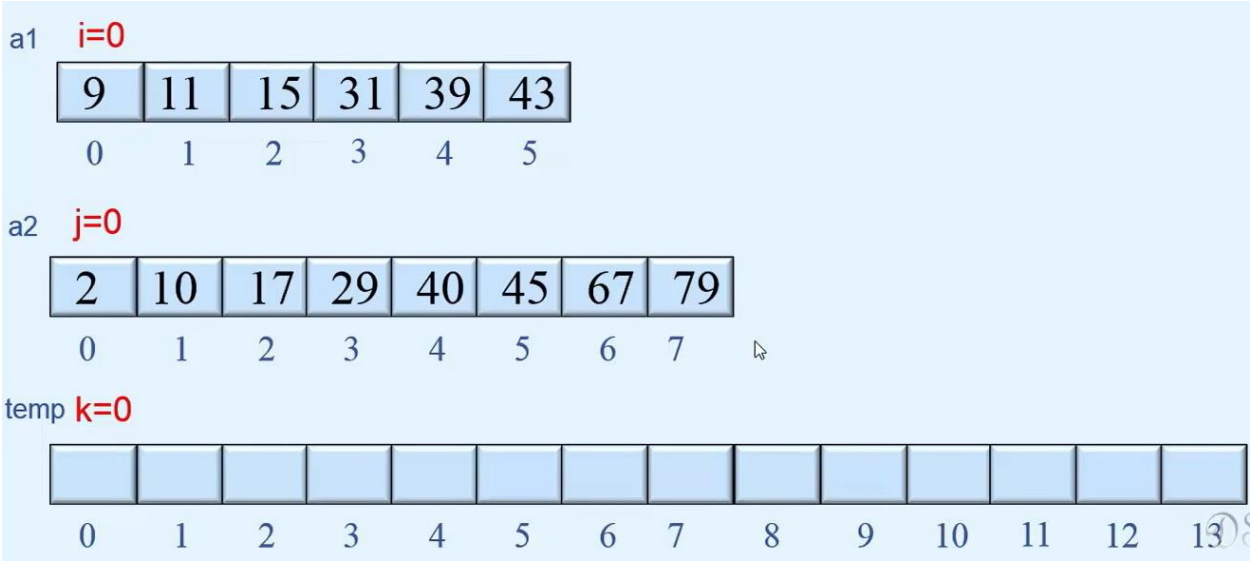
Algoritmi sortiranja -2

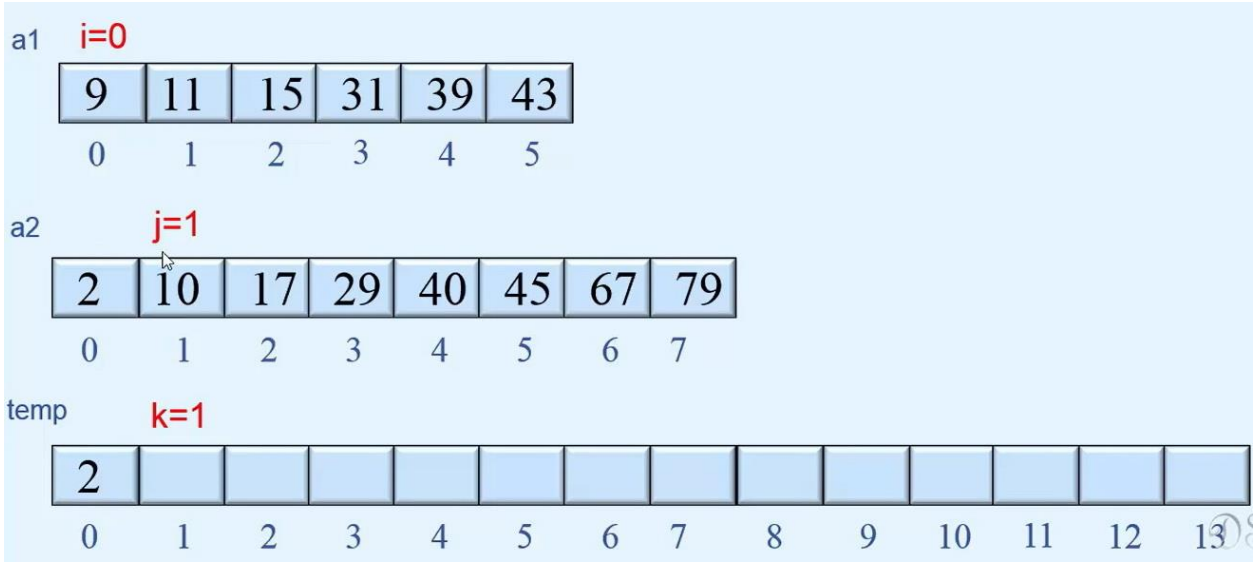
Spajanje sortiranih nizova u novi sortirani niz

