Wykład 2

1. Algorytmy liniowe c.d. - Sortowanie pozycyjne
2. Badanie algorytmów
3. Algorytmy rekurencyjne „dziel i zwyciężaj” - Sortowanie przez łączenie

1. Algorytmy liniowe c.d. - Sortowanie pozycyjne

Zadanie: Uporządkować rosnąco *N* liczb *D* cyfrowych. Proces sortowania należy wykonać przez kolejne porządkowanie liczb ustalając porządek według kolejnych cyfr pozycyjnych, zaczynając od najmniej znaczącej cyfry. Algorytm sortowania powinien być stabilny. Zakłada się, że liczby zawierają jednakową liczbę cyfr.

Algorytm sortowania - poziom konceptualny:
(1) *wskaż na najmniej znaczącą cyfrę pierwszej liczby;*
(2) *wykonaj, co następuje, D razy:*
  (2.1) *wykonaj Sortowanie_przez_zliczanie N liczb według wskazanej cyfry znaczącej;*
  (2.2) *wskaż na następną cyfrę znaczącą pierwszej liczby.*

Przykład 1

Kowalski	1963	02	09
Nowak	1964	05	07
Adamski	1966	05	07
Matelski	1968	03	09
Kowal	1964	03	06
Kunicki	1967	02	01
Kołodziej	1963	05	05

D = 1				D = 2				D = 3			
Kunicki	1967	02	*01*	Kunicki	1967	*02*	01	Kowalski	*1963*	02	09
Kołodziej	1963	05	*05*	Kowalski	1963	*02*	09	Kołodziej	*1963*	05	05
Kowal	1964	03	*06*	Kowal	1964	*03*	06	Kowal	*1964*	03	06
Nowak	1964	05	*07*	Matelski	1968	*03*	09	Nowak	*1964*	05	07
Adamski	1966	05	*07*	Kołodziej	1963	*05*	05	Adamski	*1966*	05	07
Kowalski	1963	02	*09*	Nowak	1964	*05*	07	Kunicki	*1967*	02	01
Matelski	1968	03	*09*	Adamski	1966	*05*	07	Matelski	*1968*	03	09

Algorytm sortowania pozycyjnego - poziom projektowy

(1) *x ← 1;*
(2) *dopóki x <= D, wykonuj, co następuje:*
(2.1) *i ← 0;*
(2.2) *dopóki i <= K, wykonuj co następuje:*
{zerowanie liczników , czyli elementów w tablicy *Liczniki*}
(2.2.1) *Liczniki(i) ← 0;*
(2.2.2) *i ← i+1;*
(2.3) *j ← 1;*
(2.4) *dopóki j <= N, wykonuj, co następuje:*
{zliczanie elementów z tablicy *We(N)* o wartości *j* w tablicy *Liczniki*}
(2.4.1) *y ← We(j, x);*
(2.4.2) *Liczniki(y) ← Liczniki(y) + 1;*
(2.4.3) *j ← j + 1;*
(2.5) *i ← 2;*
(2.6) *dopóki i <= K, wykonuj, co następuje:*
{zliczanie elementów mniejszych / równych *i* umieszczonych w tablicy *We(x)*}
(2.6.1) *Liczniki (i) = Liczniki(i) + Liczniki(i-1);*
(2.6.2) *i ← i +1;*
(2.7) *i ← N;*
(2.8) *dopóki i >= 1, wykonuj, co następuje:*
{Ustawianie elementów w tablicy *Wy* w porządku rosnącym według cyfr znaczących na pozycji *x*, zgodnie z zawartością elementów tablicy *Liczniki*}
(2.8.1) *j ← We(i, x);*
(2.8.2) *Wy(Liczniki(j)) ← We(i);*
(2.8.3) *Liczniki(j) ← Liczniki(j) -1;*
(2.8.4) *i ← i - 1;*
(2.9) *x ← x +1;*

Cechy algorytmu:

1. Zależność liniowa czasu wykonania od liczby elementów.
2. Stabilność: elementy o tych samych wartościach występują w tablicy wynikowej w takiej samej kolejności jak w tablicy początkowej

Przykład 2

Przykład dwuwymiarowej *7 x 3* tablicy *We*

N/D	We			D = 1			D = 2			D = 3		
1	3	2	9	7	2	*0*	7	*2*	0	*3*	2	9
2	4	5	7	3	5	*5*	3	*2*	9	*3*	5	5
3	6	5	7	4	3	*6*	4	*3*	6	*4*	3	6
4	8	3	9	4	5	*7*	8	*3*	9	*4*	5	7
5	4	3	6	6	5	*7*	3	*5*	5	*6*	5	7
6	7	2	0	3	2	*9*	4	*5*	7	*7*	2	0
7	3	5	5	8	3	*9*	6	*5*	7	*8*	3	9

N	*1*	*2*	*3*	*4*	*5*	*6*	*7*
LiczbyWe	**3 2 9**	**4 5 7**	**6 5 7**	**8 3 9**	**4 3 6**	**7 2 0**	**3 5 5**
		3 5 *5*					
D = 1	**7 2 *0***	**3 5 *5***					
	7 2 *0*	**3 5 *5***	**4 3 *6***				
	7 2 *0*	**3 5 *5***	**4 3 *6***				**8 3 *9***
	7 2 *0*	**3 5 *5***	**4 3 *6***		**6 5 *7***		**8 3 *9***
	7 2 *0*	**3 5 *5***	**4 3 *6***	**4 5 *7***	**6 5 *7***		**8 3 *9***
	7 2 *0*	**3 5 *5***	**4 3 *6***	**4 5 *7***	**6 5 *7***	**3 2 *9***	**8 3 *9***
				8 *3* 9			
D = 2		**3 *2* 9**		**8 *3* 9**			
		3 *2* 9		**8 *3* 9**			**6 *5* 7**
		3 *2* 9		**8 *3* 9**		**4 *5* 7**	**6 *5* 7**
		3 *2* 9	**4 *3* 6**	**8 *3* 9**		**4 *5* 7**	**6 *5* 7**
		3 *2* 9	**4 *3* 6**	**8 *3* 9**	**3 *5* 5**	**4 *5* 7**	**6 *5* 7**
	7 *2* 0	**3 *2* 9**	**4 *3* 6**	**8 *3* 9**	**3 *5* 5**	**4 *5* 7**	**6 *5* 7**
					***6* 5 7**		
D = 3				***4* 5 7**	***6* 5 7**		
		***3* 5 5**		***4* 5 7**	***6* 5 7**		
		***3* 5 5**		***4* 5 7**	***6* 5 7**		***8* 3 9**
		***3* 5 5**	***4* 3 6**	***4* 5 7**	***6* 5 7**		***8* 3 9**
	***3* 2 9**	***3* 5 5**	***4* 3 6**	***4* 5 7**	***6* 5 7**		***8* 3 9**
	***3* 2 9**	***3* 5 5**	***4* 3 6**	***4* 5 7**	***6* 5 7**	*7* **2 0**	***8* 3 9**
LiczbyWy	**3 2 9**	**3 5 5**	**4 3 6**	**4 5 7**	**6 5 7**	**7 2 0**	**8 3 9**

