Sortowanie przez scalanie (*merge sort*)

Sortowanie przez scalanie należy do algorytmów, które wykorzystują metodę "**dziel i zwyciężaj**". Odkrycie algorytmu przypisuje się Johnowi von Neumannowi.
Złożoność algorytmu wynosi $n \cdot \log n$, a więc jest on znacznie wydajniejszy niż sortowanie bąbelkowe, przez wstawianie czy przez selekcję, gdzie złożoność jest kwadratowa. Żeby zrozumieć zasadę działania przyjrzyjmy się najpierw dwóm posortowanym tablicom:

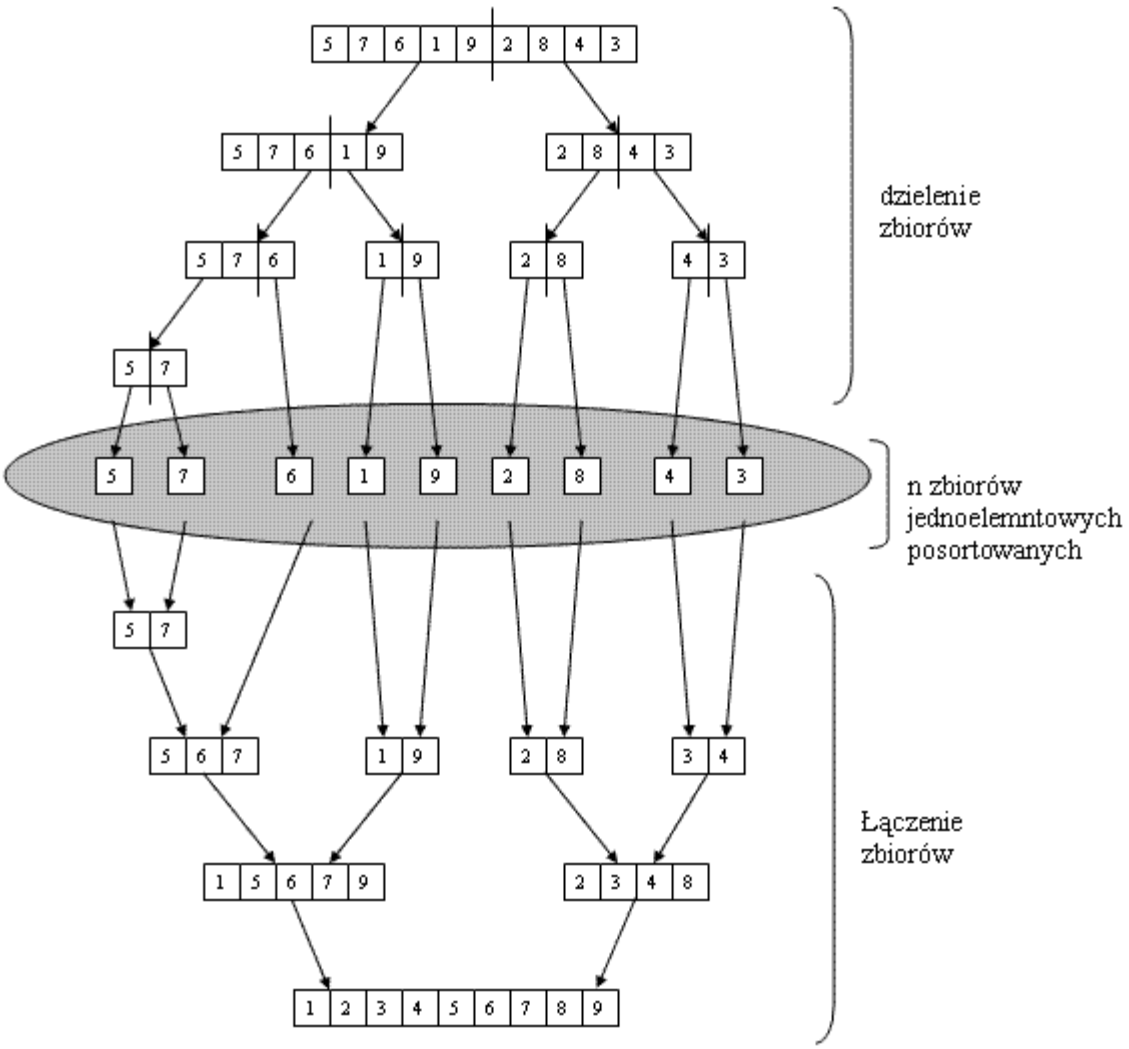
tablica 1: **2 3 7 9 10**

tablica 2: **1 3 4 8 11**

Zauważmy, że możemy **liniowo** scalić te dwa ciągi liczb i uzyskać jedną posortowaną tablicę postępując ze schematem:

- ustawiamy liczniki na początki tablic posortowanych,
- następnie porównujemy elementy i mniejszy lub równy element wskakuje jako pierwszy w scalonej tablicy,
- zwiększamy licznik w tej tablicy, z której "zabraliśmy element",
- czynność powtarzamy aż do wyczerpania danych z obu tablic.