- Neka je a1[] sortirani niz dužine n1
- Neka je a2[] sortirani niz dužine n2
- Potrebno je kreirati novi niz temp[] dužine n1+n2 od elemenata niza a1[] i a2[] tako da bude sortiran

Spajanje sortiranih nizova -1

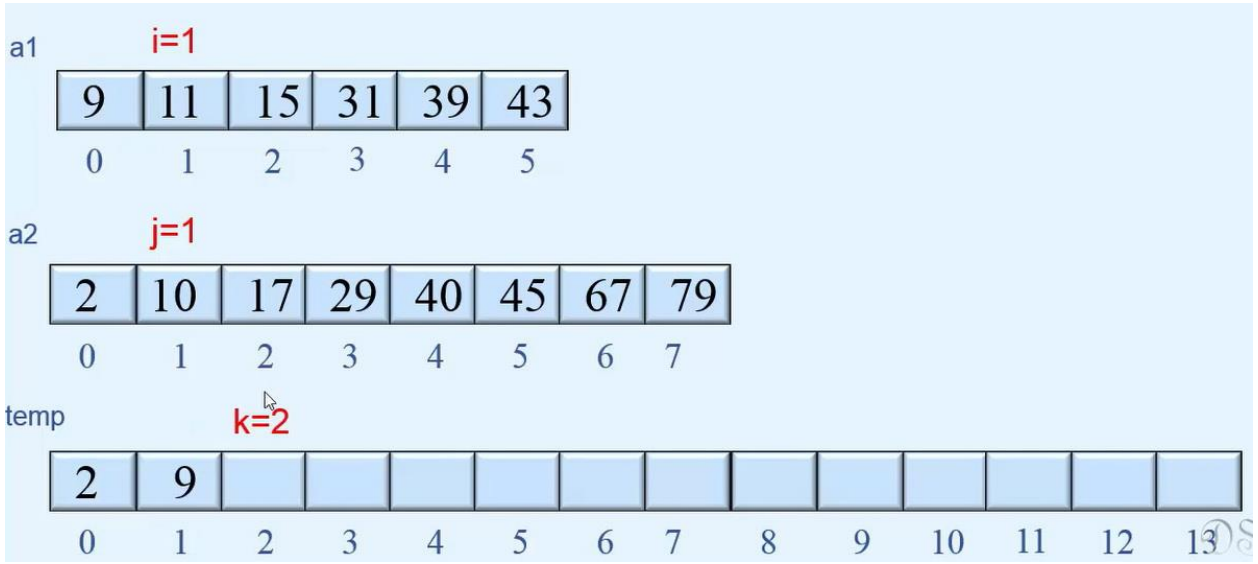


Spajanje sortiranih nizova -2



a2[0] <a1[0]
temp[0] = a2[0]
j=1; inkrementira se brojač niza a2
k=1

Spajanje sortiranih nizova -3



a1[0]<a2[1]
temp[1] = a1[0]
i=1; inkrementira se brojač prvog niza
k=2; inkrementira se brojač rezultujućeg niza

Funkcija Merge() koja spaja sortirane nizove

```
public static int[] Merge(int[] a1, int[] a2)
{
    int n1 = a1.Length;
    int n2 = a2.Length;

    int[] temp = new int[n1 + n2];

    int i = 0, j = 0, k = 0;

    while (i<n1 && j<n2)
    {
        if (a1[i] <a2[j])
        {
            temp[k++] = a1[i++];
        }
        else
        {
            temp[k++] = a2[j++];
        }
    }
    //a2 se zavrsio kopiraj preostale elemente niza a1
    while (i<n1)
    {
        temp[k++] = a1[i++];
    }

    //a1 se zavrsio kopiraj preostale elemente niza a2
    while (j<n2)
    {
        temp[k++] = a2[j++];
    }

    return temp;
}
```