Tab.2. Przebieg sortowania pozycyjnego

```
#pragma hdrstop
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <mem.h>

const int D = 3;
const int K = 10;
const int m=20;
const long N=7L;

void sortpoz(int T[], long l, long p, char We[][D] );
void sortlicz(char We[][D],char Wy[][D], int d, long l, long p);
void utworz_dane_do_sortowania_pozycyjnego(int T[], char We[][D],long l,long
p);
void przywroc_dane_po_sortowaniu_pozycyjnym(int T[],char We[][D],long l,
                                                          long p);
void wypelnij(int T[], long& ile);
void wyswietl(int T[], long ile);

int T[N]={329,457,657,839,436,720,355};
long ile=N;
char We[N][D];
void main(int argc, char* argv[])
{
  //wypelnij(T,ile);
  sortpoz(T,0,ile-1,We);
  wyswietl(T,ile);
  getch();
}
//---------------------------------------------------------------------------
 void wypelnij(int T[], long& ile)
    {   srand(3);
         for (long i=0; i<N; i++)
               T[i]=rand();
         ile=N;
    }
void wyswietl(int T[], long ile)
 {
   for (long i=0; i<ile; i++)
          { printf("%d \n", T[i]);
            if (i%m==0)
              {  char z=getch();
                 if (z=='k') return; }
          }
```

```
}
```

```
void utworz_dane_do_sortowania_pozycyjnego(int T[], char We[][D], long l,
                                                                long p)
 {
  for (long i=l; i<=p; i++)
   for (int j=D-1;j>=0;j--)
    {
     We[i][j]=T[i]%10 + 48; // T1[i]%10-wartosc cyfry pozycyjnej w systemie 10-ym,
                            // po dodaniu 48 otrzymujemy kod ASCII tej cyfry (48-kod ASCII 0)
     T[i] = T[i]/10;        //kolejna starsza pozycja w systemie dziesiętnym
    }
 }

 void przywroc_dane_po_sortowaniu_pozycyjnym(int T[], char We[][D], long l,
                                                                long p)
 { int x;
  for(long i=l; i<=p; i++)
   {
     T[i]=0;
     x=1;
    for (int j=D-1; j>=0;j--)
     {
       T[i]=T[i]+(We[i][j]-48)*x;
       x=x*10;
     }
   }
 }
void sortlicz(char We[][D], char Wy[][D], int d, long l, long p)
 { int i, j, y;
   int Cyfry[K];
                        //zerowanie liczników
   for(int i= 0; i< K;i++ )
            Cyfry[i] = 0;
          //zliczanie jedynek i zer z pozycji znaczącej d+1 elementów tablicy we w tablicy Cyfry, czyli cyfry z pozycji x
   for (int i = l;i<=p;i++)
            Cyfry[We[i][d] - 48] += 1;
          //zliczanie elementów mniejszych oraz równych "i" w tablicy Cyfry
   for (int i = 1;i<K;i++)
       Cyfry[i] += Cyfry[i-1];
   for ( int i = p; i>=l;i--)
                        //sortowanie elementów z tablicy we: (d+1)-cyfra znacząca elementu we[i]
                        //jest indeksem elementu tablicy Cyfry, zawierającego indeks id elementu
                       //id=Cyfry[j]-1) w tablicy wy (wy[id]), w którym powinien być wstawiony element we[i].
    {  j =We[i][d]-48;
       memmove(Wy[Cyfry[j]-1], We[i], D*sizeof(char));
       Cyfry[j] -=1;
    }
 }
```

```
//########sortowanie pozycyjne#######################
 void sortpoz(int T[], long l, long p, char We[][D])
  {
    char Wy[N][D];
    utworz_dane_do_sortowania_pozycyjnego(T, We, l, p);
    memmove(Wy, We, (p-l+1)*D*sizeof(char));
    for (int j=D-1; j>=0;j--)
       if (j%2 == 0)                    //zamiana tablic po kolejnym sortowaniu przez zliczanie
         sortlicz(Wy, We, j, l, p);     //tablica wyjściowa w jednym przebiegu
       else                             //jest tablicą wejściową w następnym przebiegu
         sortlicz(We, Wy, j, l, p);     //a tablica wejściowa staje się tablicą wyjściową
    przywroc_dane_po_sortowaniu_pozycyjnym(T, We, l, p);
   }
 //w ostatnim przebiegu dla największej cyfry znaczącej D, gdy j=0 tablicą wyjściową powinna być tablica We
```

Uzupełnienie wykładu - Sortowanie pozycyjne w systemie dwójkowym

		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	∑ 0	∑ 1	Pozycja w ciągu wyjściowym
1	**329**	0	0	0	0	0	0	0	1	0	1	0	0	1	0	0	***1***	**3**	**4**	**4**
2	**457**	0	0	0	0	0	0	0	1	1	1	0	0	1	0	0	***0***			**1**
3	**657**	0	0	0	0	0	0	1	0	1	0	0	1	0	0	0	***1***			**5**
4	**839**	0	0	0	0	0	0	1	1	0	1	0	0	0	1	1	***1***			**6**
5	**436**	0	0	0	0	0	0	0	1	1	0	1	1	0	1	0	***0***			**2**
6	**720**	0	0	0	0	0	0	1	0	1	1	0	1	0	0	0	***0***			**3**
7	**355**	0	0	0	0	0	0	0	1	0	1	1	0	0	0	1	***1***			**7**
1	**457**	0	0	0	0	0	0	0	1	1	1	0	0	1	0	***0***	0	**5**	**2**	**1**
2	**436**	0	0	0	0	0	0	0	1	1	0	1	1	0	1	***0***	0			**2**
3	**720**	0	0	0	0	0	0	1	0	1	1	0	1	0	0	***0***	0			**3**
4	**329**	0	0	0	0	0	0	0	1	0	1	0	0	1	0	***0***	1			**4**
5	**657**	0	0	0	0	0	0	1	0	1	0	0	1	0	0	***0***	1			**5**
6	**839**	0	0	0	0	0	0	1	1	0	1	0	0	0	1	***1***	1			**6**
7	**355**	0	0	0	0	0	0	0	1	0	1	1	0	0	0	***1***	1			**7**
1	**457**	0	0	0	0	0	0	0	1	1	1	0	0	1	***0***	0	0	**5**	**2**	**1**
2	**436**	0	0	0	0	0	0	0	1	1	0	1	1	0	***1***	0	0			**6**
3	**720**	0	0	0	0	0	0	1	0	1	1	0	1	0	***0***	0	0			**2**
4	**329**	0	0	0	0	0	0	0	1	0	1	0	0	1	***0***	0	1			**3**
5	**657**	0	0	0	0	0	0	1	0	1	0	0	1	0	***0***	0	1			**4**
6	**839**	0	0	0	0	0	0	1	1	0	1	0	0	0	***1***	1	1			**7**
7	**355**	0	0	0	0	0	0	0	1	0	1	1	0	0	***0***	1	1			**5**
1	**457**	0	0	0	0	0	0	0	1	1	1	0	0	***1***	0	0	0	**5**	**2**	**6**
2	**720**	0	0	0	0	0	0	1	0	1	1	0	1	***0***	0	0	0			**1**
3	**329**	0	0	0	0	0	0	0	1	0	1	0	0	***1***	0	0	1			**7**
4	**657**	0	0	0	0	0	0	1	0	1	0	0	1	***0***	0	0	1			**2**
5	**355**	0	0	0	0	0	0	0	1	0	1	1	0	***0***	0	1	1			**3**
6	**436**	0	0	0	0	0	0	0	1	1	0	1	1	***0***	1	0	0			**4**
7	**839**	0	0	0	0	0	0	1	1	0	1	0	0	***0***	1	1	1			**5**
1	**720**	0	0	0	0	0	0	1	0	1	1	0	***1***	0	0	0	0	**4**	**3**	**5**
2	**657**	0	0	0	0	0	0	1	0	1	0	0	***1***	0	0	0	1			**6**
3	**355**	0	0	0	0	0	0	0	1	0	1	1	***0***	0	0	1	1			**1**
4	**436**	0	0	0	0	0	0	0	1	1	0	1	***1***	0	1	0	0			**7**
5	**839**	0	0	0	0	0	0	1	1	0	1	0	***0***	0	1	1	1			**2**
6	**457**	0	0	0	0	0	0	0	1	1	1	0	***0***	1	0	0	0			**3**
7	**329**	0	0	0	0	0	0	0	1	0	1	0	***0***	1	0	0	1			**4**
1	**355**	0	0	0	0	0	0	0	1	0	1	***1***	0	0	0	1	1	**5**	**2**	**6**

