SDIZO-SORTOWANIE STOGOWE

```
void heap_create_dn(int tab[ ], int len_tab)
{
  for(int i = (len_tab-1-1)/2; i>=0; --i)
    heap_fix_down(tab, i, len_tab);
}



void heapSort(int t[ ], int n)
{
        heap_create_dn(t, n);

        for(int i=n-1;i>0; i--) {
                swap(t[0],t[i]);      //zamiana
                heap_fix_down(t,0,i); //naprawa
        }
}
```

SDIZO-SORTOWANIE STOGOWE

Tablica do posortowania	44	55	12	42	94	18	16	67

Po utworzeniu kopca	94	67	18	44	55	12	16	42

Po zamianie	42	67	18	44	55	12	16	94
Po naprawie	67	55	18	44	42	12	16	94
Po zamianie	16	55	18	44	42	12	67	94
Po naprawie	55	44	18	16	42	12	67	94
Po zamianie	12	44	18	16	42	55	67	94
Po naprawie	44	42	18	16	12	55	67	94
Po zamianie	12	42	18	16	44	55	67	94
Po naprawie	42	16	18	12	44	55	67	94
Po zamianie	12	16	18	42	44	55	67	94
Po naprawie	18	16	12	42	44	55	67	94
Po zamianie	12	16	18	42	44	55	67	94
Po naprawie	16	12	18	42	44	55	67	94
Po zamianie	12	16	18	42	44	55	67	94

SDIZO-SORTOWANIE INSERTION

17	15	29	16	17

17	15	29	16	17
17	15	29	16	17
15	17	29	16	17
15	17	29	16	17
15	17	29	16	17
15	17	16	29	17
15	16	17	29	17
15	16	17	29	17
15	16	17	17	29

15	16	17	17	29

SDIZO-SORTOWANIE INSERTION

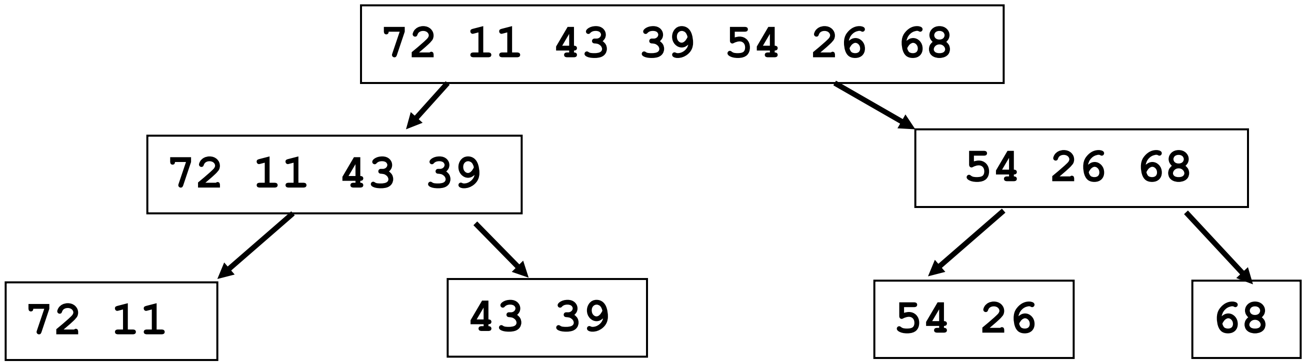
```
void insertionSort(int t[], int n)
{
        int key;
        for(int i=1;i<n;i++){
                key=t[i];
                int j=i;
                while(j>0 && t[j-1]>key){
                        t[j]=t[j-1];
                        j--;

                }
                t[j] = key;
        }
}
```

