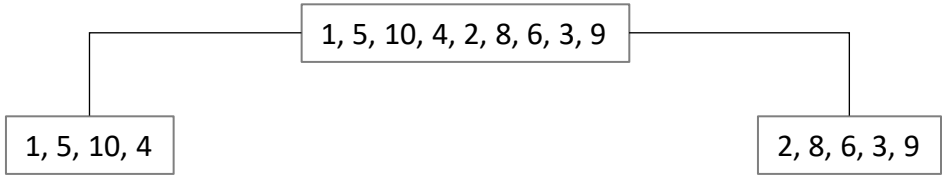
Merge sort: sortowanie przez scalanie

Znowu „dziel i zwyciężaj”…

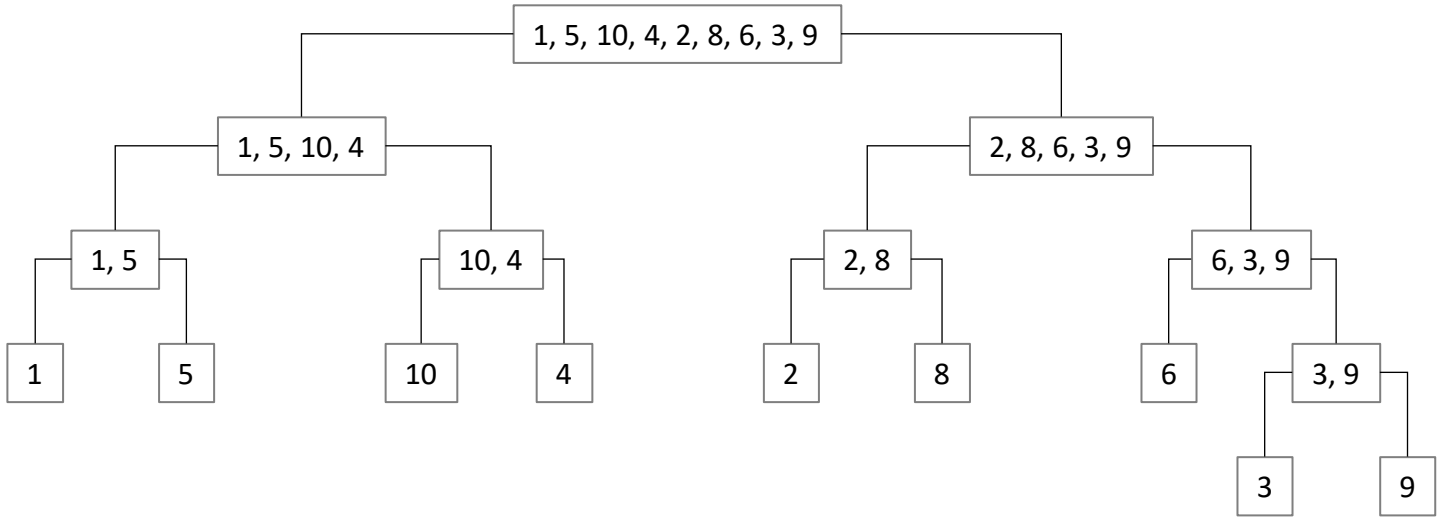
Merge sort: sortowanie przez scalanie

Rekurencyjne dzielenie zbioru na podzbiory



Merge sort: sortowanie przez scalanie

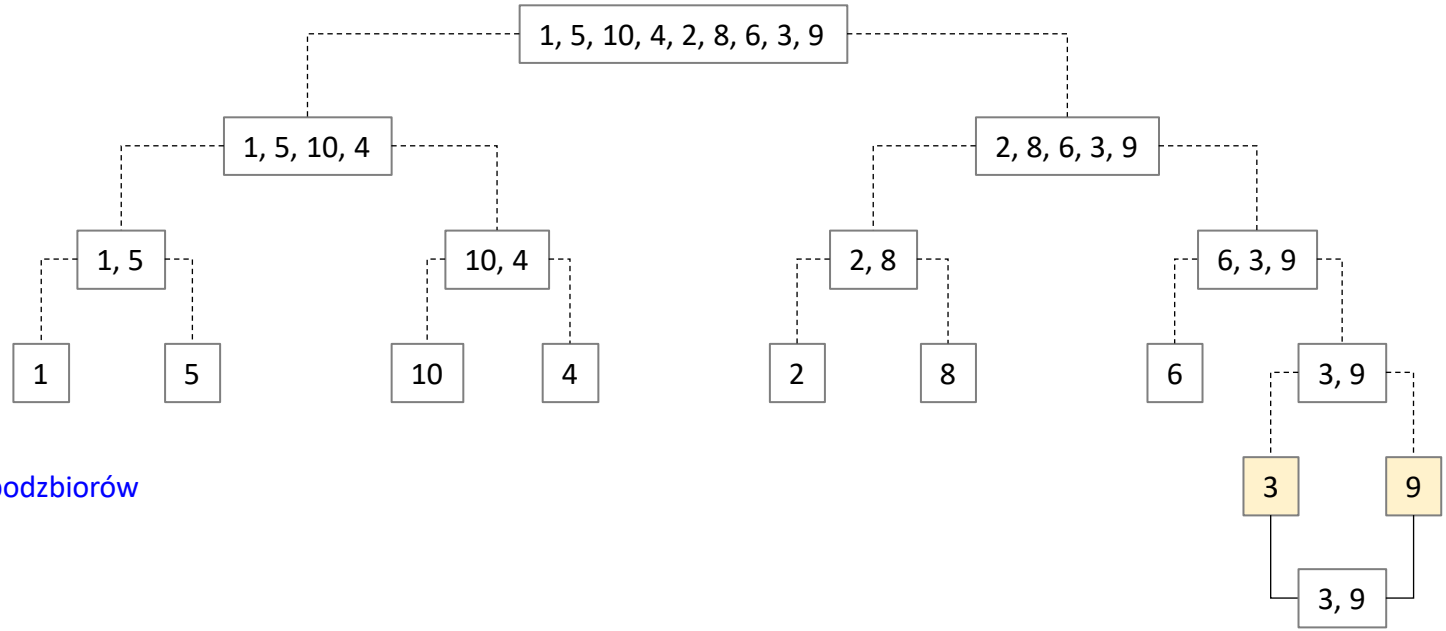
Rekurencyjne dzielenie zbioru na podzbiory



Dzielimy zbiory dopóki można dzielić.