Pomoćne funkcije

```
static int[] KreirajSortiraniNiz(int n)
{
    int[] x = new int[n];
    for (int i = 0; i < n; i++)
    {
        x[i] = rnd.Next(1, 101); // od 1 do 100
    }
    Array.Sort(x);
    return x;
}
```

```
public static  Random rnd = new Random();
// Generator je staticko polje klase Program
```

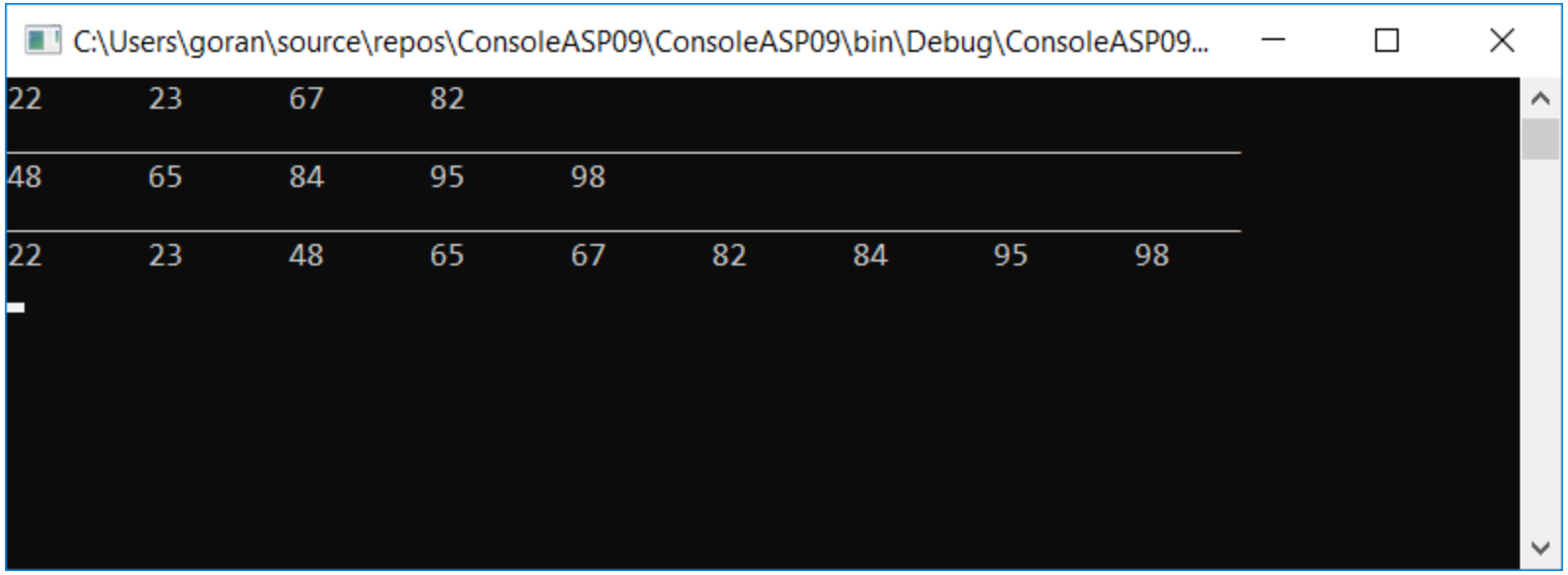
```
static void PisiNiz(int[] x)
{
    for (int i = 0; i < x.Length; i++)
    {
        Console.Write(x[i] + "\t");
    }
    Console.WriteLine();
}
```

```
static void Linija(int n)
{
    //iscrtava liniju  duzine n na konzoli
    Console.WriteLine("".PadRight(n, '_'));
}
```