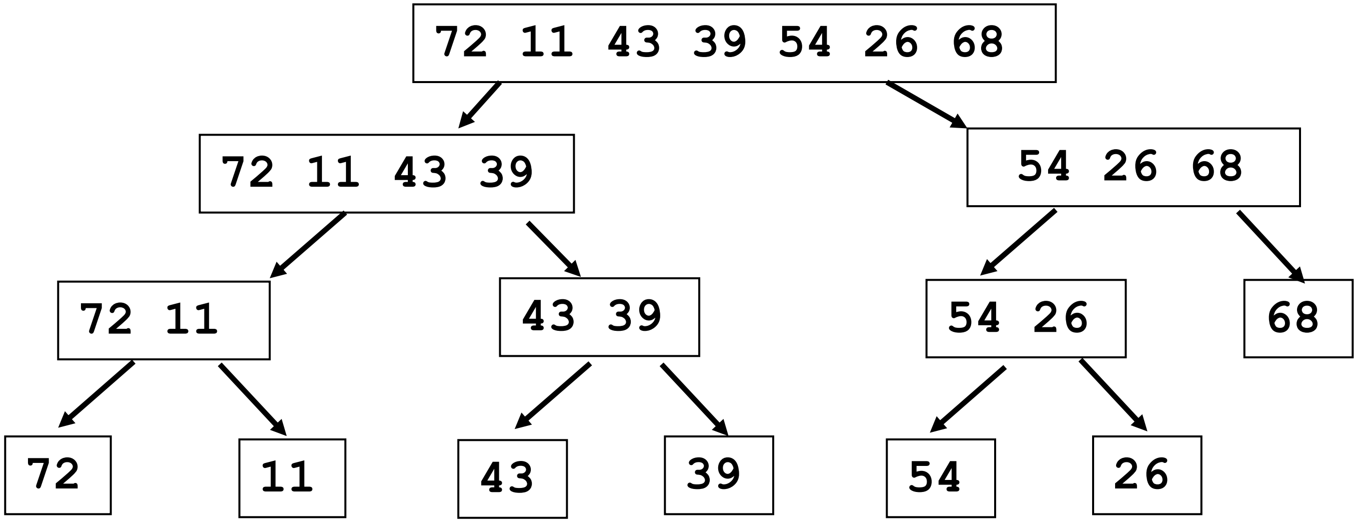
SDIZO-SORTOWANIE MERGE - IDEA

```
        72 11 43 39 54 26 68
         ↙               ↘
72 11 43 39             54 26 68
```