Merge sort: sortowanie przez scalanie

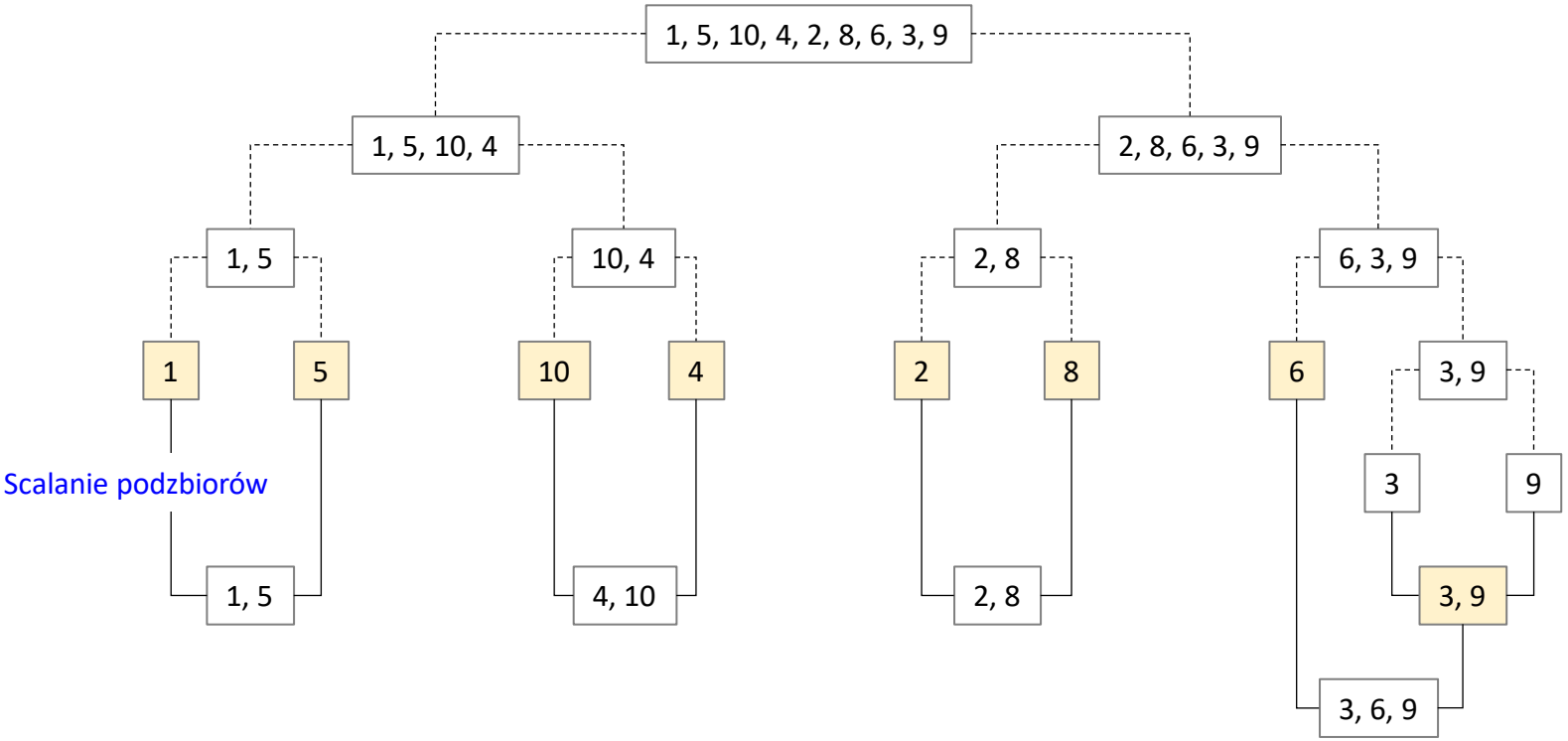
Rekurencyjne dzielenie zbioru na podzbiory



