $j=0$
2. Jeśli w ciągu a nie pozostało już nic do przetworzenia ($i \geq n$), to dołącz pozostałe elementy z b do c i zakończ
3. Jeśli w ciągu b nie pozostało już nic do przetworzenia ($j \geq m$), to dołącz pozostałe elementy z a do c i zakończ
4. Jeśli $a[i] \leq b[j]$ to dołącz $a[i]$ do c i zwiększ i o 1, w p.p. dołącz $b[j]$ do c i zwiększ j o 1.
5. Powtarzaj od 2.

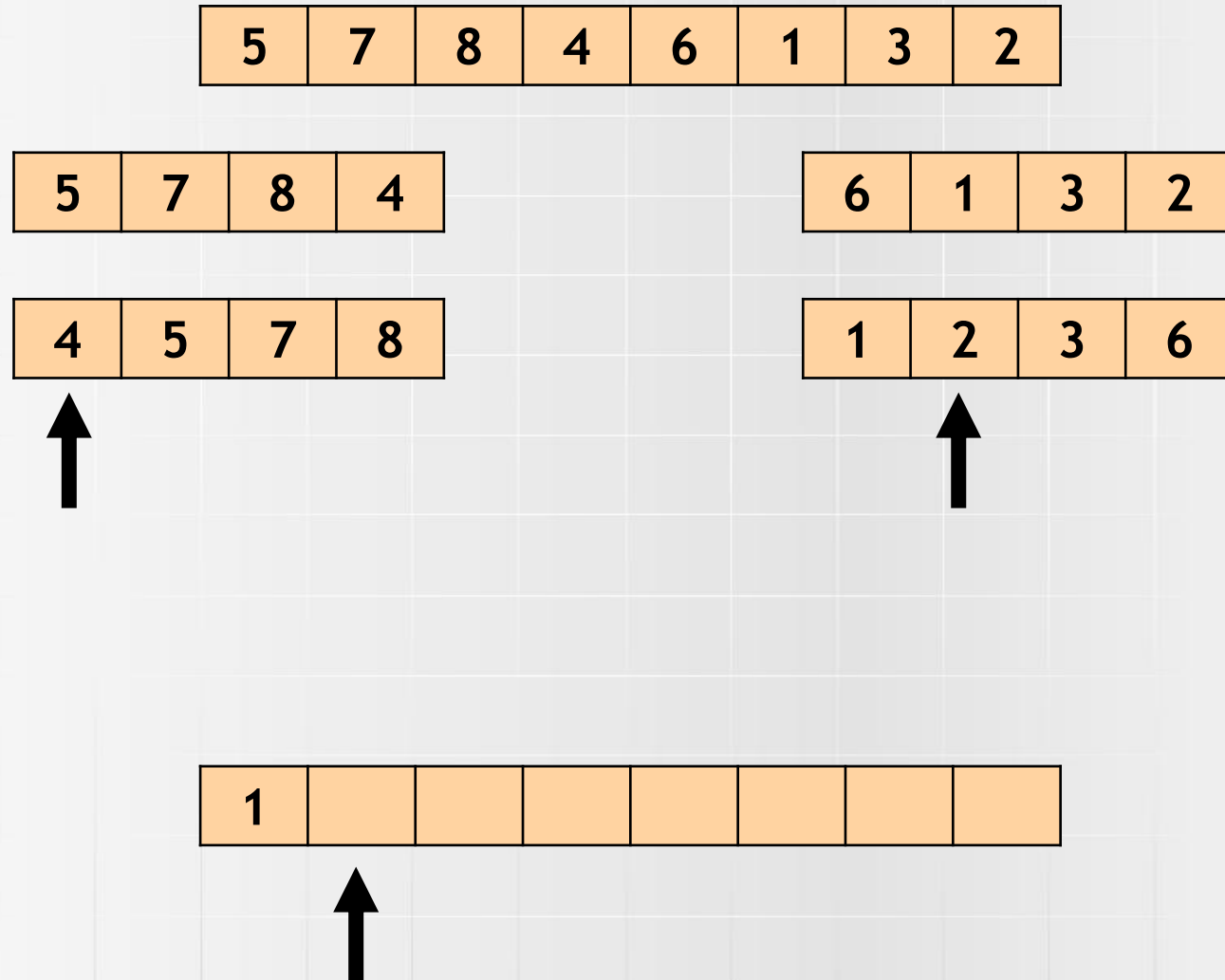