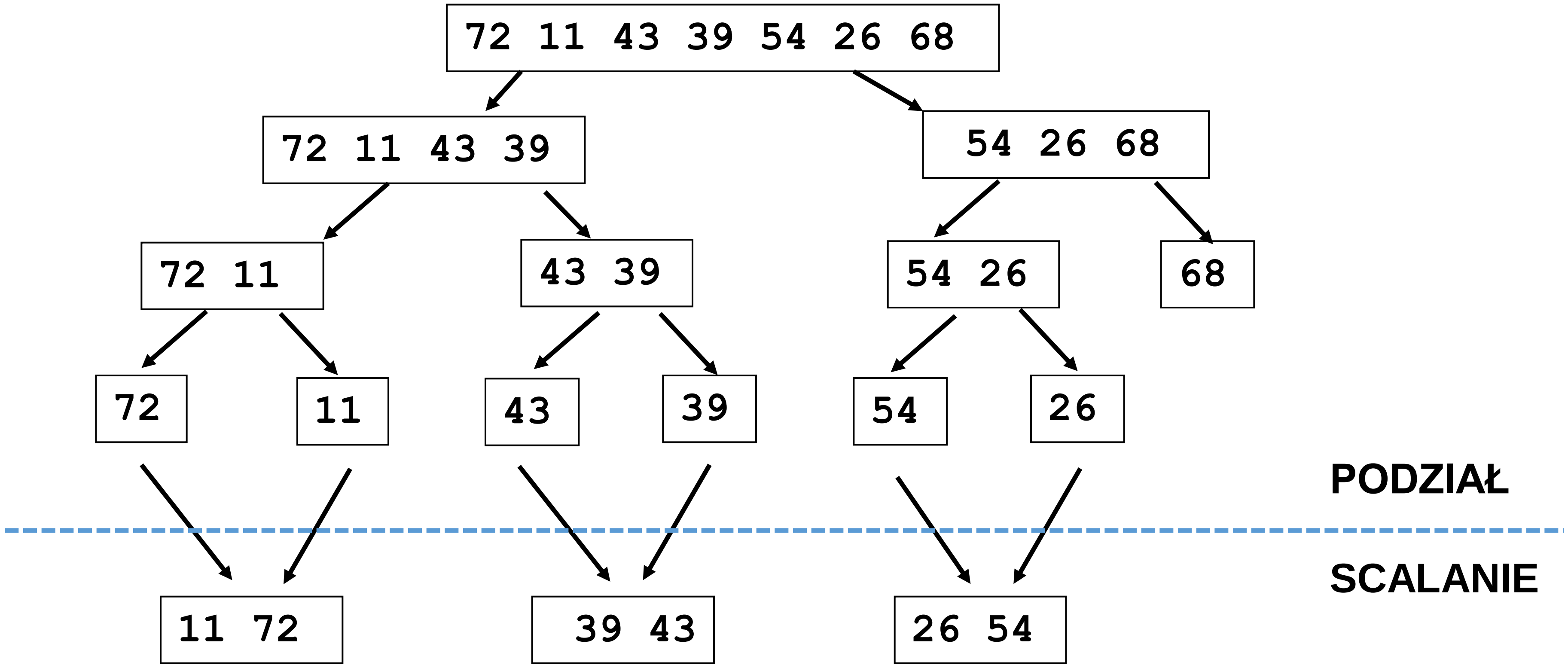
SDIZO-SORTOWANIE MERGE - IDEA



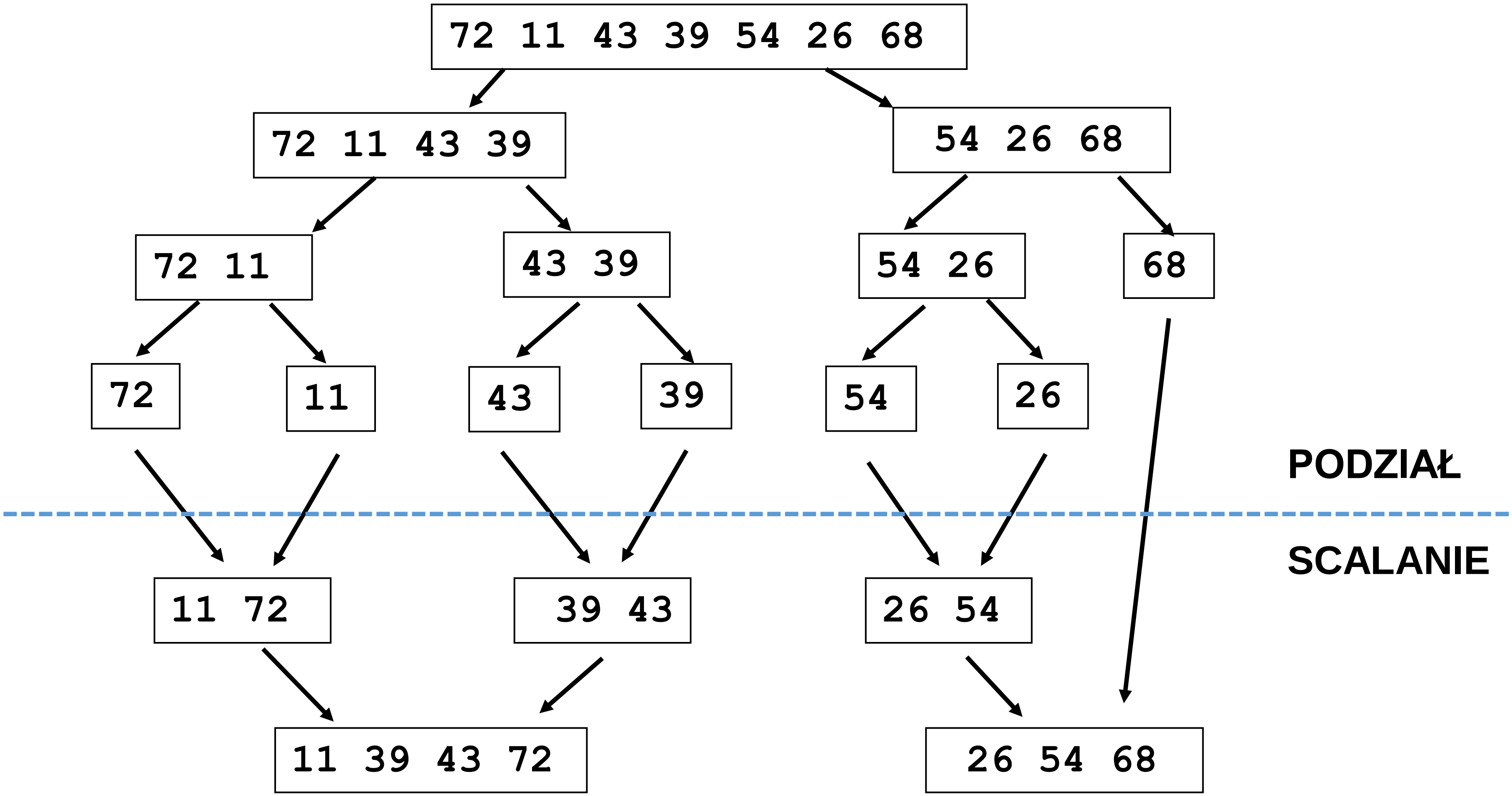
SDIZO-SORTOWANIE MERGE - IDEA



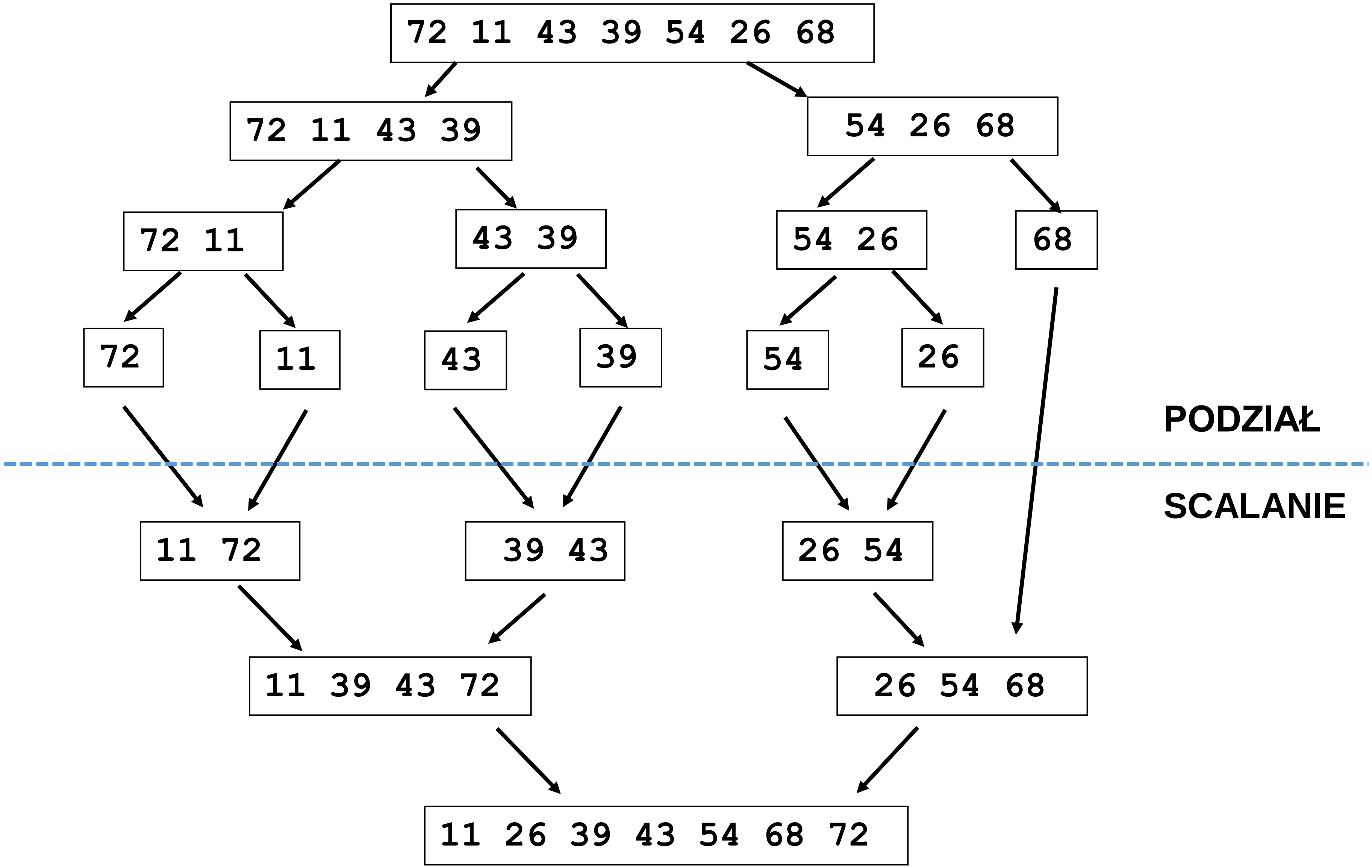
SDIZO-SORTOWANIE MERGE - IDEA



SDIZO-SORTOWANIE MERGE - IDEA



SDIZO-SORTOWANIE MERGE - IDEA