Scalanie podzbiorów

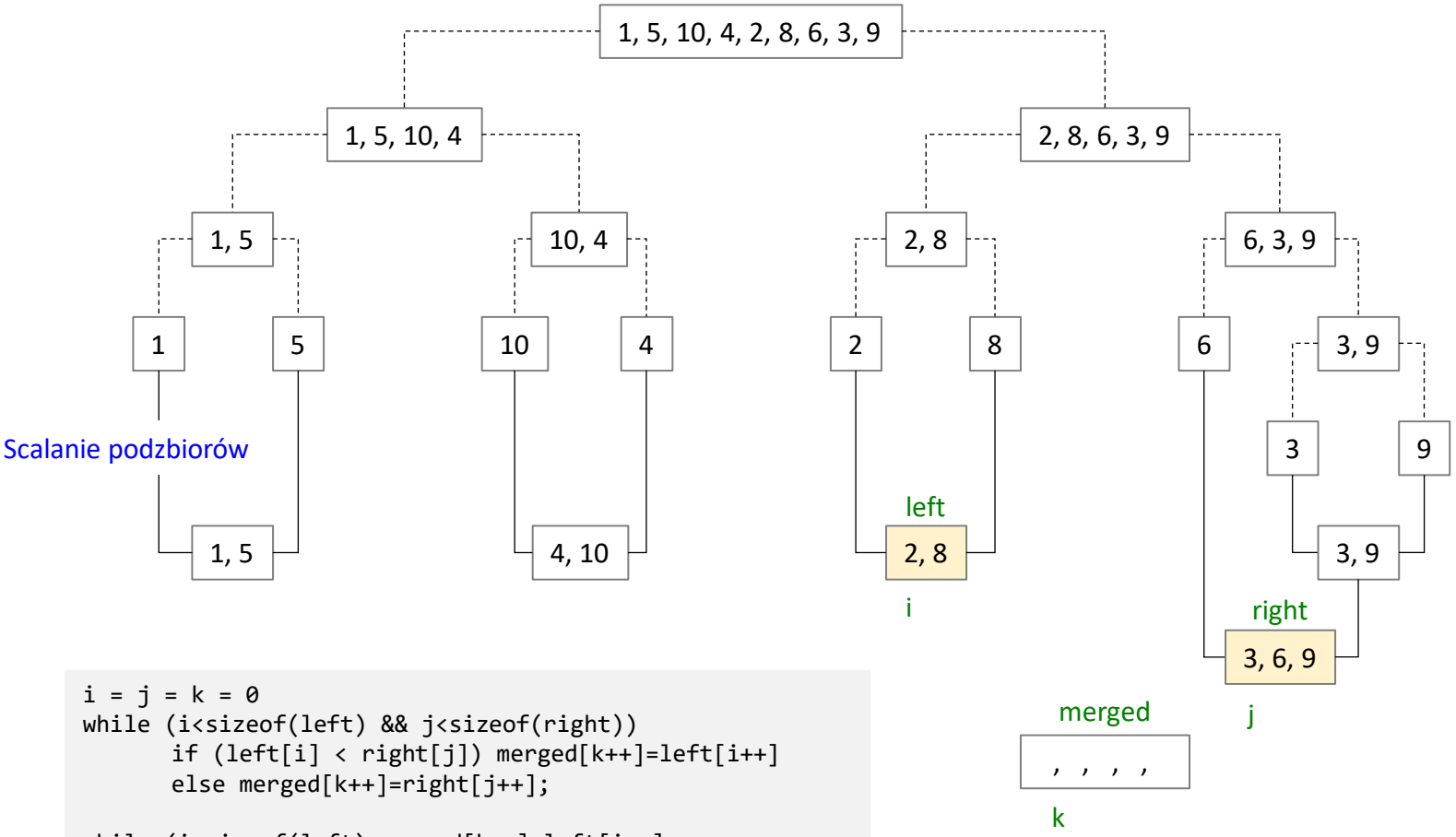
Merge sort: sortowanie przez scalanie

Rekurencyjne dzielenie zbioru na podzbiory



Merge sort: sortowanie przez scalanie

Rekurencyjne dzielenie zbioru na podzbiory

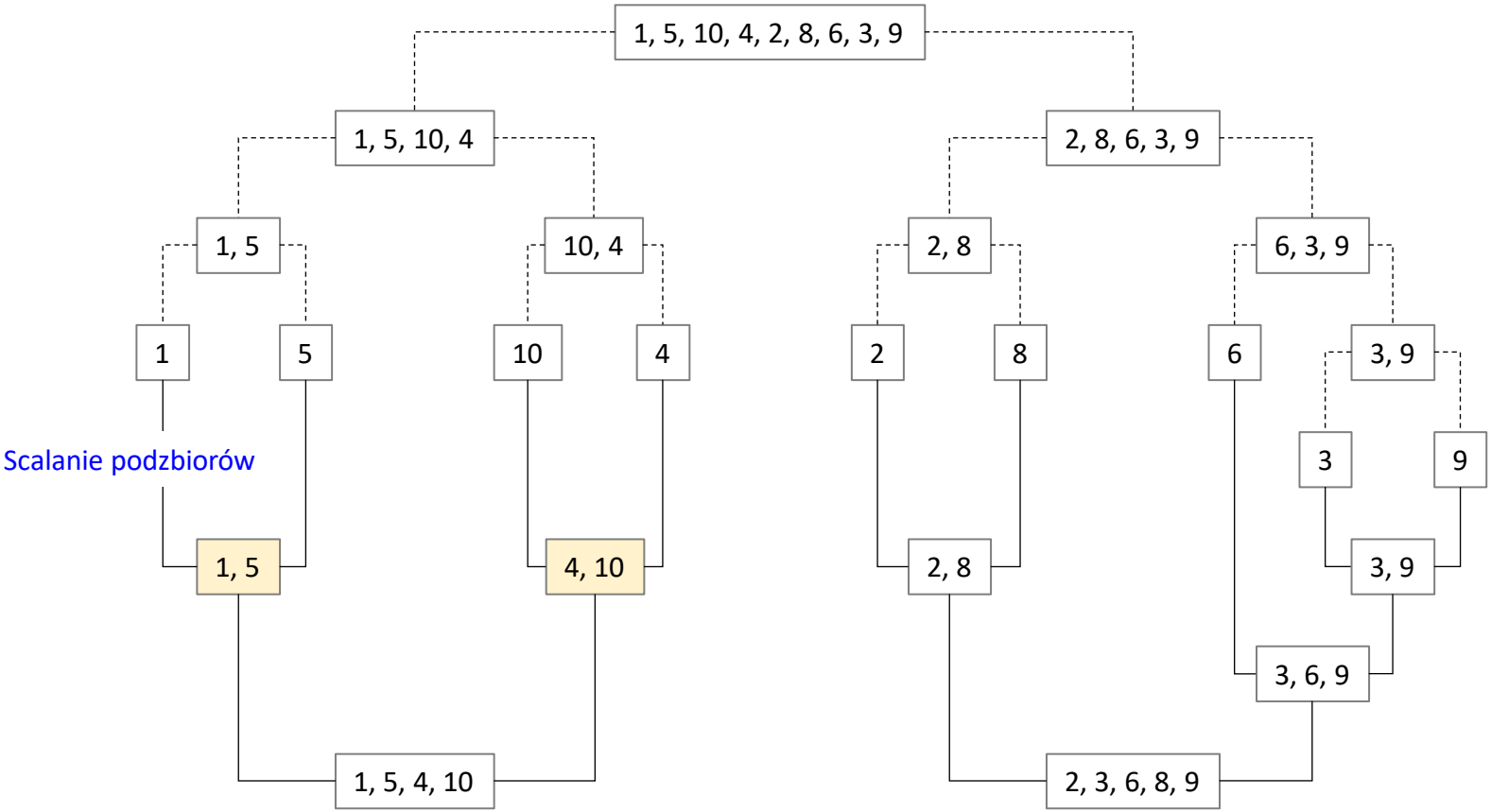



```
i = j = k = 0
while (i<sizeof(left) && j<sizeof(right))
       if (left[i] < right[j]) merged[k++]=left[i++]
       else merged[k++]=right[j++];

while (i<sizeof(left) merged[k++]=left[i++];
while (j<sizeof(right) merged[k++]=right[j++];
```