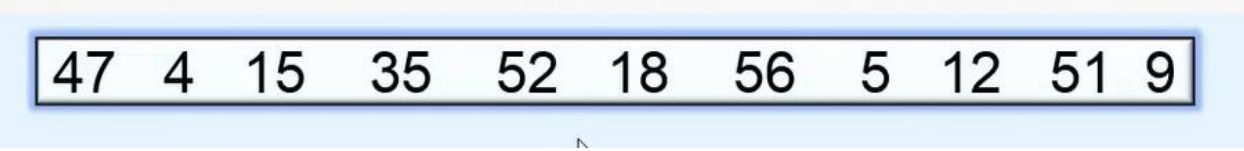
Poziv funkcije za spajanje nizova

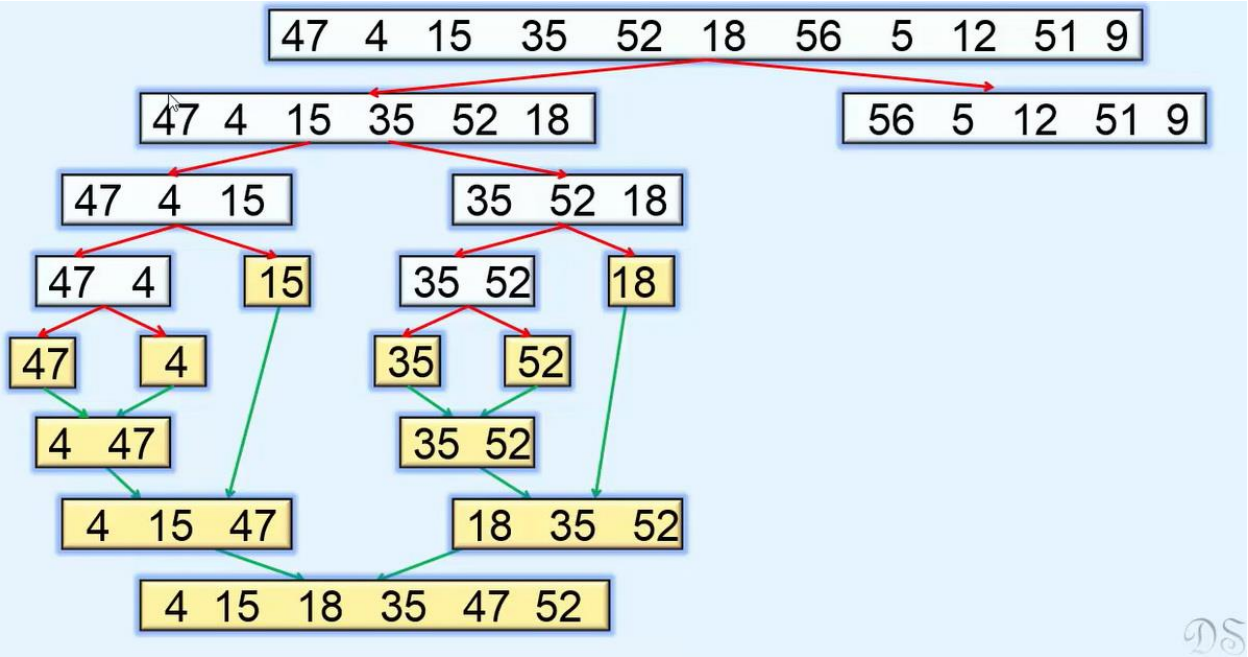
```
static void Main(string[] args)
{
    int[] a1 = KreirajSortiraniNiz(4);
    int[] a2 = KreirajSortiraniNiz(5);

    int[] temp = Merge(a1,a2);

    PisiNiz(a1);
    Linija(70);
    PisiNiz(a2);
    Linija(70);

    PisiNiz(temp);

    Console.ReadLine();
}
```