SDIZO-SORTOWANIE MERGE - SCALANIE

11 39 43 72	26 54 68	11
11 39 43 72	26 54 68	11 26
11 39 43 72	26 54 68	11 26 39
11 39 43 72	26 54 68	11 26 39 43
11 39 43 72	26 54 68	11 26 39 43 54
11 39 43 72	26 54 68	11 26 39 43 54 68
11 39 43 72	26 54 68	11 26 39 43 54 68 72

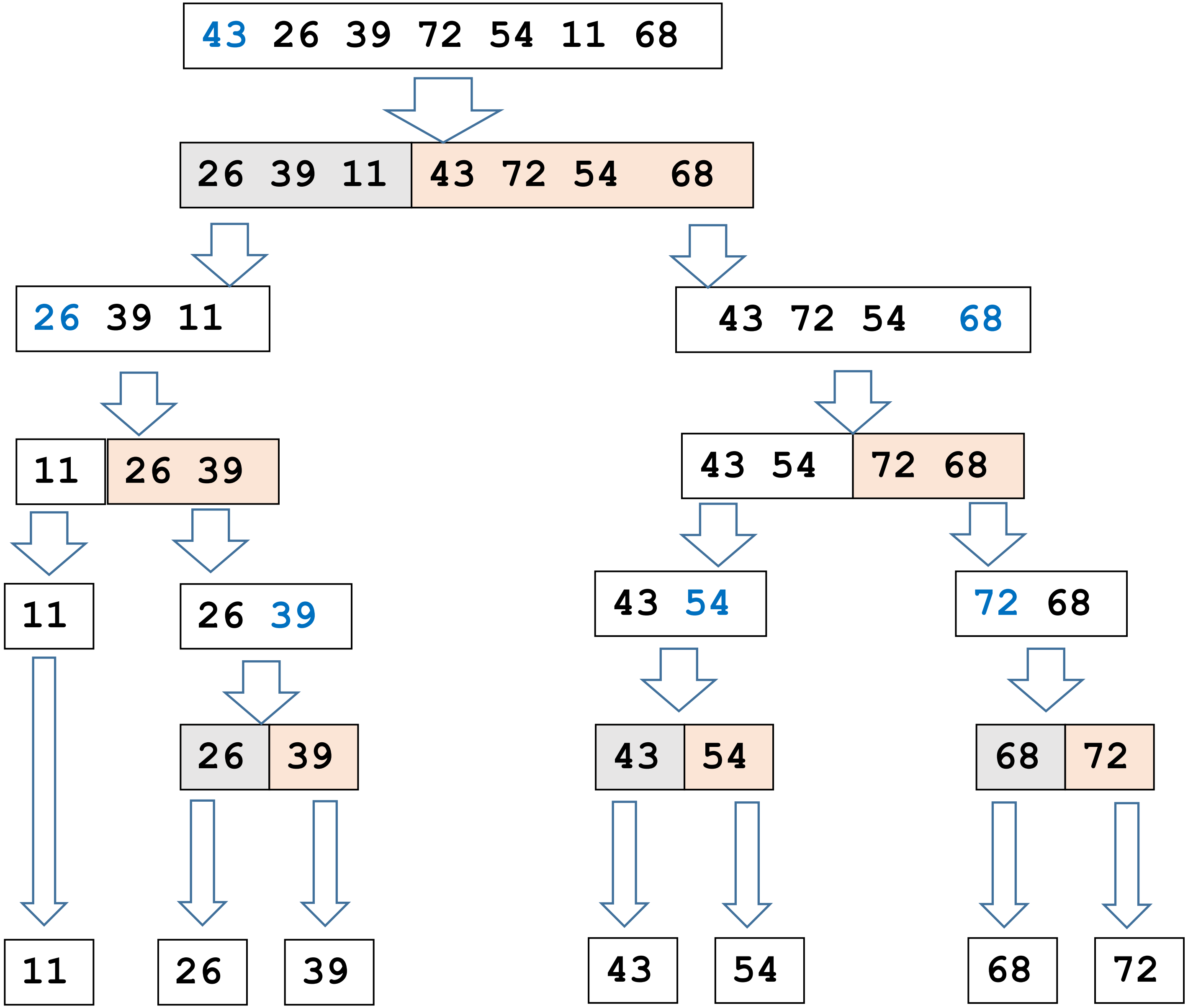
SDIZO-SORTOWANIE MERGE

```
void mergeSort(int tab[], int left, int right)
{
    if(left < right)
   {
      int m = (left+right)/2;
      mergeSort(tab,left,m);
      mergeSort(tab,m+1, right);
      mergeTab(tab,left,m,right);  //łączenie
    }
}
```