2	**839**	0	0	0	0	0	0	1	1	0	1	***0***	0	0	1	1	1			**1**
3	**457**	0	0	0	0	0	0	0	1	1	1	***0***	0	1	0	0	0			**2**
4	**329**	0	0	0	0	0	0	0	1	0	1	***0***	0	1	0	0	1			**3**
5	**720**	0	0	0	0	0	0	1	0	1	1	***0***	1	0	0	0	0			**4**
6	**657**	0	0	0	0	0	0	1	0	1	0	***0***	1	0	0	0	1			**5**
7	**436**	0	0	0	0	0	0	0	1	1	0	***1***	1	0	1	0	0			**7**
1	**839**	0	0	0	0	0	0	1	1	0	***1***	0	0	0	1	1	1	**2**	**5**	**3**
2	**457**	0	0	0	0	0	0	0	1	1	***1***	0	0	1	0	0	0			**4**
3	**329**	0	0	0	0	0	0	0	1	0	***1***	0	0	1	0	0	1			**5**
4	**720**	0	0	0	0	0	0	1	0	1	***1***	0	1	0	0	0	0			**6**
5	**657**	0	0	0	0	0	0	1	0	1	***0***	0	1	0	0	0	1			**1**
6	**355**	0	0	0	0	0	0	0	1	0	***1***	1	0	0	0	1	1			**7**
7	**436**	0	0	0	0	0	0	0	1	1	***0***	1	1	0	1	0	0			**2**
1	**657**	0	0	0	0	0	0	1	0	***1***	0	0	1	0	0	0	1	**3**	**4**	**4**
2	**436**	0	0	0	0	0	0	0	1	***1***	0	1	1	0	1	0	0			**5**
3	**839**	0	0	0	0	0	0	1	1	***0***	1	0	0	0	1	1	1			**1**
4	**457**	0	0	0	0	0	0	0	1	***1***	1	0	0	1	0	0	0			**6**
5	**329**	0	0	0	0	0	0	0	1	***0***	1	0	0	1	0	0	1			**2**
6	**720**	0	0	0	0	0	0	1	0	***1***	1	0	1	0	0	0	0			**7**
7	**355**	0	0	0	0	0	0	0	1	***0***	1	1	0	0	0	1	1			**3**
1	**839**	0	0	0	0	0	0	1	***1***	0	1	0	0	0	1	1	1	**2**	**5**	**3**
2	**329**	0	0	0	0	0	0	0	***1***	0	1	0	0	1	0	0	1			**4**
3	**355**	0	0	0	0	0	0	0	***1***	0	1	1	0	0	0	1	1			**5**
4	**657**	0	0	0	0	0	0	1	***0***	1	0	0	1	0	0	0	1			**1**
5	**436**	0	0	0	0	0	0	0	***1***	1	0	1	1	0	1	0	0			**6**
6	**457**	0	0	0	0	0	0	0	***1***	1	1	0	0	1	0	0	0			**7**
7	**720**	0	0	0	0	0	0	1	***0***	1	1	0	1	0	0	0	0			**2**
1	**657**	0	0	0	0	0	0	***1***	0	1	0	0	1	0	0	0	1	**4**	**3**	**5**
2	**720**	0	0	0	0	0	0	***1***	0	1	1	0	1	0	0	0	0			**6**
3	**839**	0	0	0	0	0	0	***1***	1	0	1	0	0	0	1	1	1			**7**
4	**329**	0	0	0	0	0	0	***0***	1	0	1	0	0	1	0	0	1			**1**
5	**355**	0	0	0	0	0	0	***0***	1	0	1	1	0	0	0	1	1			**2**
6	**436**	0	0	0	0	0	0	***0***	1	1	0	1	1	0	1	0	0			**3**
7	**457**	0	0	0	0	0	0	***0***	1	1	1	0	0	1	0	0	0			**4**
1	**329**	0	0	0	0	0	0	0	1	0	1	0	0	1	0	0	1			**1**
2	**355**	0	0	0	0	0	0	0	1	0	1	1	0	0	0	1	1			**2**
3	**436**	0	0	0	0	0	0	0	1	1	0	1	1	0	1	0	0			**3**
4	**457**	0	0	0	0	0	0	0	1	1	1	0	0	1	0	0	0			**4**
5	**657**	0	0	0	0	0	0	1	0	1	0	0	1	0	0	0	1			**5**
6	**720**	0	0	0	0	0	0	1	0	1	1	0	1	0	0	0	0			**6**
7	**839**	0	0	0	0	0	0	1	1	0	1	0	0	0	1	1	1			**7**

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <mem.h>

const int D = 16;
const int K = 2;
const int m=20;
const long N=7L;

inline int cyfra(int,int);
void sortlicz(int We[],int Wy[], int d, long l, long p);
void sortpoz(int T[], long l, long p);
void wypelnij(int T[], long& ile);
void wyswietl(int T[], long ile);
int T[N]={329,457,657,839,436,720,355};
long ile=N;

void main(int argc, char* argv[])
{
  //wypelnij(T, ile);
  sortpoz(T, 0, ile-1);
  wyswietl(T, ile);
  getch();
}
//---------------------------------------------------------------------------
 void wypelnij(int T[], long& ile)
    {   srand(3);
         for (long i=0; i<N; i++)
               T[i]=rand();
         ile=N;
    }
void wyswietl(int T[], long ile)
 { for (long i=0; i<ile; i++)
          { printf("%d \n", T[i]);
            if (i%m==0)
              {  char z=getch();
                 if (z=='k') return; }
          }
 }

 inline int cyfra(int a, int b)              //pobieranie bitów na pozycji b+1
   { return (a >> b)& 0x0001;   //tzn. przesuwanie w prawo o b bitów kolejnej danej
   }                            //gdy na pozycji b+1 jest bieżący bit
```