Rezultat spajanja nizova



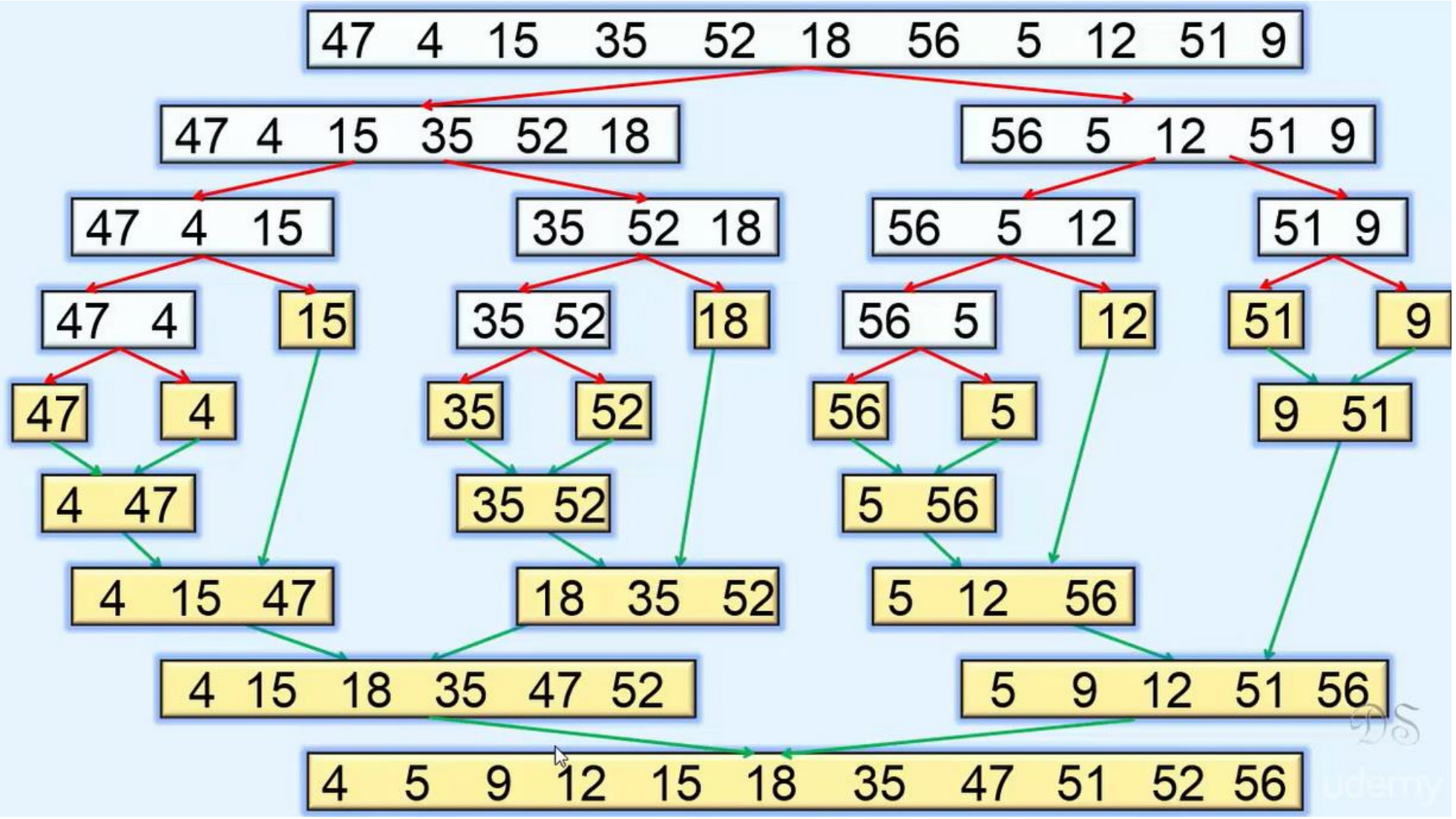
Algoritam Merge Sort





- Niz se rekurzivno deli na dva dela, sve dok dužina niza ne postane 1
- Nizovi dužine 1 mogu se smatrati sortiranim nizovima i spajaju se u novi sortirani niz
- Nastavlja se spajanje sortiranih nizova sve dok se ne dobije ceo niz

Algoritam Merge Sort



Adaptirana funkcija Merge() – 1. deo

```
// Spaja dva sortirana niza
// Spaja dva sortirana podniza x[l..m] i x[m+1..r]
static void Merge(int[] x, int l, int m, int r)
{
    // Dužine podnizova
    int n1 = m - l + 1;
    int n2 = r - m;

    // Privremeni nizovi
    int[] L = new int[n1];
    int[] R = new int[n2];

    // Kopiranje podataka iz glavnog niza x u privremene nizove L i R
    int i,j;

    // Kopiranje prvog podniza x[l..m] u L
    for ( i = 0; i < n1; ++i)
    {
        L[i] = x[l + i];
    }

    // Kopiranje drugog podniza x[m+1..r] u R
    for ( j = 0; j < n2; ++j)
    {
        R[j] = x[m + 1 + j];
    }
```