wywołanie funkcji sortującej:

```
mergeSort(A, 0, n-1);
```

SDIZO-SORTOWANIE QUICK



SDIZO-SORTOWANIE QUICK

```
void quickSort(int tab[], int l, int p)
{
      if(l>=p)          return;
      int m = partition(tab, l, p);
      quickSort(tab, l, m);
      quickSort(tab, m+1, p);
}
int partition(int tab[], int left, int right)
{     int pivot = tab[left];
      int l=left; int r = right;
      while(1) {
            while(tab[l]<pivot)   ++l;
            while(tab[r]>pivot)  --r;
            if(l < r) {
                  swap(tab[l],tab[r]);
                  ++l;
                  --r;
            }
            else {
               if (r == right) r--;
               return r;
            }
}
```

Sortowanie co n
(przez wstawianie)

32	17	84	21	56	42	86	19	7	41	25	51

32	17	84	21	56	42	86	19	7	41	25	51	**32,42,25**
32	17	84	21	56	42	86	19	7	41	25	51	**17,86,51**
32	17	84	21	56	42	86	19	7	41	25	51	**84, 19**
32	17	84	21	56	42	86	19	7	41	25	51	**21, 7**
32	17	84	21	56	42	86	19	7	41	25	51	**56, 41**

SDIZO-SORTOWANIE SHELLA

32	17	84	21	56	42	86	19	7	41	25	51

Sortowanie co 5

32	17	84	21	56	42	86	19	7	41	25	51
25	17	84	21	56	**32**	86	19	7	41	**42**	51
25	17	84	21	56	32	86	19	7	41	42	51
25	**17**	84	21	56	32	**51**	19	7	41	42	**86**
25	17	84	21	56	32	51	19	7	41	42	86
25	17	**19**	21	56	32	51	**84**	7	41	42	86
25	17	19	21	56	32	51	84	7	41	42	86
25	17	19	**7**	56	32	51	84	**21**	41	42	86
25	17	19	7	56	32	51	84	21	41	42	86
25	17	19	7	**41**	32	51	84	21	**56**	42	86