```
void sortlicz(int We[],int Wy[], int d, long l, long p)
 {              //linie oznaczone * zawieraja czynnosci pomocnicze i mozna je wylaczyc
   int i,j,y;
   int Cyfry[K];
            //zerowanie licznikow cyfr znaczących systemu dwojkowego
   for(int i= 0; i< K;i++ )
          Cyfry[i] = 0;
             //zliczanie jedynek i zer z pozycji znaczącej d+1 elementów tablicy we
   for (int i = l;i<=p;i++)                    //w tablicy Cyfry, czyli cyfry z pozycji x
       Cyfry[cyfra(We[i],d)] += 1;
                //zliczanie elementów mniejszych oraz rownych "i" w tablicy Cyfry
   for (int i = 1;i<K;i++)
      Cyfry[i] += Cyfry[i-1];
   for ( int i = p; i>=l;i--)
          //sortowanie elementow z tablicy we: (d+1)-cyfra znacząca elementu we[i]
          //jest indeksem elementu tablicy Cyfry, zawierajacego indeks id elementu
        //id=Cyfry[j]-1) w tablicy wy (wy[id]), w ktorym powinien byc wstawiony element we[i].
    {
       j =cyfra(We[i], d);
       Wy[Cyfry[j]-1]=We[i];
       Cyfry[j] -=1;
    }
 }
//######################sortowanie pozycyjne###############
 void sortpoz(int We[], long l, long p)
 {
   int Wy[N]; int D1=D-1;
   for (int j=0; j<=D1;j++)
     if (j%2 == 0)
          sortlicz(We,Wy,j,l,p);
     else
           sortlicz(Wy,We,j,l,p);
   if (D1%2 == 0)
     memmove(We,Wy,(p+1)*sizeof(int));
 }
```

2. Badanie algorytmów - analiza algorytmu sortowania pozycyjnego

Lp		Koszt	Liczba wykonań
1	**for** (**int** j=0; j<=D-1;i++)	c_1	*D+1 (D=liczba bitow))*
2	**if** (j%2 == 0)	c_2	*D*
	sortlicz(We, Wy,j,l,p);	c_3	*D*
	else sortlicz(Wy, We,j,l,p);		
	for(**int** i= 0; i< K;i++)	c_4	*D(K+1)*
3	Cyfry[i] = 0;	c_5	*DK*
4	**for** (**int** i = 0;i<=N-1;i++)	c_6	*D(N+1)*
5	Cyfry[cyfra(we[i], d)] += 1;	c_7	*DN*
6	**for** (**int** i = 1;i<K;i++)	c_8	*DK*
7	Cyfry[i] += Cyfry[i-1];	c_9	*D(K-1)*
8	**for** (**int** i = N-1; i>=0;i--)	c_{10}	*D(N+1)*
9	{ j =cyfra(We[i], d);	c_{11}	*DN*
10	Wy[Cyfry[j]-1]=We[i];	c_{12}	*DN*
11	Cyfry[j] -=1; } }	c_{13}	*DN*

$$T(N,K) = (D+1)\, c_1 + D\, c_2 + D\, c_3 + D(K+1)\, c_4 + DK\, c_5 + D(N+1)\, c_6 + DN\, c_7 + DK\, c_8 + D(K-1)\, c_9 + D(N+1)\, c_{10} + DN\, c_{11} + DN\, c_{12} + DN\, c_{13} =$$
$$= D\,(c_1 + c_2 + c_3 + c_4 + c_6 + c_8 - c_9 + c_{10}) + DK(c_4 + c_5 + c_8 + c_9) + c_1 + DN(c_6 + c_7 + c_{10} + c_{11} + c_{12} + c_{13}) =$$

$$T(N, K) = D\, c' + DK\, c'' + N\, D\, c''' + c_1$$

$$T(N,K) \approx N\, a + b \quad (K = 2,\ D \text{ - stała})$$

Wniosek: Czas wykonania algorytmu sortowania pozycyjnego jest liniowo zależny od liczby danych wejściowych, przy założeniu, że liczba cyfr pozycyjnych K jest stała i dużo mniejsza od liczby danych N (K<<N) oraz liczba miejsc pozycyjnych D (D – liczba bitów) jest stała i dużo mniejsza od liczby danych N (D<<N). Taka sama analiza dotyczy przykładu zapisu pozycyjnego dziesiętnego.

Uwagi:

1) Rola stabilności algorytmu

Rola stabilności algorytmu sortowania cyfr pozycyjnych N liczb dziesiętnych D cyfrowych od najmniej znaczącej cyfry