Adaptirana funkcija Merge() – 2. deo

```
// Resetovanje indeksa i i j na 0 kako bismo ih koristili za poređenje elemenata

// Indeks za prvi podniz L
    i = 0;

// Indeks za drugi podniz R
    j = 0;

// Indeks za glavni niz x
int k = l;

// Spajanje privremenih nizova nazad u glavni niz x[l..r]
while (i < n1 && j < n2)
{
    // Poređenje elemenata iz L i R i smeštanje manjeg elementa u x
    if (L[i] <= R[j])
    {
        x[k] = L[i];
        i++;
    }
    else
    {
        x[k] = R[j];
        j++;
    }
    // Pomeranje indeksa u glavnom nizu
    k++;
}
```

Adaptirana funkcija Merge() – 3. deo

```
// Kopiranje preostalih elemenata iz L, ako postoje
    while (i < n1)
    {
        x[k] = L[i];
        i++;
        k++;
    }

    // Kopiranje preostalih elemenata iz R, ako postoje
    while (j < n2)
    {
        x[k] = R[j];
        j++;
        k++;
    }
}
```

Rekurzivna funkcija Sort sa parametrima

```
static void Sort(int[] x, int l, int r)
{
    // `l` predstavlja početak niza koji se trenutno sortira
    // `r` predstavlja kraj niza koji se trenutno sortira
    if (l < r)
    {
        // Pronalaženje srednjeg indeksa
        int m = l + (r - l) / 2;

        // Rekurzivno sortiranje prve i druge polovine
        Sort(x, l, m);
        Sort(x, m + 1, r);

        // Spajanje sortiranih polovina
        Merge(x, l, m, r);
    }
}
```

Funkcija MergeSort()

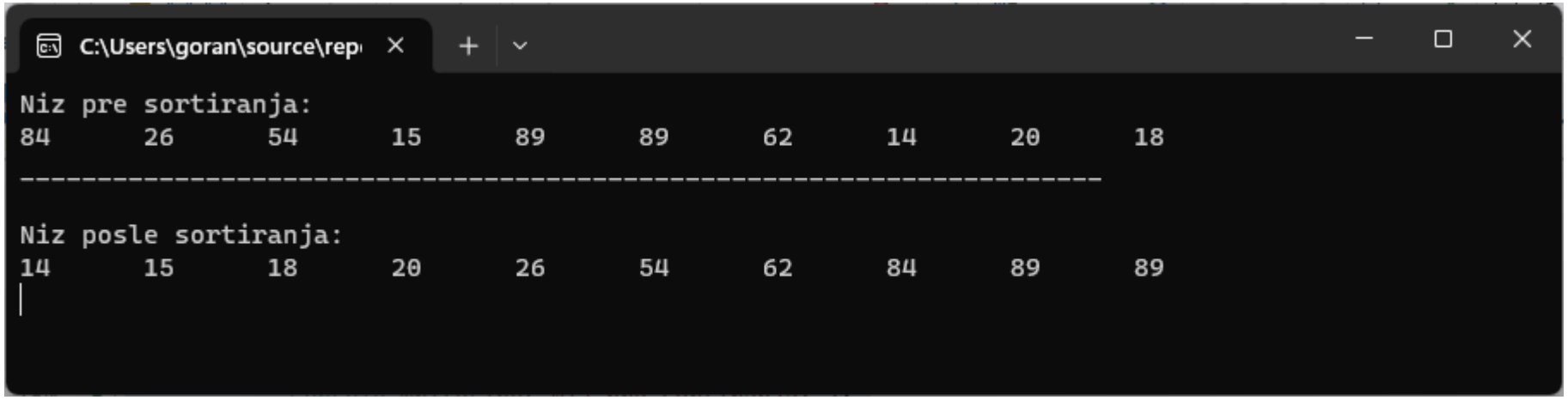
```
public static void MergeSort(int[] x)
{
    int n = x.Length;
    // Pokretanje merge sort algoritma
    Sort(x, 0, n - 1);

}
```

Pomoćna funkcija

```
static int[] KreirajNiz(int n)
{
    Random rnd = new Random();
    int[] x = new int[n];
    for (int i = 0; i < n; i++)
    {
        x[i] = rnd.Next(1, 101); // od 1 do 100
    }

    return x;
}
```