Sortowanie przez scalanie



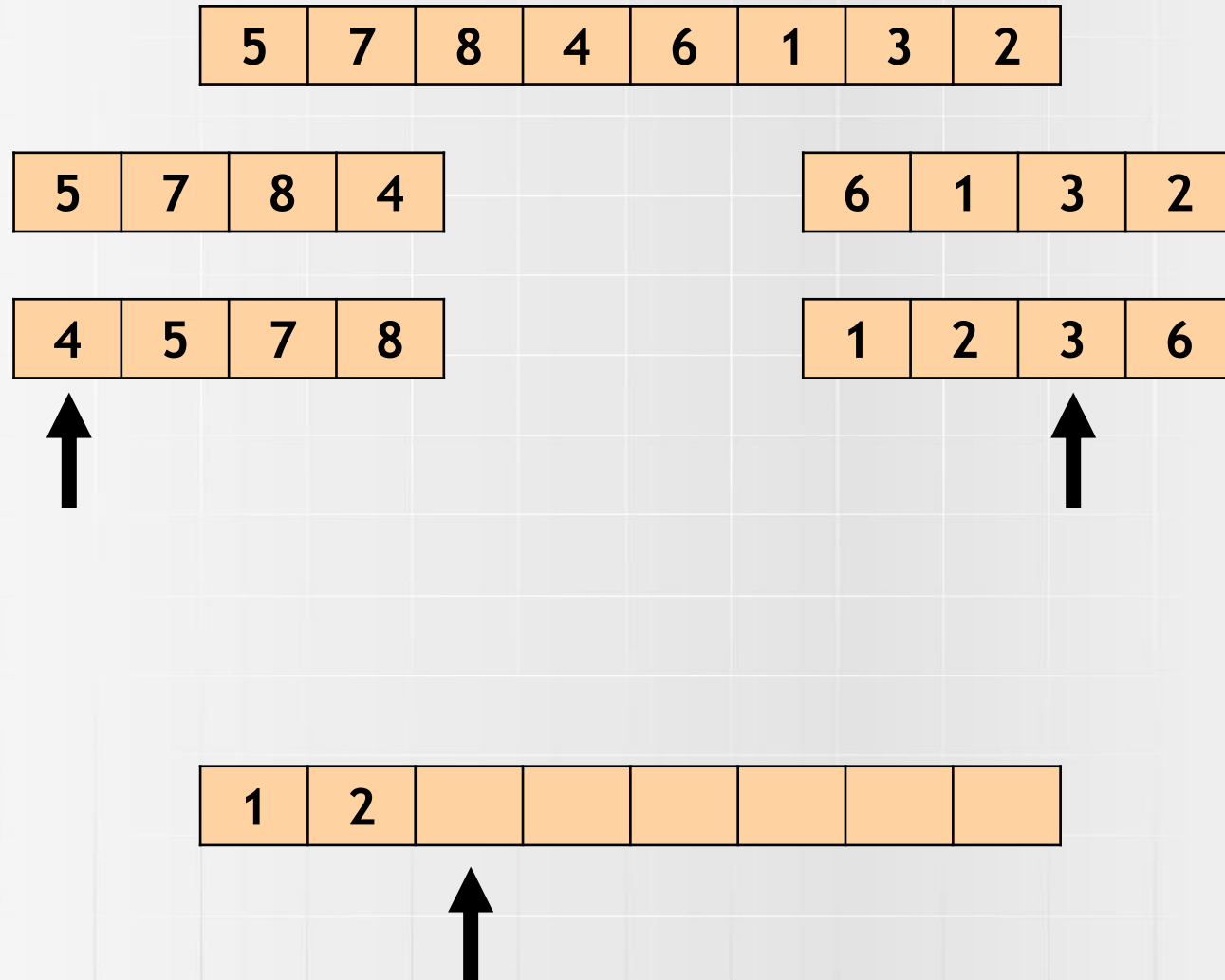
Sortowanie przez scalanie



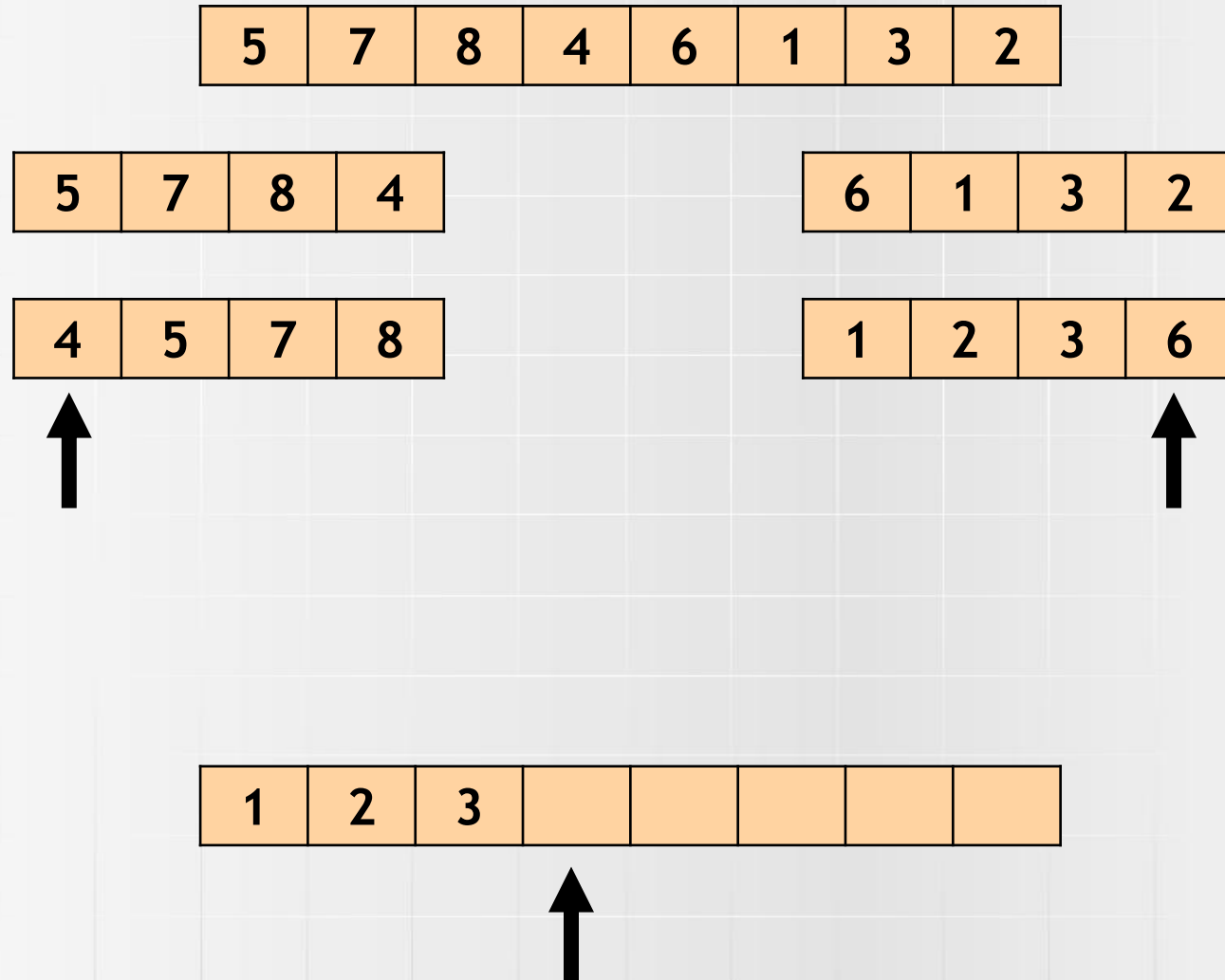
Sortowanie przez scalanie



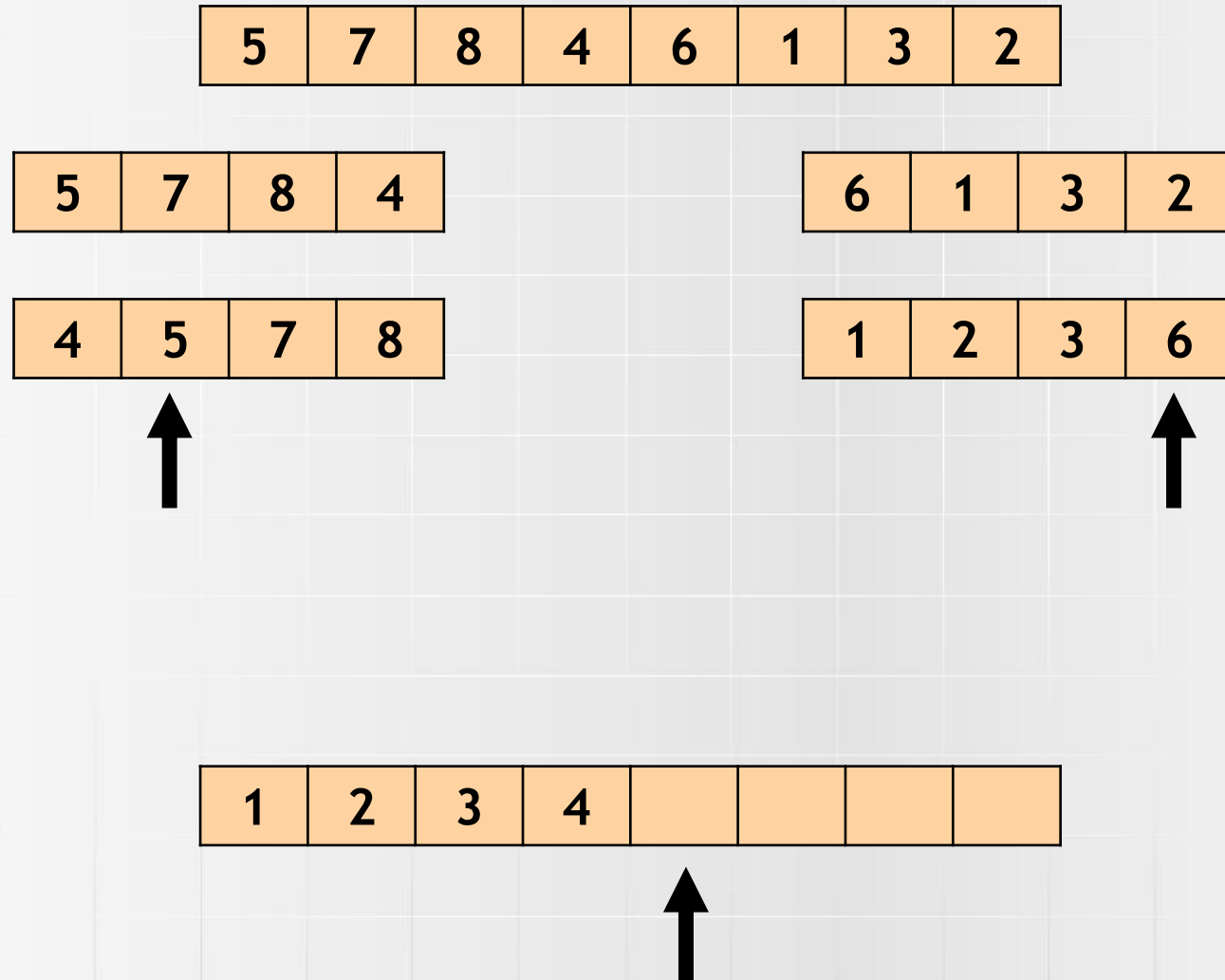