Przykład 1 - liczby o takich samych cyfrach dziesiątek są już ustawione w porządku niemalejącym przy sortowaniu wg cyfr jednostek, stąd konieczna stabilność algorytmu przy sortowaniu wg cyfr dziesiątek.

35	(1)31	11
32	11	12
31	(2)32	25
25	12	(1)31
12	(3)35	(2)32
11	25	(3)35

2) Wielokrotne wykorzystanie algorytmu przez zliczanie dla poszczególnych cyfr pozycyjnych musi być rozpoczęte od najmniej znaczącej pozycji

Przykład 2 - zastosowanie algorytmu przez zliczanie do sortowania wg kolejnych cyfr pozycyjnych wymaga rozpoczęcia od najmniej znaczącej cyfry.

35 12 11
32 11 31
31 25 12
25 35 32
12 32 25
11 31 35

3) Sortowanie pozycyjne od najbardziej znaczącej cyfry

Przykład 3 - zastosowanie algorytmu pozycyjnego wg kolejnych cyfr pozycyjnych od najbardziej znaczącej cyfry - polega grupowaniu liczb o tych samych cyfrach znaczących wg porządku rosnącego i kontynuowania procesu grupowania w wyodrębnionych grupach. Grupowanie można przeprowadzić np. za pomocą sortowania szybkiego. W przykładzie są 3 grupy wyznaczone wg cyfr dziesiątek (1,2,3) oraz w każdej grupie ustawiono liczby wg cyfr jednostek.

35	**1**2	1**1**
32	**1**1	1**2**
31	**2**5	2**5**
25	**3**5	3**1**
12	**3**2	3**2**
11	**3**1	3**5**

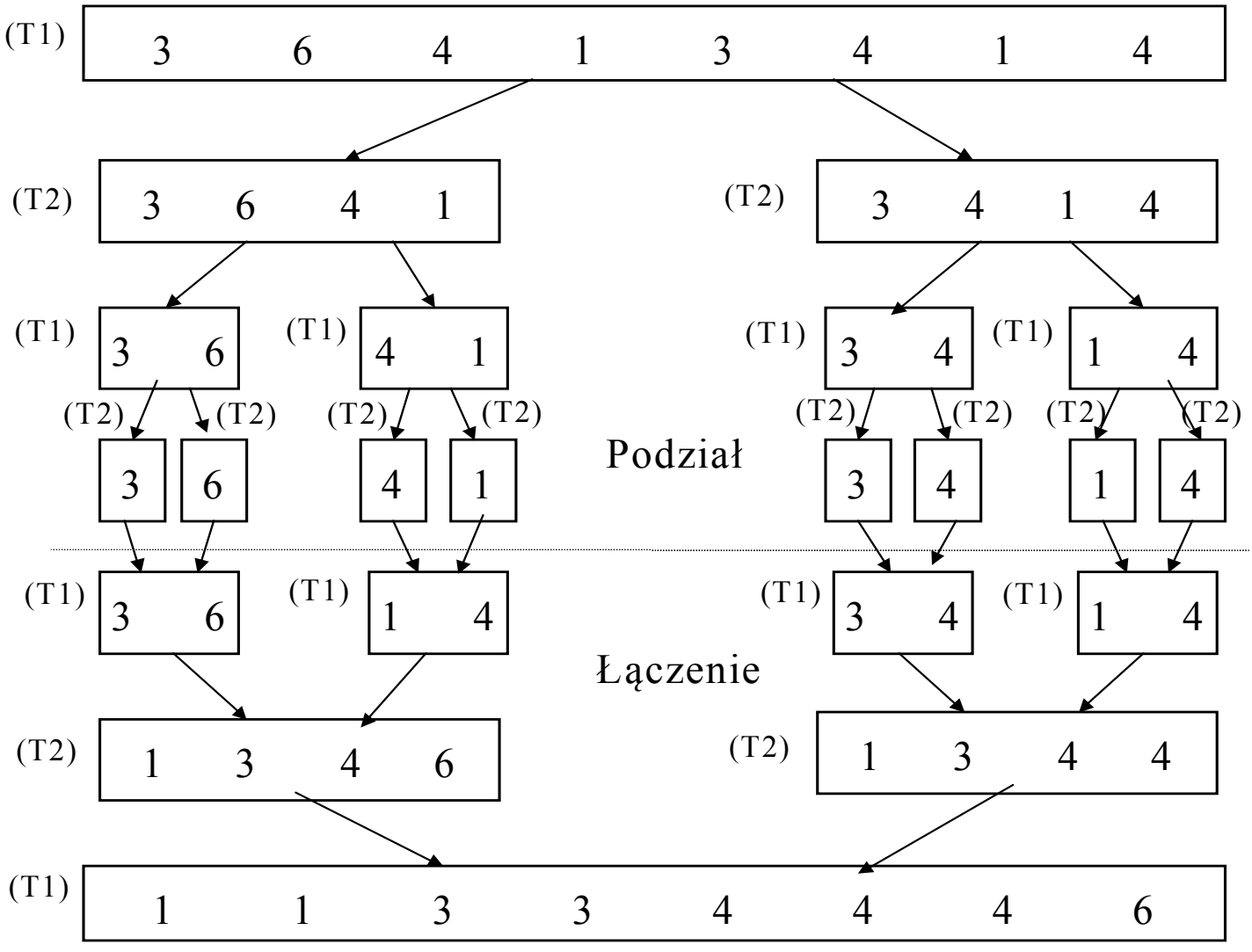
3.**Algorytmy rekurencyjne „dziel i zwyciężaj”- Sortowanie przez łączenie**

Algorytm typu „**dziel i zwyciężaj**” ma strukturę rekurencyjną. Oznacza to, że w celu rozwiązania danego problemu algorytm wywołuje sam siebie przy rozwiązywaniu podobnych podproblemów.

Kroki algorytmu typu „dziel i zwyciężaj”:

1) **Dziel:** podziel problem na podproblemy;
2) **Zwyciężaj:** rozwiąż podproblemy rekurencyjnie, chyba że są one małego
3) rozmiaru i już nie wymagają zastosowania rekursji - używa się wtedy bezpośrednich metod.
4) **Połącz:** połącz rozwiązania podproblemów tak, aby otrzymać rozwiązanie całego problemu.

3.1. Algorytmu „dziel i zwyciężaj” - algorytm sortowania przez łączenie
Poziom konceptualny - przykład sortowania parzystej liczby elementów



Algorytm sortowania przez łączenie - poziom konceptualny

(1) Dziel: *Podziel N-elementowy ciąg na dwa podciągi po N/2 elementów każdy.*

(2) Zwyciężaj: *Sortuj otrzymane podciągi, używając rekurencyjne sortowanie przez łączenie.*

(3) Połącz: *Połącz posortowane podciągi w jeden posortowany ciąg.*

Uwagi:

1. *Mechanizm rekursji nie uruchamia się, gdy ciąg przeznaczony do sortowania ma długość 1.*
2. *Podstawową operacją jest scalanie dwóch posortowanych ciągów dokonywane w kroku „Połącz".*

Algorytm sortowania przez łączenie - poziom projektowy

(1) *l ← 1; p ← N*

(2) Sort_scal (T1, T2, l, p):

(2.1) *dopóki l <p, wykonuj co następuje:*

(2.1.1) *q ← (l + p)* ***div*** *2;*

(2.1.2) *przejdź do kroku (2) i wykonaj Sort_scal(T2, T1, l, q) //podziel T2*

(2.1.3) *przejdź do kroku (2) i wykonaj Sort_scal(T2, T1, q+1, p) //podziel T2*

(2.1.4) *Polacz(T1, T2, l, q, p) //Połącz T2(l..q) z T2(q+1..p) na T1*

Algorytm łączenia podciągów posortowanych w wyniku sortowania przez łączenie:

Algorytm Połącz - poziom konceptualny