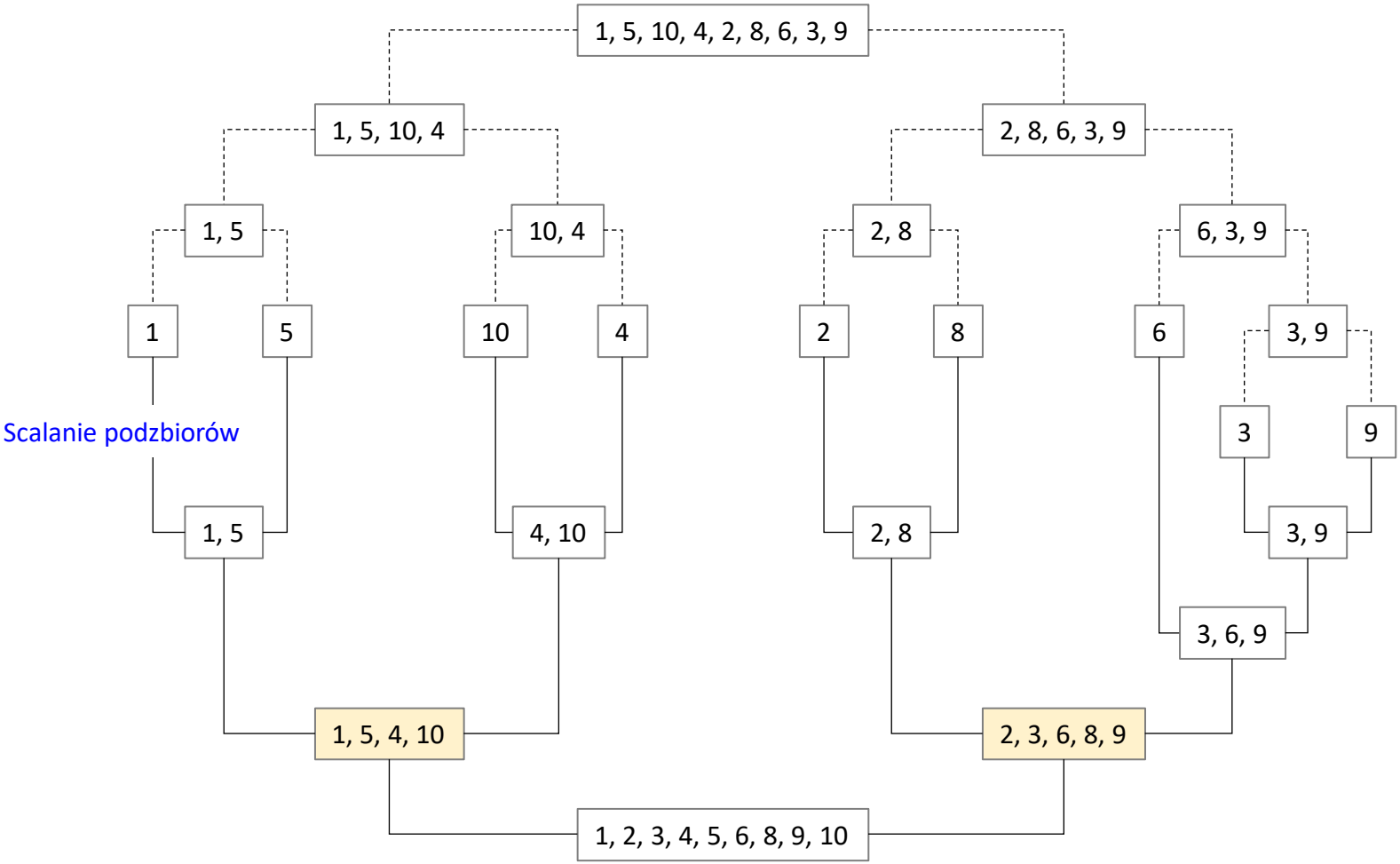
Merge sort: sortowanie przez scalanie

Rekurencyjne dzielenie zbioru na podzbiory

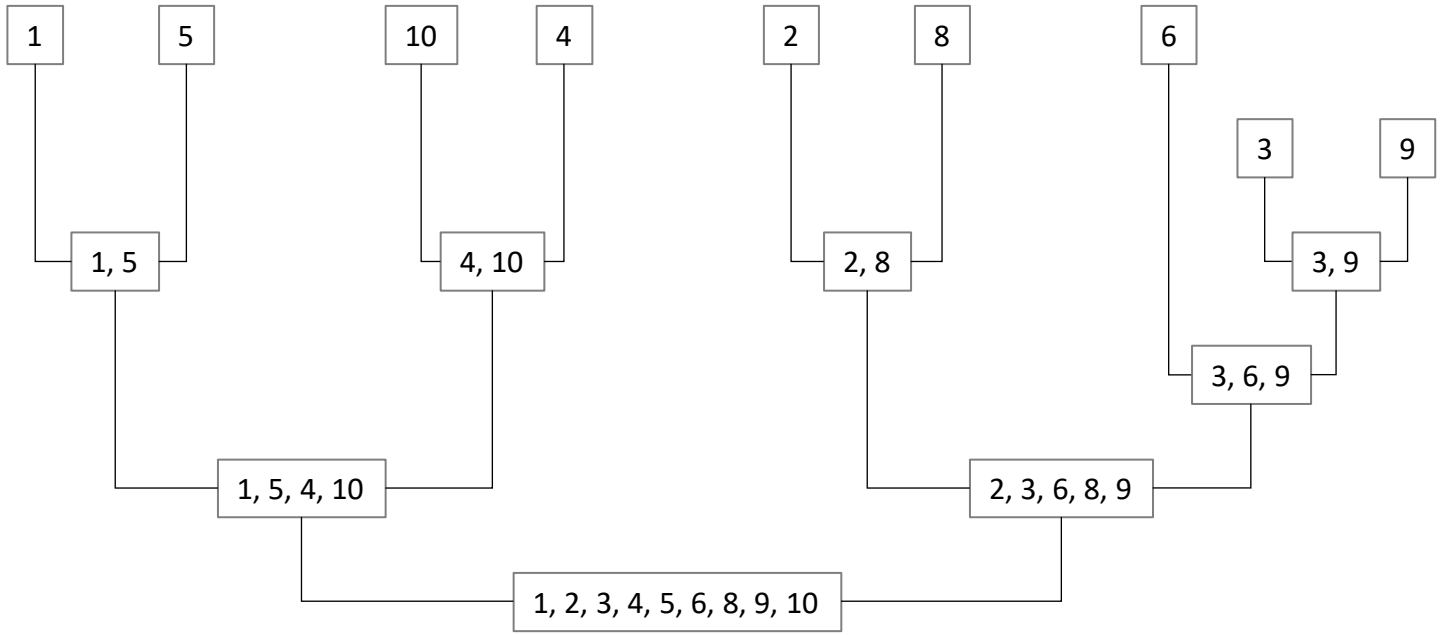


Merge sort: sortowanie przez scalanie

Rekurencyjne dzielenie zbioru na podzbiory



Czy algorytm Merge Sort może wykonać sortowanie na jednej tablicy (wejściowej)?

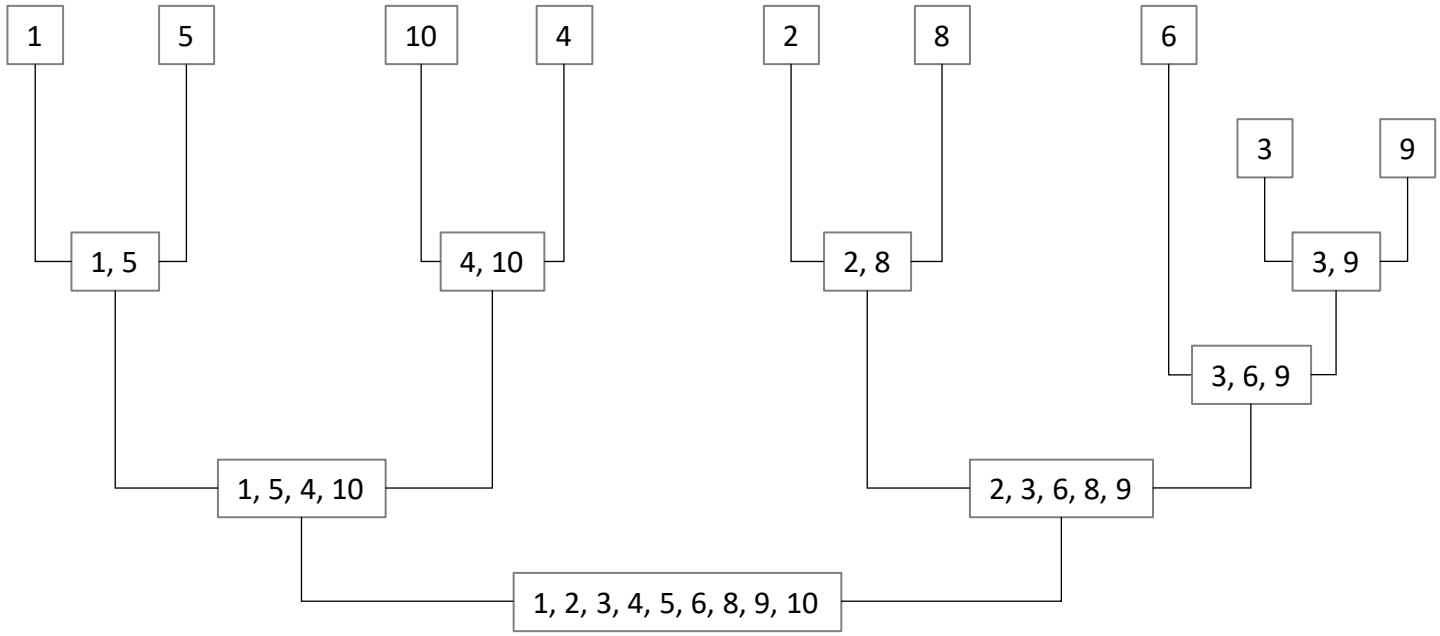


Merge sort: sortowanie przez scalanie

Czy algorytm Merge Sort może wykonać sortowanie na jednej tablicy (wejściowej)?

Nie.

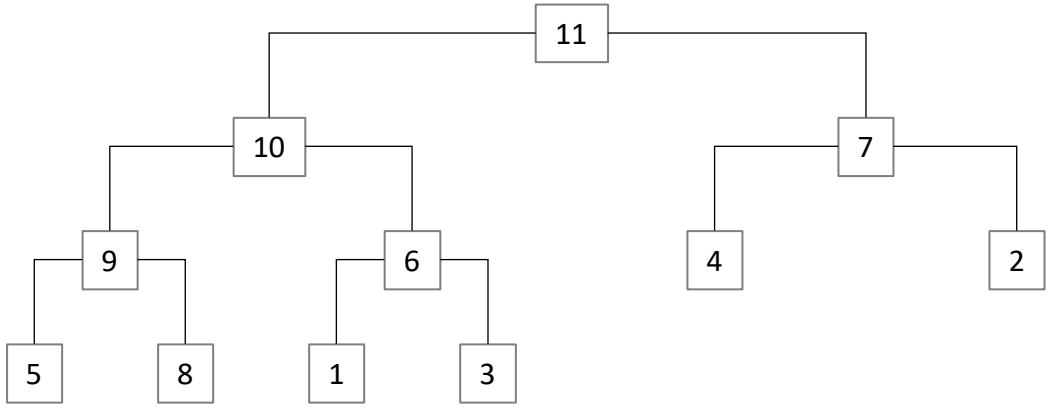
Ten algorytm **nie sortuje w miejscu**: oprócz tablicy wejściowej i stałej liczby zmiennych pomocniczych MS potrzebuje dodatkowych struktur (tablic) do scalania podzbiorów.