Sortowanie co 3

25	17	19	7	41	32	51	84	21	56	42	86

25	17	19	7	41	32	51	84	21	56	42	86

7	17	19	**25**	41	32	**51**	84	21	**56**	42	86

7	17	19	25	41	32	51	84	21	56	42	86

7	**17**	19	25	**41**	32	51	**42**	21	56	**84**	86

7	17	19	25	41	32	51	42	21	56	84	86

7	17	**19**	25	41	**21**	51	42	**32**	56	84	**86**

Sortowanie co 1

7	17	19	25	41	21	51	42	32	56	84	86

7	17	19	25	41	21	51	42	32	56	84	86
7	17	19	21	25	41	51	42	32	56	84	86

7	17	19	21	25	41	51	42	32	56	84	86
7	17	19	21	25	41	42	51	32	56	84	86

7	17	19	21	25	41	42	51	32	56	84	86
7	**17**	**19**	**21**	**25**	**32**	**41**	**42**	**51**	**56**	**84**	**86**

Posortowane !!!

SDIZO-SORTOWANIE SHELLA

Wyraz ogólny ciągu ($k \geqslant 1$)	Konkretne odstępy	Rząd złożoności pesymistycznej	Autor i rok publikacji
$\lfloor N/2^k \rfloor$	$\lfloor \frac{N}{2} \rfloor, \lfloor \frac{N}{4} \rfloor, \ldots, 1$	$\Theta(N^2)$ [gdy $N = 2^p$]	Shell, 1959
$2\lfloor N/2^{k+1} \rfloor + 1$	$2\lfloor \frac{N}{4} \rfloor + 1, \ldots, 3, 1$	$\Theta(N^{3/2})$	Frank, Lazarus, 1960
$2^k - 1$	$1, 3, 7, 15, 31, 63, \ldots$	$\Theta(N^{3/2})$	Hibbard, 1961
$2^k + 1$, na początku 1	$1, 3, 5, 9, 17, 33, 65, \ldots$	$\Theta(N^{3/2})$	Papiernow, Stasiewicz, 1965
kolejne liczby postaci $2^p 3^q$	$1, 2, 3, 4, 6, 8, 9, 12, \ldots$	$\Theta(N \log^2 N)$	Pratt, 1971
$(3^k - 1)/2$, nie większe niż $\lceil N/3 \rceil$	$1, 4, 13, 40, 121, \ldots$	$\Theta(N^{3/2})$	Knuth, 1973
$4^k + 3 \cdot 2^{k-1} + 1$, na początku 1	$1, 8, 23, 77, 281, \ldots$	$O(N^{4/3})$	Sedgewick, 1982
$\prod\limits_{\substack{0 \leqslant q < r \\ q \neq (r^2+r)/2-k}} a_q$, gdzie $r = \lfloor \sqrt{2k + \sqrt{2k}} \rfloor$, $a_q = \min\{n \in \mathbb{N} : n \geqslant (5/2)^{q+1}, \forall p : 0 \leqslant p < q \Rightarrow \mathrm{nwd}(a_p, n) = 1\}$	$1, 3, 7, 21, 48, 112, \ldots$	$O\left(N e^{\sqrt{8 \ln(5/2) \ln N}}\right)$	Incerpi, Sedgewick, 1985
$9(4^{k-1} - 2^{k-1}) + 1,\ 4^{k+1} - 6 \cdot 2^k + 1$	$1, 5, 19, 41, 109, \ldots$	$O(N^{4/3})$	Sedgewick, 1986
$h_k = \max\left\{\lfloor 5h_{k-1}/11 \rfloor, 1\right\},\ h_0 = N$	$\lfloor \frac{5N}{11} \rfloor, \lfloor \frac{5}{11} \lfloor \frac{5N}{11} \rfloor \rfloor, \ldots, 1$	?	Gonnet, Baeza-Yates, 1991
$\left\lceil \frac{9^k - 4^k}{5 \cdot 4^{k-1}} \right\rceil$	$1, 4, 9, 20, 46, 103, \ldots$	?	Tokuda, 1992
nieznany	$1, 4, 10, 23, 57, 132, \ldots$	?	Ciura, 2001

SORTOWANIE PRZEZ ZLICZANIE (COUNTING SORT)

