Algorytmy sortujące

Sortowanie przez scalanie

Sortowanie przez zliczanie

Sortowanie przez scalanie

Scalanie ciągów (łączenie) jest operacją połączenia dwóch uporządkowanych rosnąco (malejąco) ciągów elementów w jeden ciąg wynikowy, również uporządkowany rosnąco (malejąco).

W informatyce sortowanie przez scalanie (ang. **merge sort**), to **rekurencyjny algorytm** sortowania danych.

Odkrycie algorytmu przypisuje się **Johnowi von Neumannowi** - matematykowi, inżynierowi chemii, fizykowi i informatyk. Wniósł on znaczący wkład do wielu dziedzin matematyki, był głównym twórcą teorii gier, teorii automatów, i stworzył formalizm matematyczny mechaniki kwantowej.

John von Neumann
(1903 – 1957)

Sortowanie przez scalanie

Najlepszy opis sortowania przez scalanie opiera się na **rekurencji** ilustruje równocześnie bardzo korzystne zastosowanie techniki „**dziel i zwyciężaj**”.

Dziel

Dzielimy n-elementowy ciąg na dwa podciągi po n=2 elementów każdy

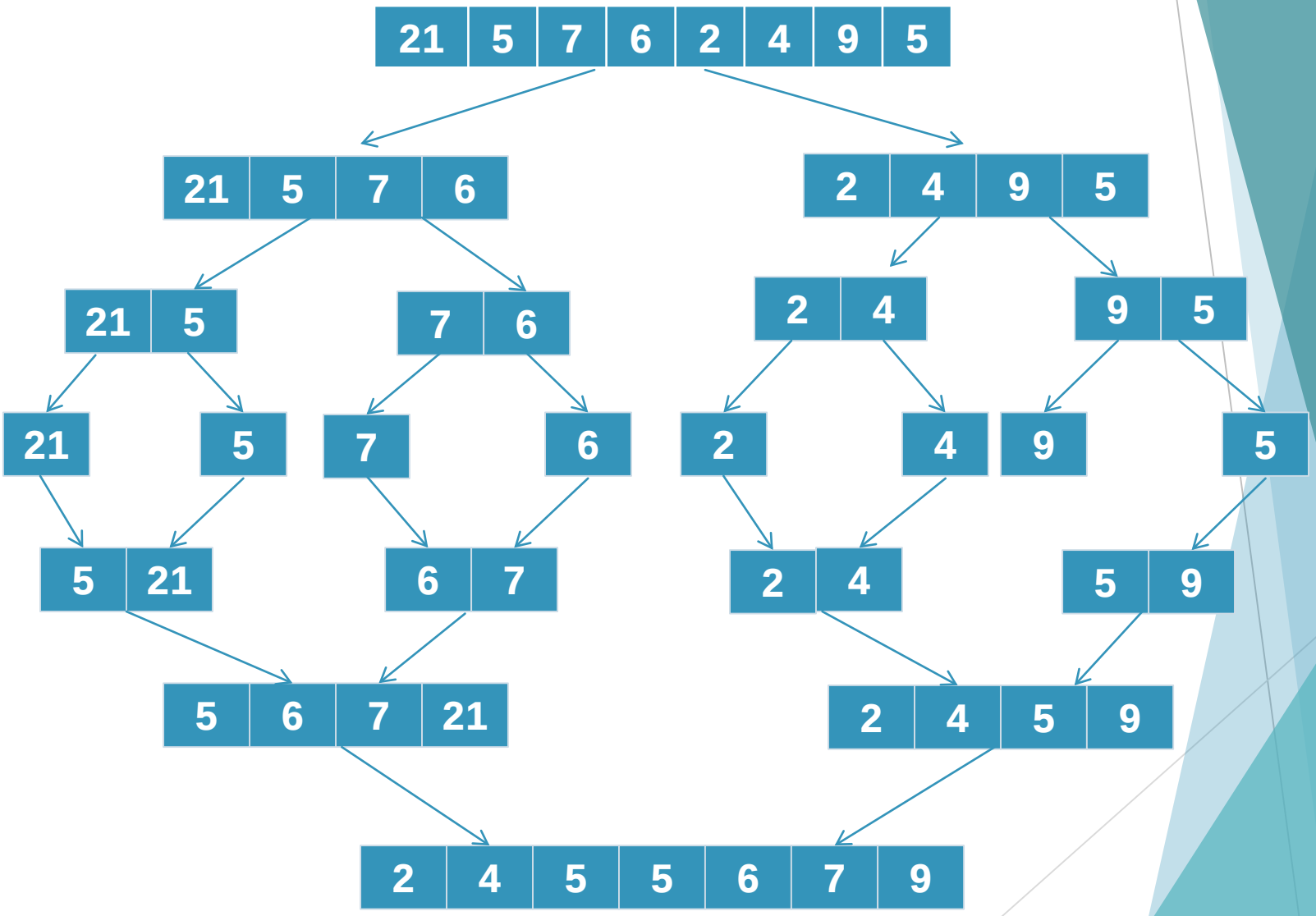
Zwyciężaj

Sortujemy otrzymane podciągi, używając rekurencyjnie sortowania przez scalanie

Scalaj

Łączymy posortowane podciągi w jeden posortowany ciąg. Każdy ciąg o długości 1 jest posortowany

Ilustracja działania algorytmu sortowania przez scalanie



Scalanie zbiorów uporządkowanych

Podstawową operacją algorytmu jest **scalanie dwóch zbiorów uporządkowanych** w jeden zbiór również uporządkowany. Operację scalania realizujemy wykorzystując pomocniczy zbiór, w którym będziemy tymczasowo odkładać scalane elementy dwóch zbiorów.

Zasada scalania zbiorów:

- Przygotowujemy pusty, tymczasowy zbiór

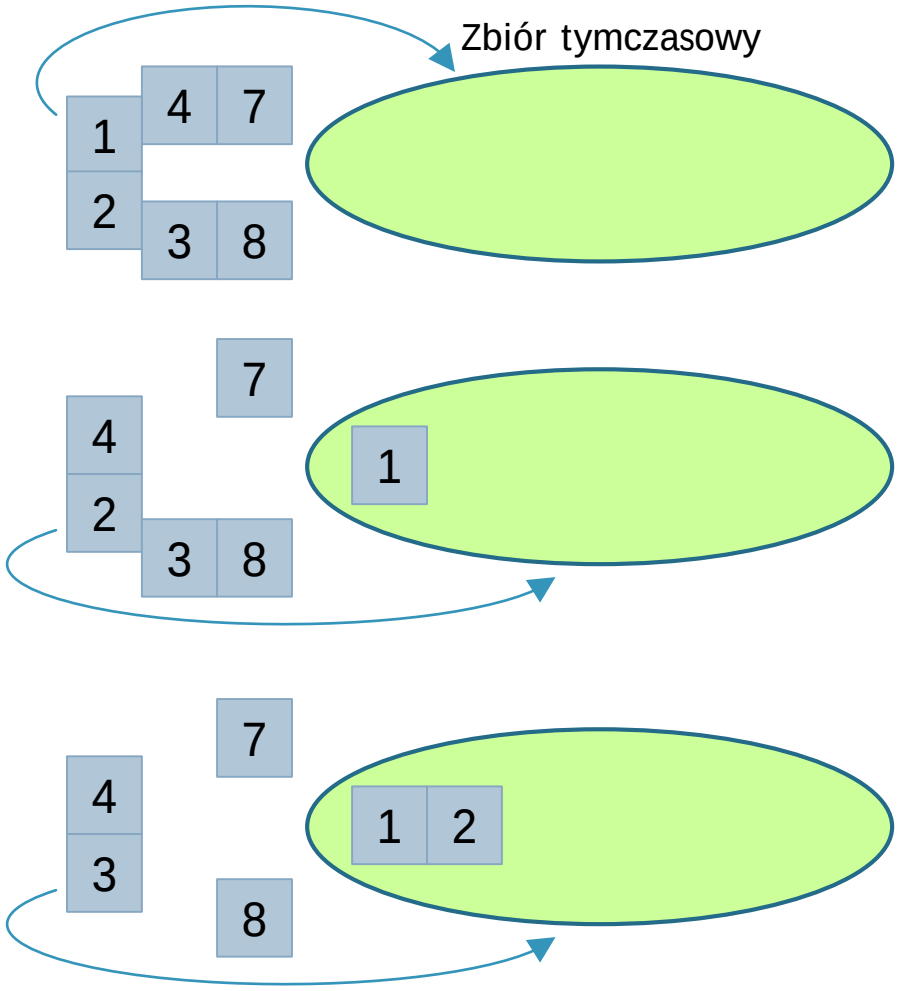
- Porównujemy pierwsze elementy zbiorów i w zbiorze tymczasowym umieszczamy mniejszy z elementów, usuwając go ze scalanego zbioru. Czynność powtarzamy do momentu opróżnienia jednego ze zbiorów