Heap sort: sortowanie stogowe (przez kopcowanie)

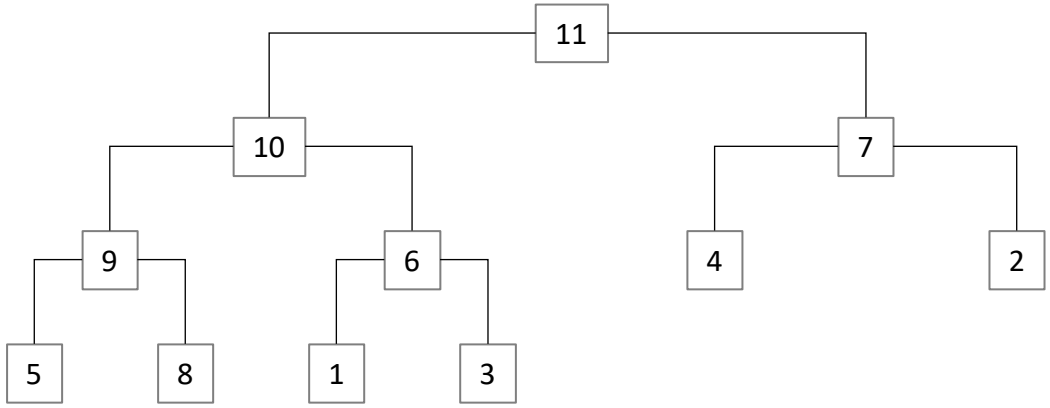
Heap sort: sortowanie stogowe (przez kopcowanie)

Co to jest kopiec (stóg)?



Heap sort: sortowanie stogowe (przez kopcowanie)

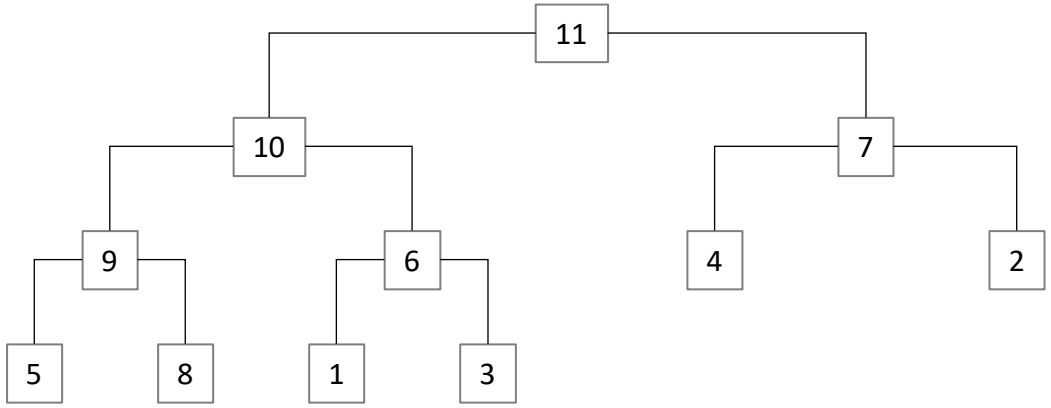
Co to jest kopiec (stóg)?



Właściwości kopca:
- drzewo binarne
- wypełniane poziomami (począwszy od pierwszego) od lewej do prawej
- ma właściwość kopca: wartość w każdym węźle jest większa niż w jego węzłach potomnych

Heap sort: sortowanie stogowe (przez kopcowanie)

Co to jest kopiec (stóg)?

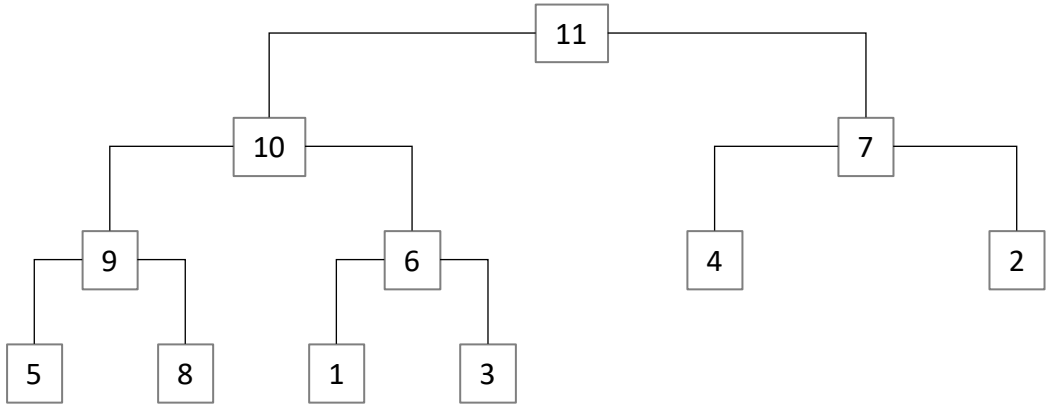


Kopiec w tablicy:

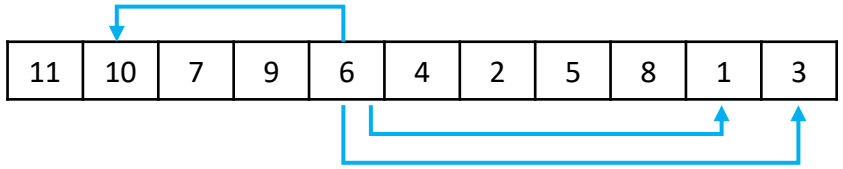
11	10	7	9	6	4	2	5	8	1	3

Heap sort: sortowanie stogowe (przez kopcowanie)

Co to jest kopiec (stóg)?



Kopiec w tablicy:

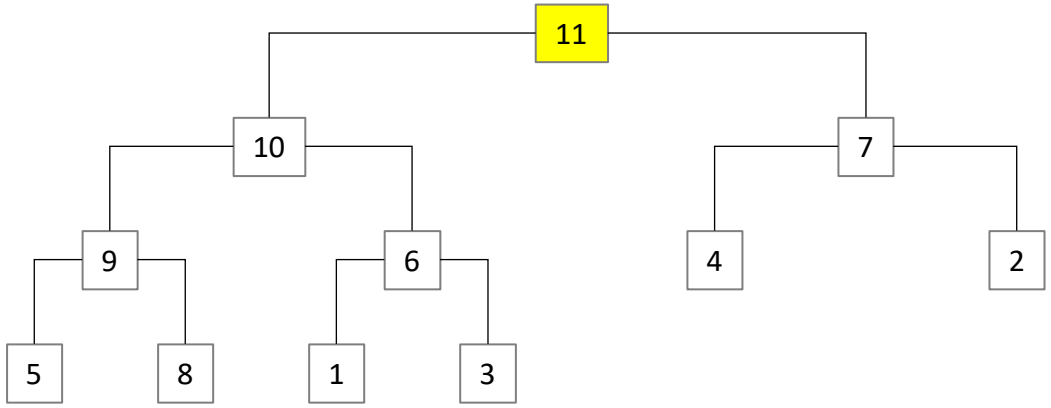


Jak w tablicy znaleźć indeks rodzica, prawego i lewego potomka węzła w:

Rodzic(w) = tab[w/2]
Lewy(w) = tab[2w]
Prawy(w) = tab[2w+1]

Heap sort: sortowanie stogowe (przez kopcowanie)

Wiemy, gdzie znaleźć **wartość maksymalną** w strukturze kopca.