Sortowanie przez scalanie



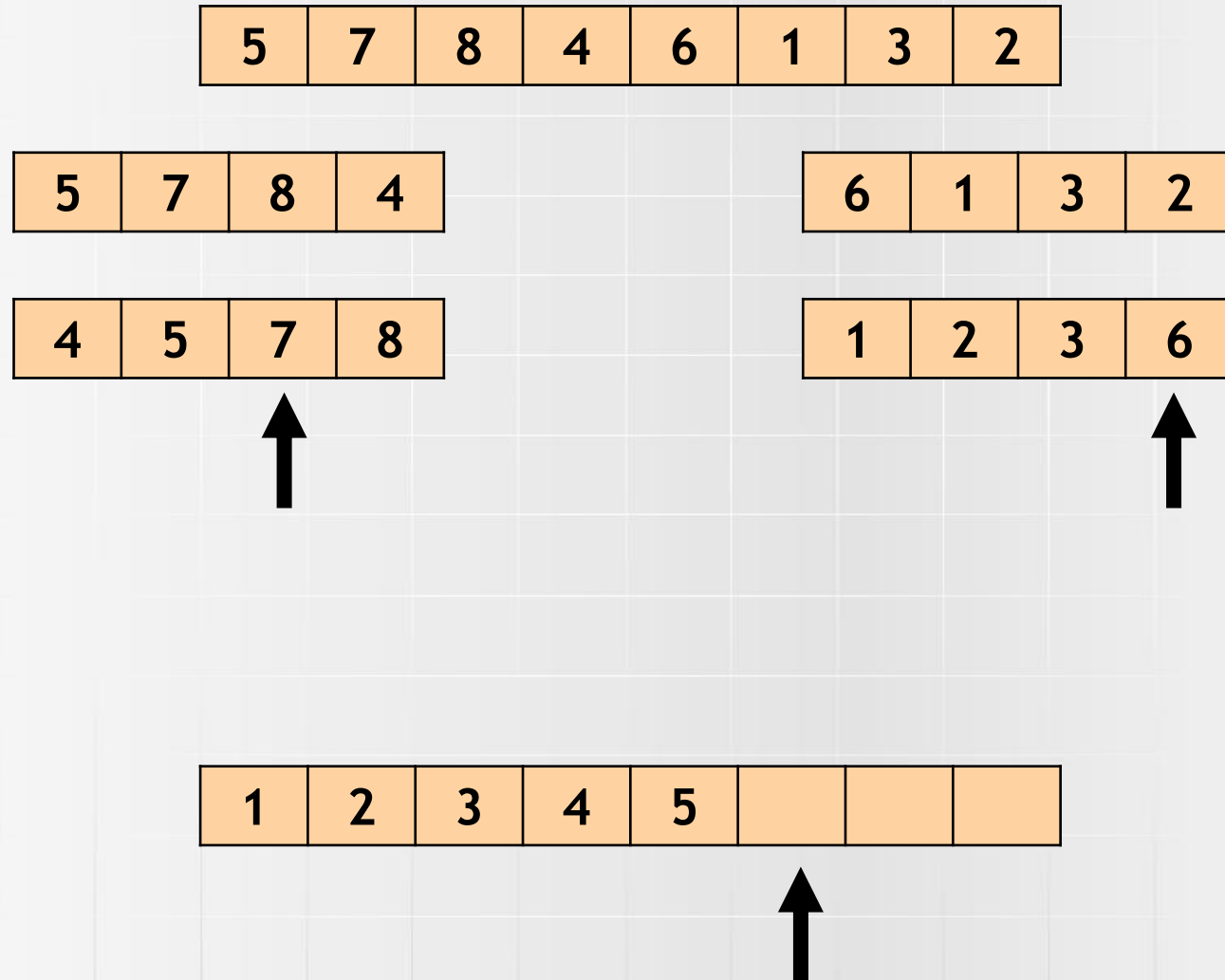
Sortowanie przez scalanie



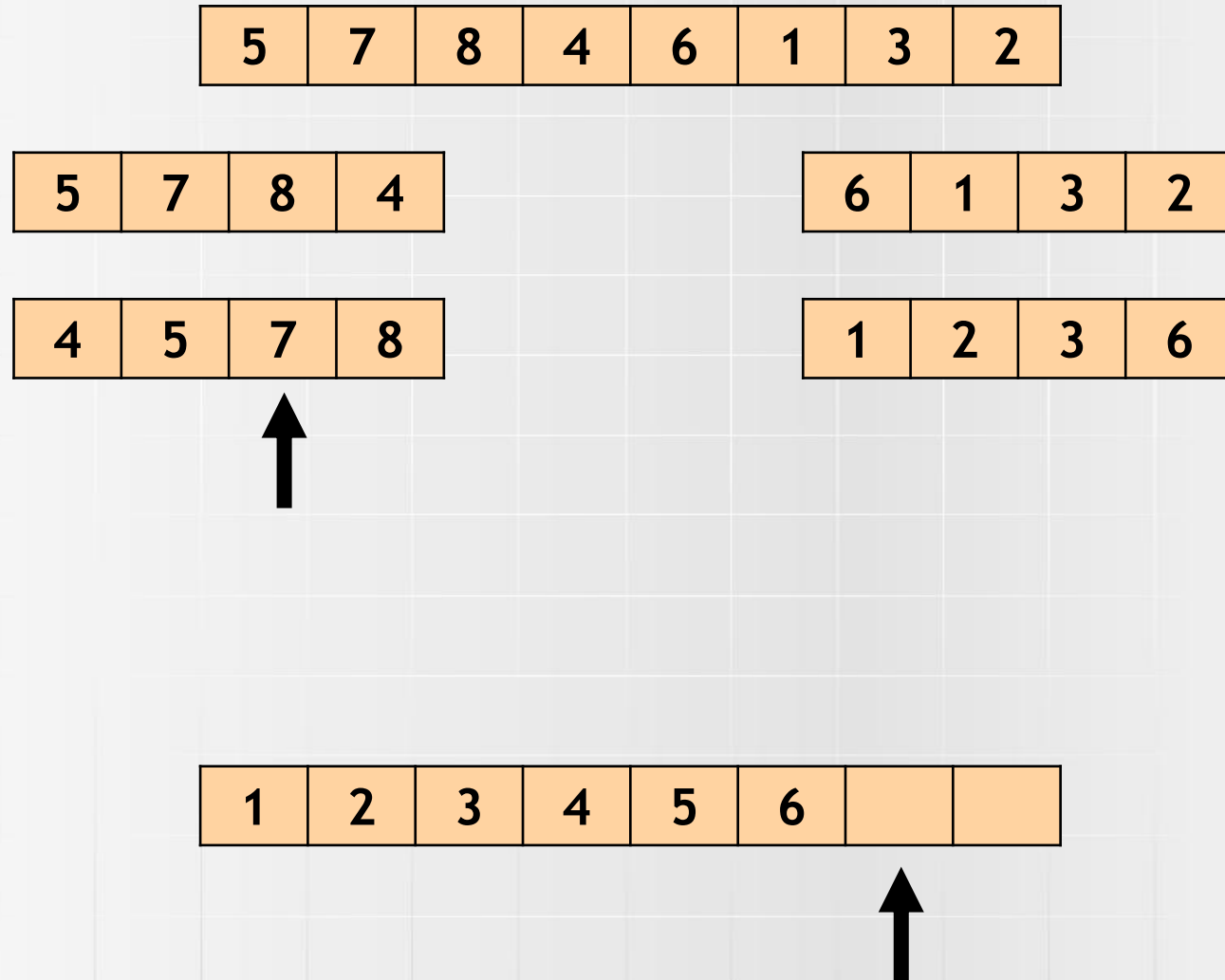
Sortowanie przez scalanie



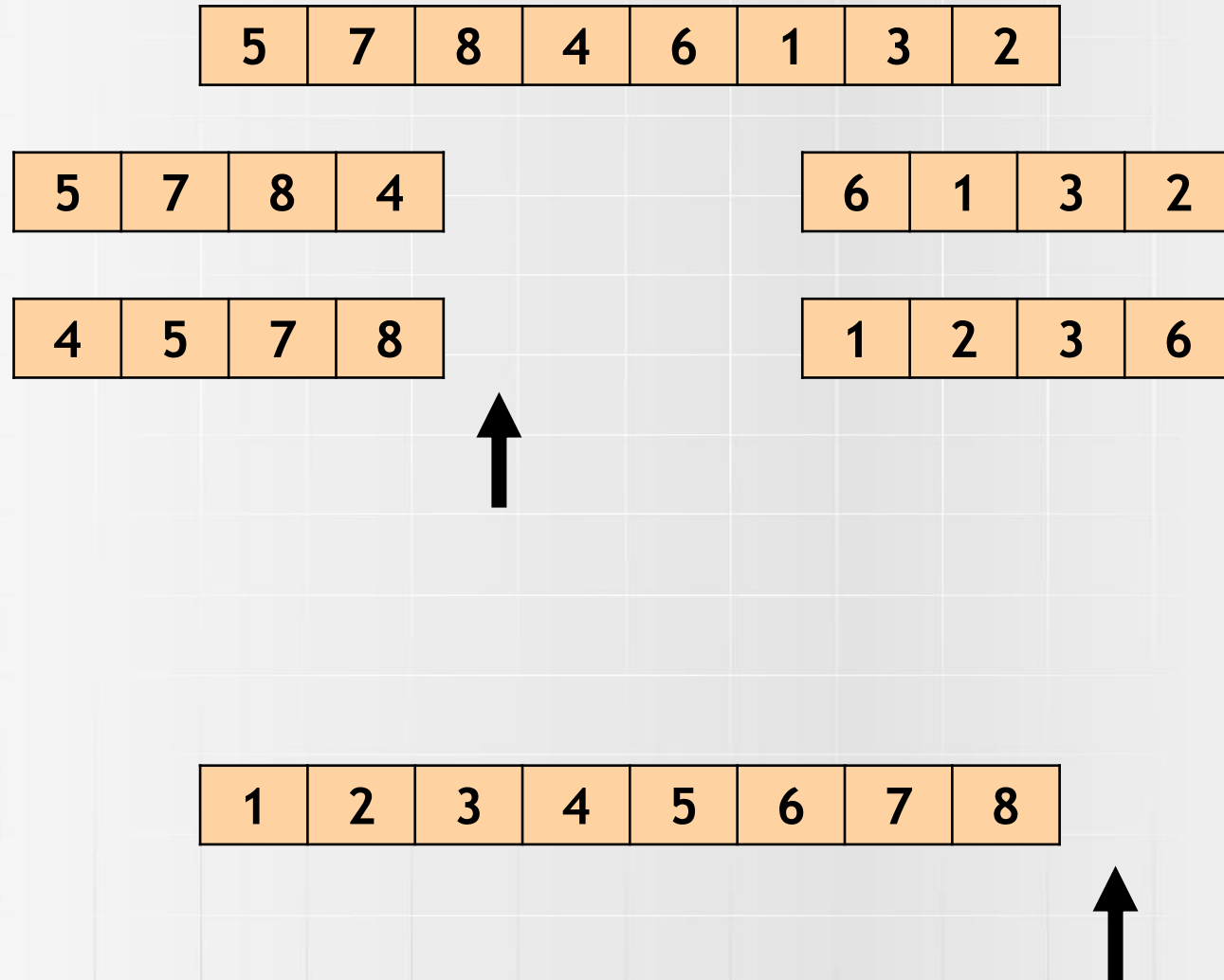
Sortowanie przez scalanie



Sortowanie przez scalanie



Sortowanie przez scalanie



Sortowanie przez scalanie

Scalanie

Procedura scalania dwóch ciągów $a[0..n-1]$ i $b[0..m-1]$ do ciągu $c[0..n+m-1]$

1. Ustaw indeksy na początek ciągów: $i=0$, $j=0$
2. Jeśli w ciągu a nie pozostało już nic do przetworzenia ($i \geq n$), to dołącz pozostałe elementy z b do c i zakończ
3. Jeśli w ciągu b nie pozostało już nic do przetworzenia ($j \geq m$), to dołącz pozostałe elementy z a do c i zakończ
4. Jeśli $a[i] \leq b[j]$ to dołącz $a[i]$ do c i zwiększ i o 1, w p.p. dołącz $b[j]$ do c i zwiększ j o 1.
5. Powtarzaj od 2.