Poziv funkcije MergeSort()

```
static void Main()
{
    int[] x = KreirajNiz(10);
    Console.WriteLine("Niz pre sortiranja:");
    PisiNiz(x);
    Linija(70);
    MergeSort(x);
    Console.WriteLine("\nNiz posle sortiranja:");
    PisiNiz(x);
    Console.ReadLine();
}
```



Analiza MergeSort algoritma

- Niz od n elemenata se iterativno deli na dva dela približno $\log_2 n$ puta
- Nakon deljenja niza $\log_2 n$ puta imamo n podnizova dužine 1
- Vremenska kompleksnost algoritma je O(nlogn)

Quick Sort algoritam

- Efikasan algoritam sortiranja
- Tehnika podeli pa vladaj
- Ceo niz x[] se deli na dve particije: levu i desnu
- Svi elementi leve particije su manji od elementa koji se zove pivot
- Svi elementi desne grupe su veći ili jednaki od pivot elementa
- Inicijalno se za pivot postavlja bilo koji element niza. npr prvi:
  pivot= x[0]
- Svakim prolaskom kroz algoritam određuje se tačna pozicija pivota
- Leva i desna particija se ponovo dele na particije na isti način

Podela niza i podniza na particije

```
static int Particija(int[] x, int low, int high)
```

- **low** parametar označava početni indeks opsega niza koga delimo na particije. U početku je low =0.
- **high** parametar označava krajnji indeks opsega niza opsega niza koga delimo na particije. Na početku high =n-1
- U funkciji Particija, pivot se obično bira kao element na poziciji low

```
int pivot = x[low];
int left = low + 1;
int right = high;
```

Indeksi left i right

- Pivot se poredi sa elementima pomoću left i right indeksa
- Tokom particionisanja, petlja povećava index left dok ne pronađe element veći od pivota
- Petlja smanjuje index right dok ne pronađe element manji od pivota
- Kada se petlja završi, index left će biti pozicioniran iza poslednjeg elementa manjeg od pivota
- Kada se petlja završi, index right će biti pozicioniran ispred poslednjeg elementa većeg od pivota.

```
while (true)
   {
       while (left <= right && x[left] < pivot)
       {
           left++;
       }

       while (left <= right && x[right] > pivot)
       {
           right--;
       }
```

Razmena elemenata na pozicijama left i right

- Ako je element na poziciji left manj ili jednak od elementa na poziciji right, to znači da smo pronašli par elemenata koji treba razmeniti
- Razmenjujemo ove elemente kako bismo postigli delimično sortiranje u odnosu na pivot

```
if (left <= right)
{
    // razmeni elemente left i right
    int temp = x[left];
    x[left] = x[right];
    x[right] = temp;
}
```

Indeks left preskače indeks right

- Kada indeks left postane veći od indeksa right to znači da smo došli do tačke gde je left "preskočio" preko right
- Odnosno da smo prošli kroz ceo niz i postavili elemente u ispravan redosled u odnosu na pivot
- Sada se pivot razmenjuje sa elementom na poziciji right jer znamo da je right indeks sada postavljen tačno tamo gde treba biti pivot

```
else
{
        //razmena pivota sa elementom na poziciji right
    int temp = x[low];
    x[low] = x[right];
    x[right] = temp;
    // vrati index pivota
    return right;
}
```

Funkcija za particionisanje niza

```
static int Particija(int[] x, int low, int high)
{
    int pivot = x[low];
    int left = low + 1;
    int right = high;

    while (true)
    {
        while (left <= right && x[left] < pivot)
        {
            left++;
        }

        while (left <= right && x[right] > pivot)
        {
            right--;
        }

        if (left <= right)
        {
            // razmeni elemente left i right
            int temp = x[left];
            x[left] = x[right];
            x[right] = temp;
        }
        else
        {
                //razmena pivota sa elementom na poziciji right
            int temp = x[low];
            x[low] = x[right];
            x[right] = temp;
            // vrati index pivota
            return right;
        }
    }
}
```

Quick Sort algoritam

```
static void QuickSort(int[] x, int low, int high)
{
    if (low < high)
    {
        int pivotIndex = Particija(x, low, high);

        QuickSort(x, low, pivotIndex - 1);
        QuickSort(x, pivotIndex + 1, high);
    }
}
```