(1) *wskaż na pierwszy element pierwszego podciągu;*

(2) *wskaż na pierwszy element drugiego podciągu;*

(3) *wybierz najmniejszy z dwóch wskazanych elementów;*

(4) *wstaw wybrany element jako pierwszy w ciągu wyjściowym;*

(5) *dopóki nie wyczerpiesz elementów z jednego z podciągów, wykonuj:*

(5.1) *wskaż na następny element z ciągu, z którego pobrano najmniejszy element;*

(5.2) *wybierz najmniejszy element z dwóch wskazanych elementów;*

(5.3) *wstaw wybrany element jako kolejny w ciągu wyjściowym;*

(6) *wstaw kolejno do ciągu wyjściowego wszystkie elementy z niepustego podciągu wejściowego.*

Algorytm Połącz - poziom projektowy

Dane: T2(l..q) oraz T2(q + 1..p) jako ciąg wejściowy i T1 jako ciąg wyjściowy

Polacz (T1, T2, l, q, p)

(1) *i ← l; j ← q + 1; x ← l;*

(2) *dopóki i <= q oraz j<= p, wykonuj co następuje:*

{wybór najmniejszego elementu z dwóch podtablic T2(l..q) oraz T2(q+1..p) i ustawienie go w porządku niemalejącym w tablicy T1(l..p) na pozycji „x”}

(2.1) *jeśli T2(i) > T2(j), to:*

(2.1.1) *T1(x) ← T2(j);*

(2.1.2) *j ← j + 1;*

(2.2) *w przeciwnym przypadku:*

(2.2.1) *T1(x) ← T2(i);*

(2.2.2) *i ← i + 1;*

(2.3) *x ← x + 1;*

{przepisanie reszty elementów z niepustej podtablicy T2(l..q) lub T2(q+1..p) na kolejne miejsca tablicy T1(l..p)}

(3) *jeśli j > p, to:*

(3.1) *y ← i*

(3.2) *dopóki y <= q, wykonuj co następuje:*

(3.2.1) *T1(x) ← T2(y);*

(3.2.2) *x ← x + 1;*

(3.2.3) *y ← y + 1;*

(4) *w przeciwnym przypadku:*

(4.1) *y ← j;*

(4.2) *dopóki y <=p, wykonuj co następuje:*

(4.2.1) *T1(x) ← T2(y);*

(4.2.2) *x ← x + 1;*

(4.2.3) *y ← y + 1;*

Przykład sortowania parzystej liczby elementów przez łączenie
Poziom implementacji

a) sortowanie pierwszej połowy tablicy wejściowej

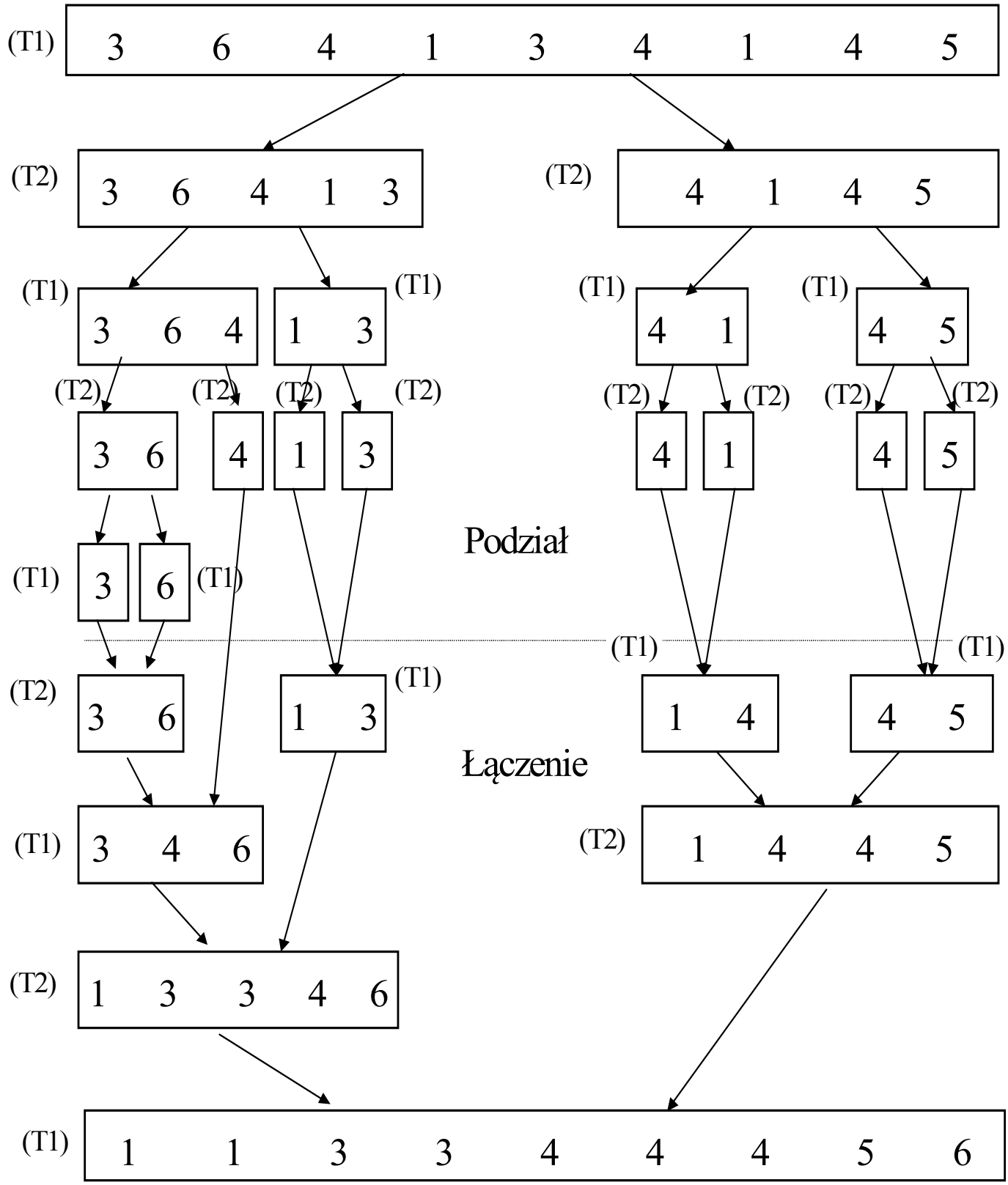
Działanie		dane działań				T1								T2							
		l	*q*	*q+1*	*p*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
Sort_Lacz		1	4	5	8	3	6	4	1	3	4	1	4	3	6	4	1	3	4	1	4
Sort_Lacz		1	2	3	4									3	6	4	1				
Sort_Lacz		1	1	2	2	3	6														
Sort_Lacz	K	1			1									3							
Sort_Lacz	K	2			2										6						
Sort_Lacz		1	1	2	2									***3***	***6***						
Lacz		1	1	2	2	***3***	***6***														
Sort_Lacz	K	1	1	2	2	3	6														
Sort_Lacz		3	3	4	4	3	6	4	1												
Sort_Lacz	K	3			3	3	6									4					
Sort_Lacz	K	4			4	3	6										1				
Sort_Lacz		3	3	4	4	3	6									***4***	***1***				
Lacz		3	3	4	4	3	6	***1***	***4***												
Sort_Lacz	K	3	3	4	4	3	6	1	4												
Sort_Lacz		1	2	3	4	***3***	***6***	***1***	***4***												
Lacz		1	2	3	4									***1***	***3***	***4***	***6***				
Sort_Lacz	K	1	2	3	4									1	3	4	6				

b) sortowanie drugiej połowy tablicy wejściowej

Działanie						T1								T2							
		l	*q*	*q+1*	*p*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
Sort_Lacz		5	6	7	8									*1*	*3*	*4*	*6*	3	4	1	4
Sort_Lacz		5	5	6	6					3	4			*1*	*3*	*4*	*6*				
Sort_Lacz	K	5			5									*1*	*3*	*4*	*6*	3			
Sort_Lacz	K	6			6									*1*	*3*	*4*	*6*		4		
Sort_Lacz		5	5	6	6									*1*	*3*	*4*	*6*	***3***	***4***		