Sortowanie szybkie

- Podziel tablicę na dwie części zawierające elementy mniejsze i większe od wybranego elementu
- Zastosuj sortowanie szybkie do każdej z nich oddzielnie
- Połącz posortowane podciągi w jeden ciąg posortowany

Sortowanie szybkie

5	7	8	4	6	1	3	2
---	---	---	---	---	---	---	---

2	7	8	4	6	1	3	5
---	---	---	---	---	---	---	---



2	7	8	4	6	1	3	5
---	---	---	---	---	---	---	---



Sortowanie szybkie

5	7	8	4	6	1	3	2
---	---	---	---	---	---	---	---

2	7	8	4	6	1	3	5
---	---	---	---	---	---	---	---



2	7	8	4	6	1	3	5
---	---	---	---	---	---	---	---



Sortowanie szybkie

5	7	8	4	6	1	3	2
---	---	---	---	---	---	---	---

2	7	8	4	6	1	3	5
---	---	---	---	---	---	---	---



2	7	8	4	6	1	3	5
---	---	---	---	---	---	---	---



Sortowanie szybkie

5	7	8	4	6	1	3	2
---	---	---	---	---	---	---	---

2	7	8	4	6	1	3	5
---	---	---	---	---	---	---	---



2	7	8	4	6	1	3	5
---	---	---	---	---	---	---	---



Sortowanie szybkie

5	7	8	4	6	1	3	2
---	---	---	---	---	---	---	---

2	7	8	4	6	1	3	5
---	---	---	---	---	---	---	---



2	4	8	7	6	1	3	5
---	---	---	---	---	---	---	---



Sortowanie szybkie

5	7	8	4	6	1	3	2
---	---	---	---	---	---	---	---

2	7	8	4	6	1	3	5
---	---	---	---	---	---	---	---



2	4	8	7	6	1	3	5
---	---	---	---	---	---	---	---



Sortowanie szybkie

5	7	8	4	6	1	3	2
---	---	---	---	---	---	---	---

2	7	8	4	6	1	3	5
---	---	---	---	---	---	---	---



2	4	8	7	6	1	3	5
---	---	---	---	---	---	---	---



Sortowanie szybkie

5	7	8	4	6	1	3	2
---	---	---	---	---	---	---	---

2	7	8	4	6	1	3	5
---	---	---	---	---	---	---	---



2	4	1	7	6	8	3	5
---	---	---	---	---	---	---	---



Sortowanie szybkie

5	7	8	4	6	1	3	2
---	---	---	---	---	---	---	---

2	7	8	4	6	1	3	5
---	---	---	---	---	---	---	---



2	4	1	7	6	8	3	5
---	---	---	---	---	---	---	---



Sortowanie szybkie

5	7	8	4	6	1	3	2
---	---	---	---	---	---	---	---

2	7	8	4	6	1	3	5
---	---	---	---	---	---	---	---



2	4	1	3	6	8	7	5
---	---	---	---	---	---	---	---



Sortowanie szybkie

5	7	8	4	6	1	3	2
---	---	---	---	---	---	---	---

2	7	8	4	6	1	3	5
---	---	---	---	---	---	---	---



2	4	1	3	6	8	7	5
---	---	---	---	---	---	---	---



Sortowanie szybkie

5	7	8	4	6	1	3	2
---	---	---	---	---	---	---	---

2	7	8	4	6	1	3	5
---	---	---	---	---	---	---	---



2	4	1	3	5	8	7	6
---	---	---	---	---	---	---	---

Sortowanie szybkie

5	7	8	4	6	1	3	2
---	---	---	---	---	---	---	---

2	7	8	4	6	1	3	5
---	---	---	---	---	---	---	---



2	4	1	3	5	8	7	6
---	---	---	---	---	---	---	---

1	2	3	4	5	8	7	6
---	---	---	---	---	---	---	---

1	2	3	4	5	8	7	6
---	---	---	---	---	---	---	---

1	2	3	4	5	8	7	6
---	---	---	---	---	---	---	---

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

Podsumowanie

- Co to jest algorytm?
- Co to jest złożoność algorytmu?
- Algorytmy sortowania:
 - Sortowanie przez wybór (selectsort)
 - Sortowanie bąbelkowe (bubblesort)
 - Sortowanie przez scalanie (mergesort)
 - Sortowanie szybkie (quicksort)