- W tymczasowym zbiorze umieszczamy zawartość tego scalanego zbioru, który zawiera jeszcze elementy

- Do zbioru wynikowego wpisujemy zawartość zbioru tymczasowego. Koniec algorytmu

Scalanie zbiorów uporządkowanych - przykład

Scalmy dwa uporządkowane zbiory [1 4 7] i [2 3 8]

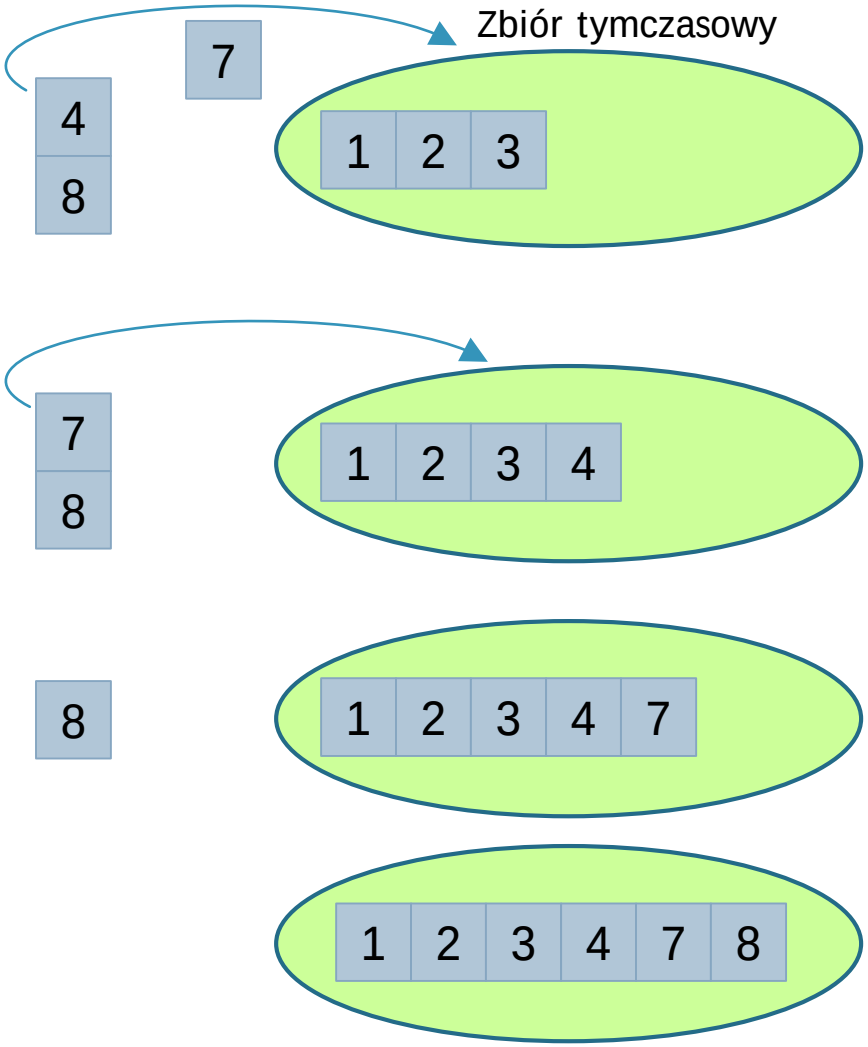


Pierwszym krokiem jest porównanie ze sobą najmniejszych elementów ze scalanych zbiorów.

W zbiorze tymczasowym umieszczamy najmniejszy element, usuwając go ze zbioru sortowanego. W naszym przypadku będzie to liczba 1.

Porównujemy kolejne elementy zbiorów.
Za każdym razem najmniejszy element zostaje przeniesiony do zbioru tymczasowego.

Scalanie zbiorów uporządkowanych - przykład



Poszczególne kroki porównywania najmniejszych elementów zbiorów scalanych powtarzamy do momentu gdy jeden ze zbiorów zostanie pusty.

Zakończyliśmy proces porównywania poszczególnych elementów.
Do zbioru tymczasowego wprowadzamy pozostałe elementy z drugiego zbioru.

Uzyskaliśmy zbiór uporządkowany, którego zawartość możemy przepisać do zbioru docelowego.

Sortowanie przez zliczanie

Kolejny algorytm, którego zasadę działania poznamy to sortowanie przez zliczanie. W metodzie tej stosowane są pewne założenia wobec sortowanego zbioru.