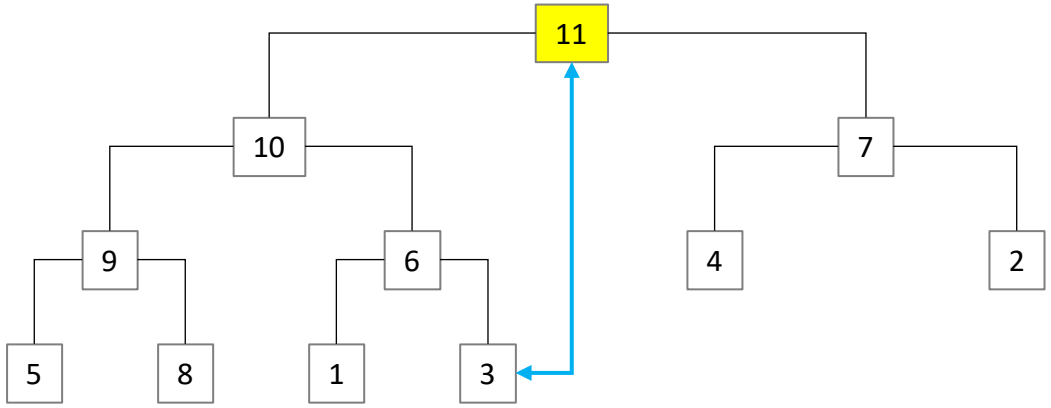


Kopiec w tablicy:

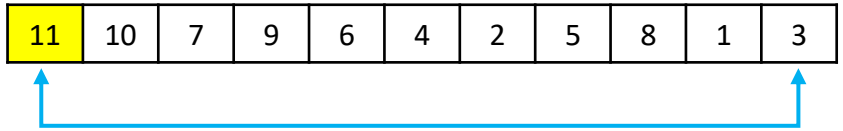
11	10	7	9	6	4	2	5	8	1	3

Heap sort: sortowanie stogowe (przez kopcowanie)

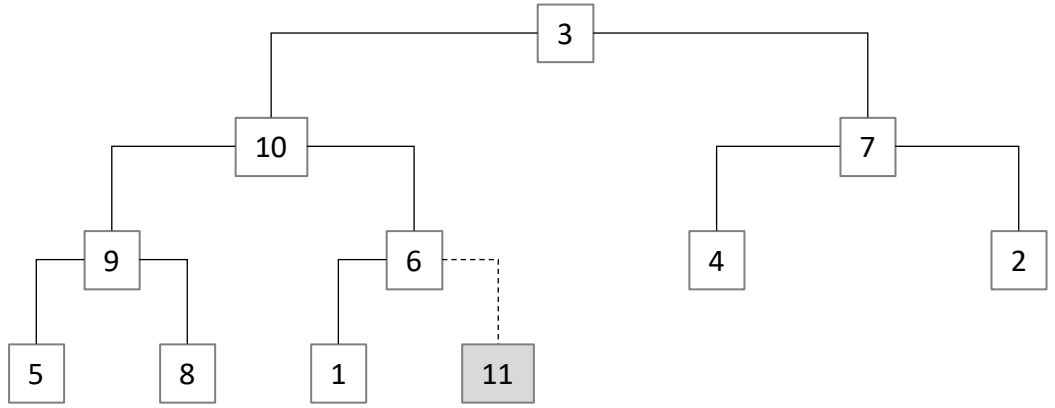
Wiemy, gdzie znaleźć **wartość maksymalną** w strukturze kopca.



Wiemy, gdzie powinna być wartość maksymalna w posortowanym ciągu.

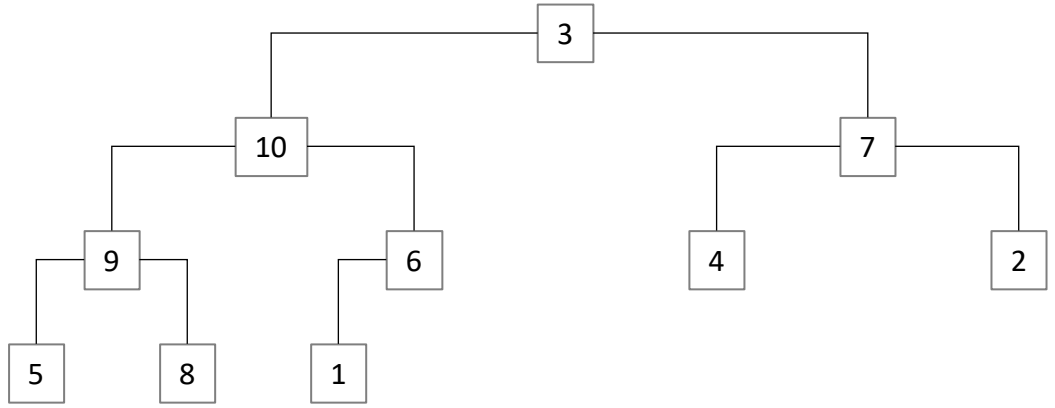


Heap sort: sortowanie stogowe (przez kopcowanie)



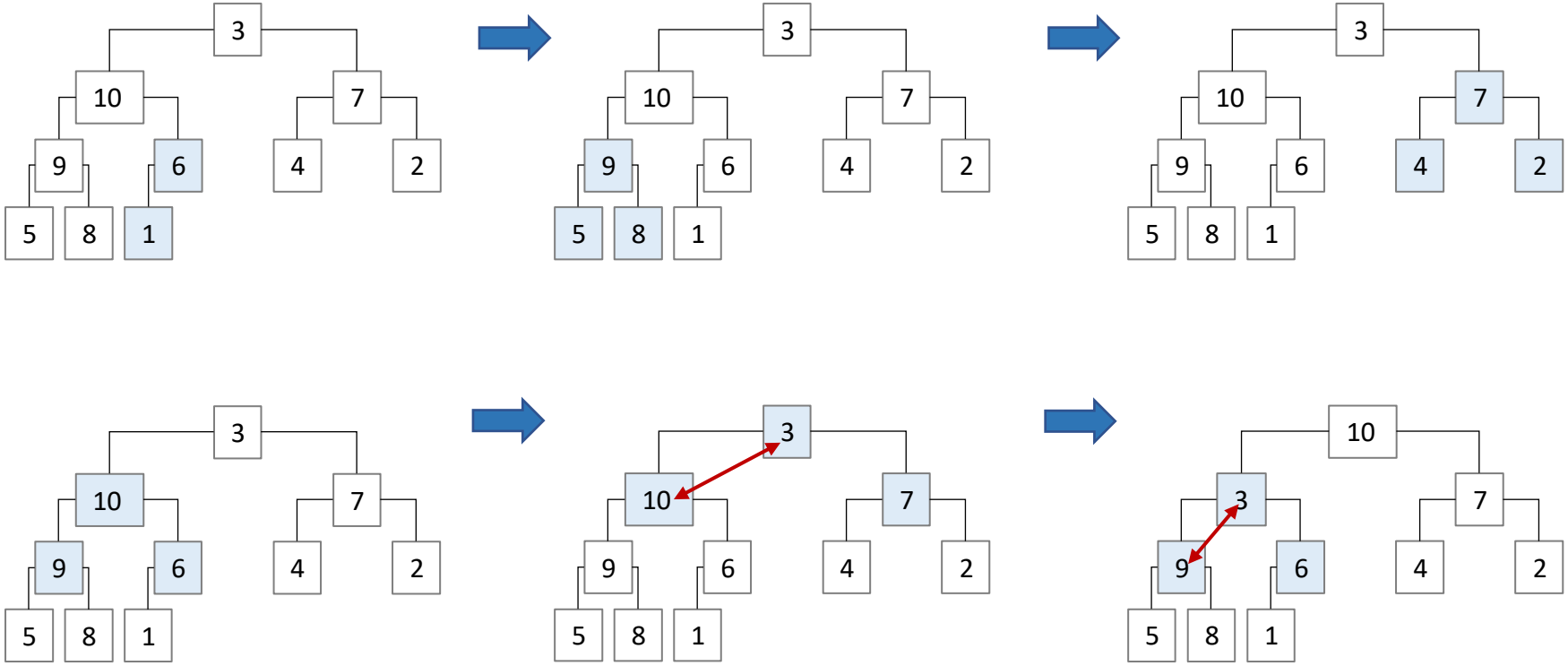
Heap sort: sortowanie stogowe (przez kopcowanie)

Przywracamy własność kopca (nie interesują nas elementy posortowane)