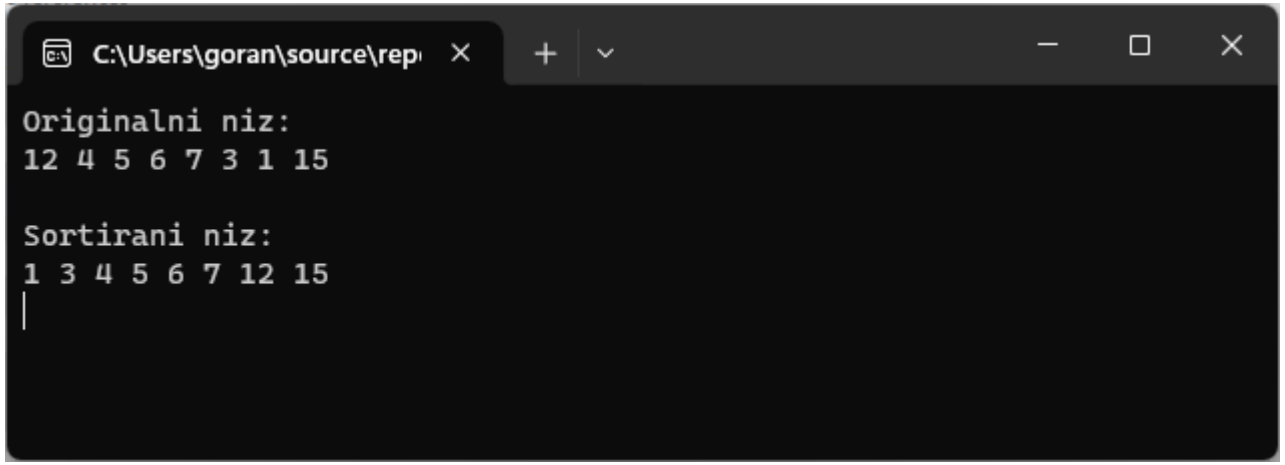
Quick Sort algoritam

- Ako low nije veće ili jednako high, to znači da podniz ima više od jednog elementa i da ga treba dalje sortirati
- Ako low postane veće od high, to znači da podniz ima jedan ili nijedan element i ne zahteva dalje sortiranje
- Funkcija Particija bira pivot, podeli niz na dva dela (levo sa elementima manjim od pivota, desno sa elementima većim od pivota) i vraća indeks gde se pivot trenutno nalazi
- Nakon particionisanja, rekurzivno pozivamo QuickSort funkciju za levi podniz, tj. podniz sa elementima manjim od pivota :
  - **QuickSort(x, low, pivotIndex - 1);**
- Slično tome, rekurzivno pozivamo QuickSort funkciju za desni podniz, tj. podniz sa elementima većim od pivota:
  - **QuickSort(x, pivotIndex + 1, high);**

Poziv funkcije QuickSort()

```
static void Main()
{
    int[] x = { 12, 4, 5, 6, 7, 3, 1, 15 };

    Console.WriteLine("Originalni niz:");
    StampajNiz(x);

    QuickSort(x, 0, x.Length - 1);

    Console.WriteLine("\nSortirani niz:");
    StampajNiz(x);

    Console.ReadLine();
}
```

Poziv funkcije QuickSort()



Karakteristike Quick Sort algoritma

- Kada se dobro implementira dvostruko je brži od Merge Sort algoritma
- Prosečna vremenska kompleksnost je O(nlogn)

Pitanje 1

Algoritam Quick Sort je:
a. iterativni algoritam
b. rekurzivni algoritam

Odgovor: b

Pitanje 2

Tehnika podeli pa vladaj nije karakteristična za sledeći algoritam soriranja:

a. Merge Sort
b. Selection sort
c. Quick sort

Odgovor: b

Pitanje 3

Prosečna vremenska kompleksnost Quick Sort algoritma je:

a. $O(n^2)$
b. $O(n)$
c. $O(nlogn)$

Odgovor: c

Pitanje 4

Kod Quick Sort algoritma svi elementi desne particije su:

a. veći od pivot elementa
b. manji od pivot elementa
c. mogu biti i veći i manji od pivot elementa

Odgovor: a