Sortowanie przez zliczanie, oprócz zalety w postaci szybkości, ma swoje dwie poważne wady. Po pierwsze - do sortowania tego typu potrzebna jest dodatkowa pamięć (czyli nie jest to sortowanie w miejscu), a po drugie - tym sposobem można sortować tylko liczby całkowite z ograniczonego zakresu.

Zasada działania algorytmu - w tablicy A mamy zapisane wszystkie liczby do posortowania. W tablicy B będziemy zapisywać ile razy występują elementy z tablicy A. Na początku tablicę B wypełniamy zerami. Sprawdzamy kolejne pola w tablicy A. Jeśli element numer *i* jest równy np. 10, to powiększamy o 1 dziesiąty element tablicy B.

Ogólnie idea algorytmu polega na sprawdzeniu **ile wystąpień danego klucza występuje w sortowanej tablicy.**

Sortowanie przez zliczanie - przykład

Posortujmy rosnąco dany zbiór liczb:

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

Następnie, dla każdej wartości w zbiorze dopisujemy licznik ustawiony na 0.

[0:0] [1:0] [2:0] [3:0] [4:0] [5:0] [6:0] [7:0] [8:0] [9:0]

Kolejnym krokiem, jest zliczanie poszczególnych elementów w zbiorze i przypisywanie ich do konkretnych wartości liczbowych, np. dla liczby 4, każdy element w zbiorze sortowanym o tej wartości zwiększa wartość licznika o 1.

W tym konkretnym przykładzie otrzymamy:

[0:1] [1:3] [2:3] [3:2] [4:2] [5:1] [6:4] [7:1] [8:5] [9:1]

Teraz poczynając od drugiego licznika sumujemy zawartość licznika oraz jego poprzednika i otrzymujemy:

[0:1] [1:4] [2:7] [3:9] [4:11] [5:12] [6:16] [7:17] [8:22] [9:23]

Dzięki tej operacji licznik podaje nam ilość wartości mniejszych lub równych konkretnej wartości ze zbioru, np.:

[2:7] - w zbiorze do sortowania jest siedem wartości mniejszych lub równych 2

[6:16] - w zbiorze do sortowania jest szesnaście wartości mniejszych lub równych 6 itd.

Otrzymujemy uporządkowany zbiór elementów, w którym stan licznika określa ostatnią pozycję w zbiorze danego elementu :

Sortowana liczba	0	1	1	1	2	2	2	3	3	4	4	5	6	6	6	6	7	8	8	8	8	8	9
Pozycja w zbiorze	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

Sortowanie przez zliczanie - przykład

W zasadzie proces sortowania moglibyśmy zakończyć na tym etapie, jednak jeżeli chcemy aby nasz algorytm był stabilny podczas sortowania wartości „kluczy” (czyli takich wartości, które są przypisane pewnym większym strukturom danych) musimy wykonać **obieg dystrybucyjny.**

Obieg dystrybucyjny ma za zadanie obliczyć **dystrybuantę**, czyli ilość elementów mniejszych lub równych od danej wartości.

W tym celu ponownie przeglądamy zbiór danych wejściowych idąc **od ostatniego elementu do pierwszego –** inaczej algorytm nie byłby stabilny. Po kolei, każdy element umieszczamy w zbiorze wynikowym na pozycji równej zawartości licznika dla tego elementu. Po wykonaniu tej operacji licznik zmniejszamy o 1.

Na przykładzie będzie to wyglądało w następujący sposób: np. dla liczby 4 – jej licznik ma zawartość **[4:11]** – dlatego liczbę 4 umieszczamy w zbiorze wynikowym na pozycji 11 i zmniejszamy stan licznika o 1, w wyniku czego otrzymamy **[4:10]**. Kolejna liczba 4 trafi na pozycję 10, itd.

Dla liczby 6 wartość licznika wynosi **[6:16]** – czyli 6 będzie na 16 pozycji, zmniejszamy licznik o 1 i otrzymujemy **[6:15]**.

Analogicznie postępujemy ze wszystkimi liczbami w sortowanym zbiorze, do momentu za zbiór zostanie całkowicie uporządkowany w sposób rosnący.

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

Bibliografia

- Sysło M.: *Algorytmy,* WSiP, Warszawa 1997
- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein: *Wprowadzenie do algorytmów,* Wydawnictwo Naukowo-Techniczne, wyd. VI 2004
- Wróblewski P.: Algorytmy, struktury danych i techniki programowania, Wyd. Helion, 2003
- www.eioba.pl
- www.pl.wikipedia.org
- www.algorytm.org
- www.encyklopedia.pwn.pl
- www.edu.i-lo.tarnow.pl