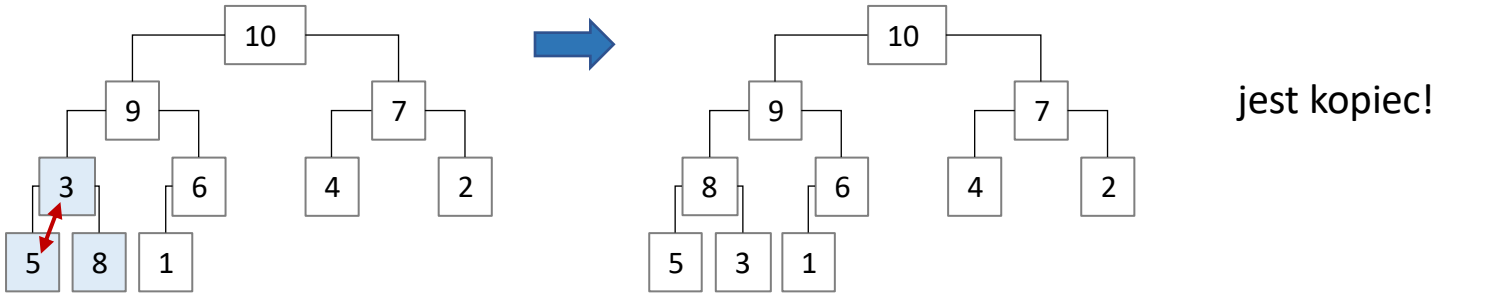
Heap sort: sortowanie stogowe (przez kopcowanie)

Przywracamy własność kopca

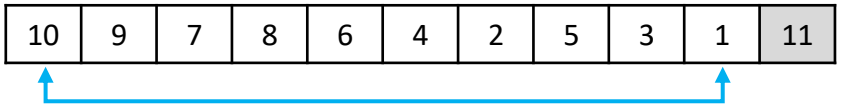


Heap sort: sortowanie stogowe (przez kopcowanie)

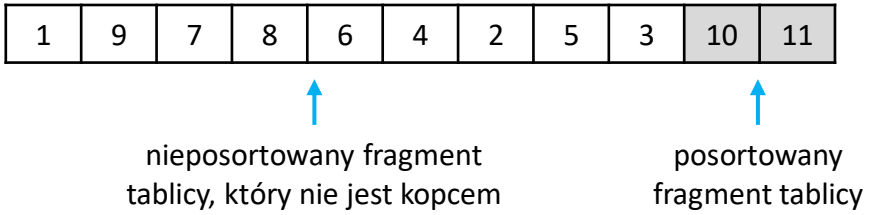
Przywracamy własność kopca



jest kopiec!



Znowu zamieniamy korzeń z ostatnim liściem itd



Heap sort: sortowanie stogowe (przez kopcowanie)

Sortowanie stogowe:

1. Zbuduj kopiec (przywracanie własności kopca na losowym ciągu)
2. Zamień korzeń z „ostatnim” liściem (ostatni element tablicy)
3. Przywróć własność kopca
4. Jeśli w tablicy są elementy nieposortowane idź do punktu (2)

Heap sort: sortowanie stogowe (przez kopcowanie)

Sortowanie stogowe:

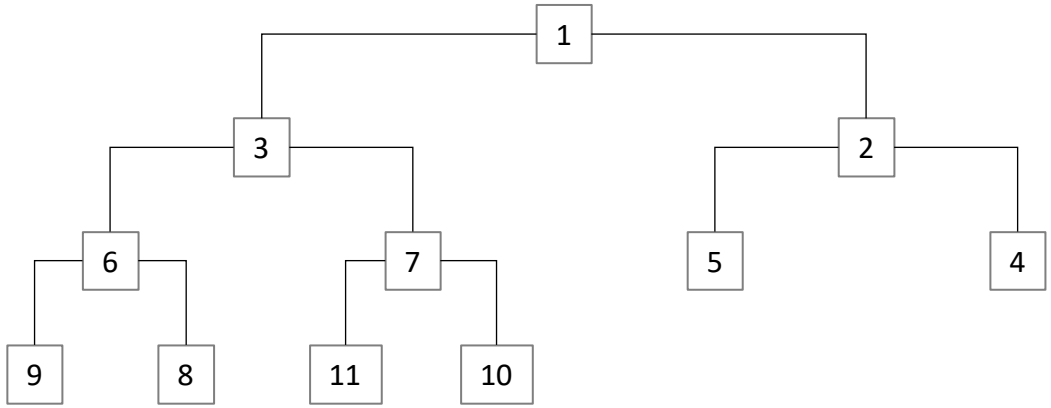
1. Zbuduj kopiec (przywracanie własności kopca na losowym ciągu)
2. Zamień korzeń z „ostatnim” liściem (ostatni element tablicy)
3. Przywróć własność kopca
4. Jeśli w tablicy są elementy nieposortowane idź do punktu (2)

A jeśli chcemy sortować malejąco …

Heap sort: sortowanie stogowe (przez kopcowanie)

Budujemy kopiec odwrócony = kopiec minimalny

Ciąg wejściowy: 11, 10, 5, 8, 7, 2, 4, 9, 6, 1, 3,



1	3	2	6	7	5	4	9	8	11	10

Shell sort: sortowanie Shella (metodą malejących przyrostów)

Jak zoptymalizować algorytm Insertion Sort…

Shell sort: sortowanie Shella (metodą malejących przyrostów)

4	14	7	2	1	10	3	8	11	5	12

n = 11

Shell sort: sortowanie Shella (metodą malejących przyrostów)

4	14	7	2	1	10	3	8	11	5	12

n = 11

Posortujmy podzbiory zawierające co k-ty element algorytmem Insertion Sort

k = n/2 = 5

Każdy podzbiór oznaczono innym kolorem:

4	14	7	2	1	10	3	8	11	5	12

Shell sort: sortowanie Shella (metodą malejących przyrostów)

4	14	7	2	1	10	3	8	11	5	12

n = 11

Posortujmy podzbiory zawierające co k-ty element algorytmem Insertion Sort

k = n/2 = 5

Każdy podzbiór oznaczono innym kolorem:

4	14	7	2	1	10	3	8	11	5	12

sortowanie

4					10					12

4	3				10	14				12

4	3	7			10	14	8			12

4	3	7	2		10	14	8	11		12

4	3	7	2	1	10	14	8	11	5	12

Tak wygląda 5-posortowany ciąg wejściowy (ciąg posortowany z przyrostem k=5).

Shell sort: sortowanie Shella (metodą malejących przyrostów)

Ciąg posortowany z przyrostem k=5:

4	3	7	2	1	10	14	8	11	5	12

Zmniejszamy k:

k = k/2 = 2

Znowu posortujmy podzbiory zawierające co k-ty element algorytmem Insertion Sort

Każdy podzbiór oznaczono innym kolorem:

4	3	7	2	1	10	14	8	11	5	12

sortowanie

1		4		7		11		12		14

1	2	4	3	7	5	11	8	12	10	14

Tak wygląda 2-posortowany ciąg wejściowy (ciąg posortowany z przyrostem k=2).

Shell sort: sortowanie Shella (metodą malejących przyrostów)

Ciąg posortowany z przyrostem k=2:

1	2	4	3	7	5	11	8	12	10	14

Zmniejszamy k:

k = k/2 = 1

Znowu posortujmy podzbiory zawierające co k-ty element algorytmem Insertion Sort

Teraz cały ciąg należy do jednego zbioru. Uruchamiamy IS:

1	2	3	4	5	7	8	10	11	12	14

Shell sort: sortowanie Shella (metodą malejących przyrostów)

Inne warianty sortowania Shella:

- różne sposoby wyznaczania przyrostu (dla każdego z nich Shell Sort ma inną złożoność obliczeniową)

- zastosowanie innego algorytmu bazowego (np. Bubble Sort zamiast Insertion Sort)

Counting sort: sortowanie przez zliczanie

Sortowanie bez porównywania elementów…

Counting sort: sortowanie przez zliczanie

Ciąg wejściowy: 4, 1, 3, 6, 1, 5, 3, 5, 1, 7

n = 10

Warunek 1: sortujemy liczby całkowite dodatnie.