Zakładamy, że każdy z n sortowanych elementów jest liczbą z przedziału 1 do k (dla ustalonego k)
Idea – wyznaczenie dla każdej sortowanej liczby x ilości elementów mniejszych od x ->daje to pozycję w ciągu posortowanym

Dane: A-tablica z liczbami do posortowania, B-tablica posortowana, C-tablica pomocnicza

Kroki:

1. **Wyzeruj C**
2. **Uzupełnij C tak, że C[A[i]] zwiera ilość liczb mniejszych od A[i]**
3. **Na podstawie A i C wypełnij tablicę B**
   B[C[A[i]]] := A[i]

Złożoność – jeśli k=O(n) to sortowanie O(n)
Sortowanie jest stabilne tzn. liczby o tych samych wartościach występują w tablicy wynikowej w takiej samej kolejności jak w tablicy wejściowej

SORTOWANIE PRZEZ ZLICZANIE (COUNTING SORT)

```
COUNTING_SORT(A,B,k)
    for i:=1 to k do
        C[ i ] := 0
    for i:=1 to length(A) do
        C[A[i]] := C[A[i]]+1
W tym miejscu C[i] zawiera liczbę elementów równych i
    for i:=2 to k do
        C[i]:=C[i]+C[i-1]
W tym miejscu C[i] zawiera liczbę elementów <=i
    for i:=length(A) downto 1 do
        B[ C[A[i]]]:=A[i]
        C[A[i]] := C[A[i]] - 1
```

SORTOWANIE PRZEZ ZLICZANIE (COUNTING SORT)

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

A – tablica z danymi

	1	2	3	4	5	6
C	2	0	2	3	0	1

C[i] = i

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

C[i] zawiera liczbę elementów <= i

Wypełnianie tablicy B

	1	2	3	4	5	6	7	8
B							4	

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

pierwsza iteracja

	1	2	3	4	5	6	7	8
B		1					4	

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

druga iteracja

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

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

trzecia iteracja

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

<- posortowane

SORTOWANIE POZYCYJNE (RADIX-SORT)

Sortowanie pozycyjne sortuje po cyfrach sortowanych liczb
Intuicja podpowiada zaczynanie sortowania od najbardziej znaczącej cyfr, ale w rzeczywistości zaczynamy od najmniej znaczących.
Niech d oznacza ilość cyfr, a A tablicę z liczbami do posortowania
Algorytm jest następujący:

```
RADIX-SORT (A,d)
1      for i:=1 to d  do
2        posortuj stabilnie tablicę A względem cyfry i
```

Złożoność – zależy od złożoności 2 linii. Jeżeli zastosujemy w tej linii sortowanie przez zliczanie to złożoność będzie O(d*n) czyli O(n)

SORTOWANIE POZYCYJNE (RADIX-SORT)

Przykład:

329		720		720		329
457		355		329		355
657		436		436		436
839	⇒	457	⇒	839	⇒	457
436		657		355		657
720		329		457		720
355		839		657		839
		↑		↑		↑

SORTOWANIE KUBEŁKOWE(BUCKET-SORT)

Idea:

1.Podziel zadany przedział liczb na k podprzedziałów (kubełków) o równej długości.

2.Przypisz liczby z sortowanej tablicy do odpowiednich kubełków.

3.Sortuj liczby w niepustych kubełkach (przez wstawianie – insertion sort).

4.Wypisz po kolei zawartość niepustych kubełków.

Przy analizie zakładamy, że liczby do posortowania (w tabeli A) są z przedziału [0..1), B będzie tablicą list o indeksach 0..n-1

```
BUCKET-SORT
n:=length(A)
for i:=1 to n do
     wstaw A[ i ] do listy B[ ⌊ n*A[ i ] ⌋ +1]     -
for i:=1 to k do
        posortuj listę B[i] przez wstawianie
 połącz listy B[1], B[2],…B[k] dokładnie w tej samej kolejności
```

SORTOWANIE PRZEZ ZLICZANIE (COUNTING SORT)

Złożoność wynosi O(n) – krokiem algorytmu decydującym o złożoności są kroki 5 i 6. Można udowodnić, że złożoność tego fragmentu algorytmu wynosi O(n) więc złożoność całego O(n)