Lacz		5	5	6	6					***3***	***4***			*1*	*3*	*4*	*6*				
Sort_Lacz	K	5	5	6	6					3	4			*1*	*2*	*3*	*6*				
Sort_Lacz		7	7	8	8					3	4	1	4	*1*	*3*	*4*	*6*				
Sort_Lacz	K	7			7					3	4			*1*	*3*	*4*	*6*			1	
Sort_Lacz	K	8			8					3	4			*1*	*3*	*4*	*6*				4
Sort_Lacz		7	7	8	8					3	4			*1*	*3*	*4*	*6*			***1***	***4***
Lacz		7	7	8	8					3	4	***1***	***4***	*1*	*3*	*4*	*6*				
Sort_Lacz	K	7	7	8	8					3	4	1	4	*1*	*3*	*4*	*6*				
Sort_Lacz		5	6	7	8					***3***	***4***	***1***	***4***	*1*	*3*	*4*	*6*				
Lacz		5	6	7	8									*1*	*3*	*4*	*6*	***1***	***3***	***4***	***4***
Sort_Lacz	K	5	6	7	8									*1*	*3*	*4*	*6*	1	3	4	4
Sort_Lacz		1	4	5	8									***1***	***3***	***4***	***6***	***1***	***3***	***4***	***4***
Lacz		1	4	5	8	***1***	***1***	***3***	***3***	***4***	***4***	***4***	***6***								
Sort_Lacz	K	1	4	5	8	1	1	3	3	4	4	4	6								

K - zakończenie wykonania danego egzemplarza procedury *Sort_Lacz*

3.2. Algorytmu „dziel i zwyciężaj” - algorytm sortowania przez łączenie
Poziom konceptualny -przykład sortowania nieparzystej liczby elementów



Przykład sortowania nieparzystej liczby elementów przez łączenie
Poziom implementacji

a) sortowanie pierwszej części tablicy wejściowej T[1..5]

Działanie		dane działań				T1									T2								
		l	*q*	*q+1*	*p*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
Sort_Lacz		1	5	6	9	3	6	4	1	3	4	1	4	5	3	6	4	1	3	4	1	4	5
Sort_Lacz		1	3	4	5										3	6	4	1	3				
Sort_Lacz		1	2	3	3	3	6	4															
Sort_Lacz		1	1	2	2										3	6							
Sort_Lacz	K	1			1	3																	
Sort_Lacz	K	2			2		6																
Sort_Lacz		1	1	2	2	***3***	***6***																
Lacz		1	1	2	2										***3***	***6***							
Sort_Lacz	K	1	1	2	2										3	6							
Sort_Lacz	K	3			3												4						
Sort_Lacz		1	2	3	3										***3***	***6***	***4***						
Lacz		1	2	3	3	***3***	***4***	***6***															
Sort_Lacz	K	1	2	3	3	3	4	6															
Sort_Lacz		4	4	5	5	3	4	6	1	3													
Sort_Lacz	K	4			4	3	4	6										1					
Sort_Lacz	K	5			5	3	4	6											3				
Sort_Lacz		4	4	5	5	3	4	6										***1***	***3***				
Lacz		4	4	5	5	3	4	6	***1***	***3***													
Sort_Lacz	K	4	4	5	5	3	4	6	1	3													
Sort_Lacz		1	3	4	5	***3***	***4***	***6***	***1***	***3***													
Lacz		1	3	4	5										***1***	***3***	***3***	***4***	***6***				
Sort_Lacz	K	1	3	4	5										1	3	3	4	6				

b) sortowanie drugiej części tablicy wejściowej T[6..9]

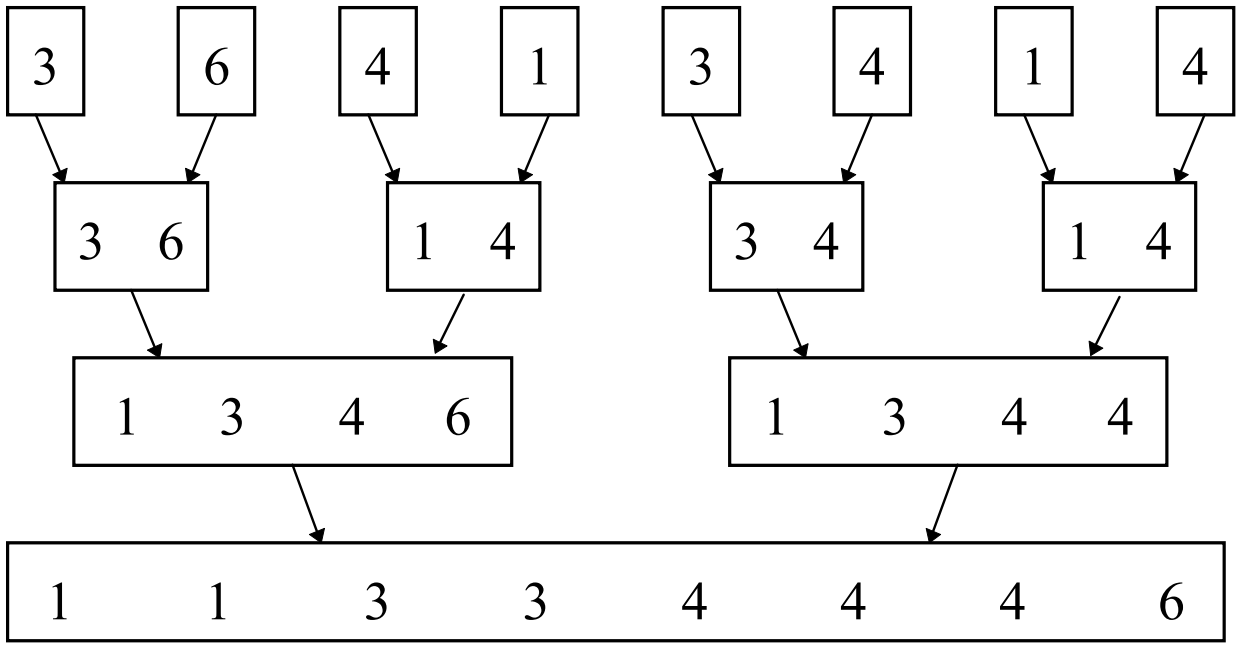
Działanie		dane działań				T1									T2								
		l	*q*	*q+1*	*p*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
Sort_Lacz		6	7	8	9						4	1	4	5	*1*	*3*	*3*	*4*	*6*				
Sort_Lacz		6	6	8	7						4	1			*1*	*3*	*3*	*4*	*6*				
Sort_Lacz	K	6			6										*1*	*3*	*3*	*4*	*6*	4			
Sort_Lacz	K	7			7										*1*	*3*	*3*	*4*	*6*		1		
Sort_Lacz		6	6	7	7										*1*	*3*	*3*	*4*	*6*	***4***	***1***		
Lacz		6	6	7	7						***1***	***4***			*1*	*3*	*3*	*4*	*6*				
Sort_Lacz	K	6	6	7	7						1	4			*1*	*3*	*3*	*4*	*6*				
Sort_Lacz		8	8	9	9						1	4	4	5	*1*	*3*	*3*	*4*	*6*				