Warunek 2: znamy maksymalną wartość, jaka pojawia się w ciągu wejściowym.

MAX = 7

Counting sort: sortowanie przez zliczanie

Tablica wejściowa A:

0	4	1	3	6	1	5	3	5	1	7

n = 10

Warunek 1: sortujemy liczby całkowite dodatnie.

Warunek 2: znamy maksymalną wartość, jaka pojawia się w ciągu wejściowym.

MAX = 7

Tablica pomocnicza B (do zliczania):

0	1	2	3	4	5	6	7

Potrzebna będzie dodatkowa tablica o długości MAX do zliczania wartości.

Counting sort: sortowanie przez zliczanie

Tablica wejściowa A:

0	4	1	3	6	1	5	3	5	1	7

n = 10
MAX = 7

Tablica pomocnicza B przed rozpoczęciem zliczania:

0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7

1. Zliczanie wartości z tablicy wejściowej

```
for (i = 0; n; ++i)
     ++B[A[i]];
```

Tablica pomocnicza B po zliczeniu:

0	3	0	2	1	2	1	1
0	1	2	3	4	5	6	7

Counting sort: sortowanie przez zliczanie

Tablica wejściowa A:

0	4	1	3	6	1	5	3	5	1	7

n = 10
MAX = 7

Tablica pomocnicza B przed rozpoczęciem zliczania:

0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7

1. Zliczanie wartości z tablicy wejściowej

```
for (i = 0; i <= n; ++i)
     ++B[A[i]];
```

Tablica pomocnicza B po zliczeniu:

0	3	0	2	1	2	1	1
0	1	2	3	4	5	6	7

2. Modyfikacja tablicy pomocniczej

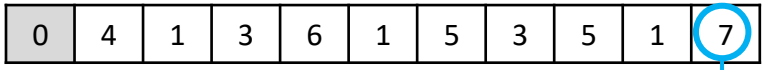
```
for (i = 1; i <= MAX; ++i)
     B[i] + = B[i-1];
```

Tablica pomocnicza B po modyfikacji:

0	3	3	5	6	8	9	10
0	1	2	3	4	5	6	7

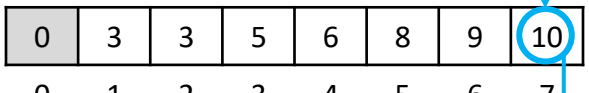
Counting sort: sortowanie przez zliczanie

Tablica wejściowa A:



n = 10
MAX = 7

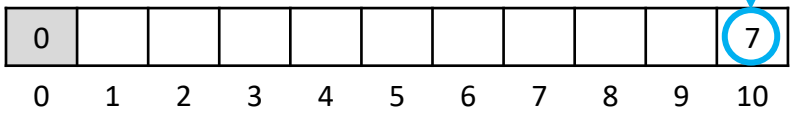
Tablica pomocnicza B:



3. Wypełnianie tablicy wyjściowej

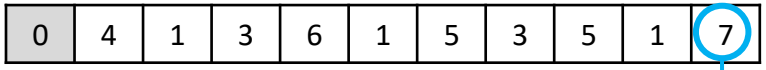
```
for (i = n; i >= 0; i--)
     C[B[A[i]]] = A[i];
```

Tablica wyjściowa C:

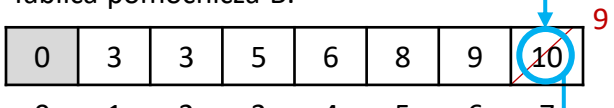


Counting sort: sortowanie przez zliczanie

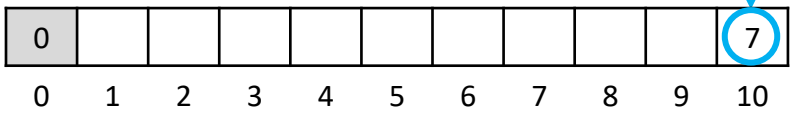
Tablica wejściowa A:



Tablica pomocnicza B:



Tablica wyjściowa C:



n = 10
MAX = 7

3. Wypełnianie tablicy wyjściowej

```
for (i = n; i >= 0; i--)
     C[B[A[i]]] = A[i];
      --B[A[i]];
```

Counting sort: sortowanie przez zliczanie

Tablica wejściowa A:

0	4	1	3	6	1	5	3	5	1	7

n = 10
MAX = 7

Zawartość tablic B i C po zakończeniu sortowania:

Tablica pomocnicza B:

0	0	3	3	5	6	8	9
0	1	2	3	4	5	6	7

Tablica wyjściowa C:

0	1	1	1	3	3	4	5	5	6	7
0	1	2	3	4	5	6	7	8	9	10

Counting Sort w tej wersji sortuje stabilnie.

Stabilność algorytmu sortowania

Algorytm stabilny
Zawsze zachowuje wejściowy porządek elementów o tej samej wartości.

Tablica wejściowa:

2	4	1	3	6	1	5	8	9	1	7

Tablica wyjściowa:

1	1	1	2	3	4	5	6	7	8	9

Algorytm niestabilny
Nie gwarantuje, że elementy o tej samej wartości będą względem siebie uporządkowane na wyjściu tak samo jak w ciągu wejściowym.

Tablica wejściowa:

2	4	1	3	6	1	5	8	9	1	7

Tablica wyjściowa:

1	1	1	2	3	4	5	6	7	8	9

Stabilność algorytmu sortowania jest istotna jeśli sortujemy złożone dane hierarchicznie.

Właściwości algorytmów sortowania: stabilność

Algorytm sortowania jest stabilny jeśli liczby o tych samych wartościach występują w tablicy wynikowej w takiej samej kolejności jak w tablicy początkowej [Cormen *i in*.].

Algorytm	Czy jest stabilny?
Insertion Sort	TAK
Selection Sort*	NIE
Bubble Sort	TAK
Heap Sort	NIE
Merge Sort	TAK
Quick Sort	NIE
Shell Sort	NIE
Counting Sort*	TAK

* istnieje wersja stabilna i niestabilna, powszechnie implementuje się wersję jak zaznaczono w tabeli

Właściwości algorytmów sortowania: sortowanie w miejscu

Algorytm sortuje w miejscu jeśli podczas działania algorytmu sortowane elementy są cały czas przechowywane w tej samej tablicy wejściowej z wyjątkiem stałej liczby elementów. Po zakończeniu działania algorytmu tablica wejściowa zawiera posortowany ciąg [Cormen *i in*.].

Algorytm	Czy sortuje w miejscu?
Insertion Sort	TAK
Selection Sort	TAK
Bubble Sort	TAK
Heap Sort	TAK
Merge Sort	NIE
Quick Sort	TAK
Shell Sort	TAK
Counting Sort	NIE

Właściwości algorytmów sortowania: złożoność obliczeniowa

Algorytm	Złożoność obliczeniowa		
	Optymistyczna	Średnia	Pesymistyczna
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Heap Sort	$O(n \log_2 n)$	$O(n \log_2 n)$	$O(n \log_2 n)$
Merge Sort	$O(n \log_2 n)$	$O(n \log_2 n)$	$O(n \log_2 n)$
Quick Sort	$O(n \log_2 n)$	$O(n \log_2 n)$	$O(n^2)$
Shell Sort	zależna od metody wyznaczania przyrostów		
Counting Sort	$O(n+k)$	$O(n+k)$	$O(n+k)$

n – liczba sortowanych elementów
k – maksymalna wartość występująca w ciągu wejściowym