Ideą działania algorytmu jest dzielenie zbioru danych na mniejsze zbiory, aż do uzyskania *n* zbiorów jednoelementowych, które same z siebie są posortowane :), następnie zbiory te są łączone w coraz większe zbiory posortowane, aż do uzyskania jednego, posortowanego zbioru *n*-elementowego. Etap dzielenia nie jest skomplikowany, dzielenie następuje bez sprawdzania jakichkolwiek warunków. Dzięki temu, w przeciwieństwie do algorytmu QuickSort, następuje pełne rozwinięcie wszystkich gałęzi drzewa. Z kolei łączenie zbiorów posortowanych wymaga odpowiedniego wybierania poszczególnych elementów z łączonych zbiorów z uwzględnieniem faktu, że wielkość zbioru nie musi być równa (parzysta i nieparzysta ilość elementów), oraz tego, iż wybieranie elementów z poszczególnych zbiorów nie musi następować naprzemiennie, przez co jeden zbiór może osiągnąć swój koniec wcześniej niż drugi. Robi się to w następujący sposób. Kopiujemy zawartość zbioru głównego do struktury pomocniczej. Następnie, operując wyłącznie na kopii, ustawiamy wskaźniki na początki kolejnych zbiorów i porównujemy wskazywane wartości. Mniejszą wartość wpisujemy do zbioru głównego i przesuwamy odpowiedni wskaźnik o 1 i czynności powtarzamy, aż do momentu, gdy jeden ze wskaźników osiągnie koniec zbioru. Wówczas mamy do rozpatrzenia dwa przypadki, gdy zbiór 1 osiągnął koniec i gdy zbiór 2 osiągnął koniec. W przypadku pierwszym nie będzie problemu, elementy w zbiorze głównym są już posortowane i ułożone na właściwych miejscach. W przypadku drugim trzeba skopiować pozostałe elementy zbioru pierwszego po kolei na koniec. Po zakończeniu wszystkich operacji otrzymujemy posortowany zbiór główny.
Przykładowy przebieg algorytmu ukazany jest na schemacie poniżej:



Implementacja algorytmu przez scalanie:

```
import math
import random

#merge sort - sortowanie przez scalanie
#Rafal Gawlik
#www.algorytm.org


#sortowanie - operacja scalania
def merge(L,start, center,finish):
        """Operacja scalania"""
        i = start
        j = center + 1

        L2 = [] #lista pomocnicza

        #wybieraj odpowiednie elementy z dwoch tablic
        while (i <= center) and (j <= finish):
                if L[j] < L[i]:
```

```
                L2.append(L[j])
                j = j + 1
            else:
                L2.append(L[i])
                i = i + 1

        #jedna z tablic skonczyla sie przepisz reszte z pozostalej
        if i <= center:
            while i <= center:
                L2.append(L[i])
                i = i + 1
        else:
            while j <= finish:
                L2.append(L[j])
                j = j + 1

        #przepisz wyniki z tablicy tymczasowej
        s = finish - start + 1
        i = 0
        while i < s:
            L[start + i] = L2[i]
            i = i + 1

        return L

#sortowanie przez scalanie
def merge_sort(L, start, finish):
        """sortowanie merge sort"""
        if start != finish:

            #dzielimy dablice do konca
            center = int(math.floor((start + finish)/2))
            #na lewo
            merge_sort(L, start, center)
            #na prawo
            merge_sort(L,center+1,finish)

            #operacja scalania
            merge(L,start, center,finish)
        return L

#generuje dane wejsciowe w porzadku malejacym
def malejaco(n):
    L = []
    for i in range(0,n):
        L.append(n-i-1)
    return L

#generuje dane wejsciowe w porzadku losowym
def losowe(n):
    L = []
    a = range(0,n)
    for i in range(0,n):
        L.append(random.choice(a))
    return L
```

```
#przyklad uzycia
n = input('Podaj ilosc danych: ')
w = input('Dane wjsciowe: Losowo/Malejaco [1/2]: ')

if int(w) == 1:
	L = losowe(int(n))
	print(L)
elif int(w) == 2:
	L = malejaco(int(n))
	print(L)
else:
	print('napisales bzdure')

L = merge_sort(L,0,len(L)-1)

print('Po posortowaniu:')
print(L)
```