Sort_Lacz	K	8			8										*1*	*3*	*3*	*4*	*6*			4	
Sort_Lacz	K	9			9										*1*	*3*	*3*	*4*	*6*				5
Sort_Lacz		8	8	9	9						1	4			*1*	*3*	*3*	*4*	*6*			***4***	***5***
Lacz		8	8	9	9						1	4	***4***	***5***	*1*	*3*	*3*	*4*	*6*				
Sort_Lacz	K	8	8	9	9						1	4	4	5	*1*	*3*	*3*	*4*	*6*				
Sort_Lacz		6	7	8	9						***1***	***4***	***4***	***5***	*1*	*3*	*3*	*4*	*6*				
Lacz		6	7	8	9										*1*	*3*	*3*	*4*	*6*	***1***	***4***	***4***	***5***
Sort_Lacz	K	6	7	8	9										*1*	*3*	*3*	*4*	*6*	1	4	4	5
Sort_Lacz		1	5	6	9										***1***	***3***	***3***	***4***	***6***	***1***	***4***	***4***	***5***
Lacz		1	5	6	9	***1***	***1***	***3***	***3***	***4***	***4***	***4***	***5***	***6***									
Sort_Lacz	K	1	5	6	9	1	1	3	3	4	4	4	5	6									

3.3. Sortowanie wstępujące przez łączenie

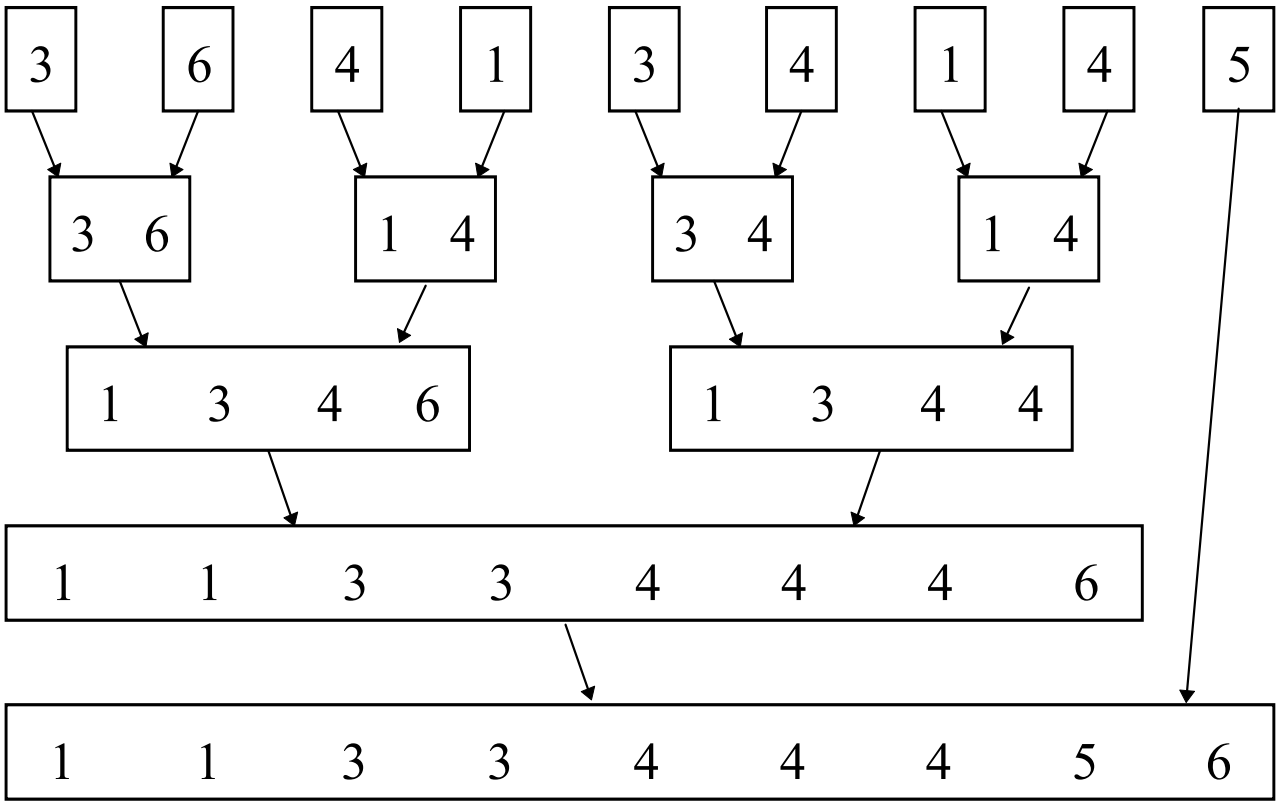
Oprócz sortowania przez łączenie zstępujące typu „dziel i rządź” istnieje sortowanie wstępujące (nierekurencyjne) typu „łącz i zwyciężaj, które jest o około 10% mniej wydajne od zstępującego.

Przykłady sortowania wstępującego zbiorów parzystych i nieparzystych - poziom konceptualny

a)parzystych



b)nieparzystych



3.4. Sortowanie przez łączenie zstępujące, implementacja w C/C++

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <mem.h>

const int m=20;
const long N=40000L;

void lacz(int T1[], int T2[], long l, long q, long p);
void sort_lacz(int T1[], int T2[], long l, long p);
void sortlacz(int T [], long l, long p);

void wypelnij(int T[], long& ile);
void wyswietl(int T[], long ile);

void main(int argc, char* argv[])
{ long ile=0;
  int T[N];
  wypelnij(T,ile);
  sortlacz(T,0,ile-1);
  wyswietl(T,ile);
  getch();
}
void wypelnij(int T[], long& ile)
    {   srand(3);
        for (long i=0; i<N; i++)
             T[i]=rand();
        ile=N;
     }
void wyswietl(int T[], long ile)
 {
    for (long i=0; i<ile; i++)
         { printf("%d \n", T[i]);
           if (i%m==0)
             {  char z=getch();
                if (z=='k') return; }
          }
  }
```

```
//##########sortowanie przez laczenie###################
 void lacz(int T1[], int T2[], long l, long q, long p)
  { long x,y,i,j;
     i=l;  j=q+1;  x=l;
    while (i <=q && j<=p)
     { if (T2[i] > T2[j])
         { T1[x]=T2[j];
           j++;}
        else
         { T1[x]=T2[i];
            i++;  }
        x++;
       }
  //kopiuj reszte podtablicy T2 niepustej do tablicy wyjsciowej T1
   if (j>p)
    for (long y=i; y<=q; y++)
     {  T1[x] = T2[y];
        x++; }
     else
      for (long y=j; y<=p; y++)
       {  T1[x] = T2[y];
          x++; }
   }

 void sort_lacz(int T1[], int T2[], long l, long p)
 //T1 - tablica glowna, T2 tablica pomocnicza
 // (w programie na stosie procedury sortlacz)
 { long q;
    if (l < p)
     {  q= (l+p)/2;
        sort_lacz(T2,T1,l,q);
        sort_lacz(T2,T1,q+1,p);
        lacz(T1,T2,l,q,p);
     }
   }
 void sortlacz(int T [], long l, long p)
  {  int Pom[N];
     memmove(Pom,T,(p+1)*sizeof(int));
     sort_lacz(T,Pom,l,p);
  }
```