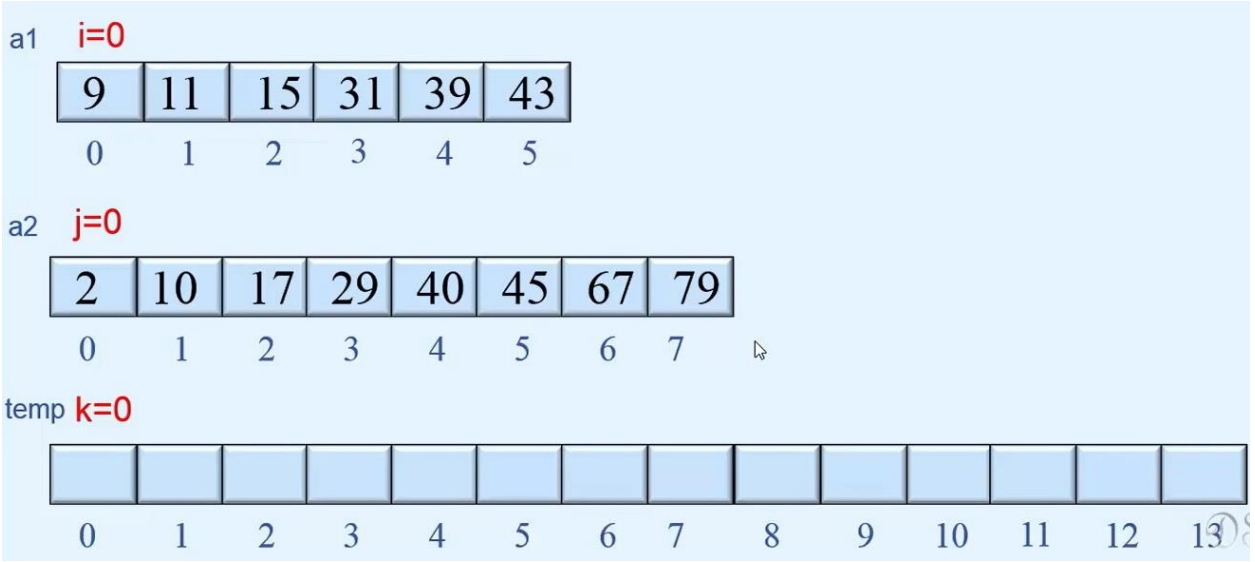
Algoritmi sortiranja -2

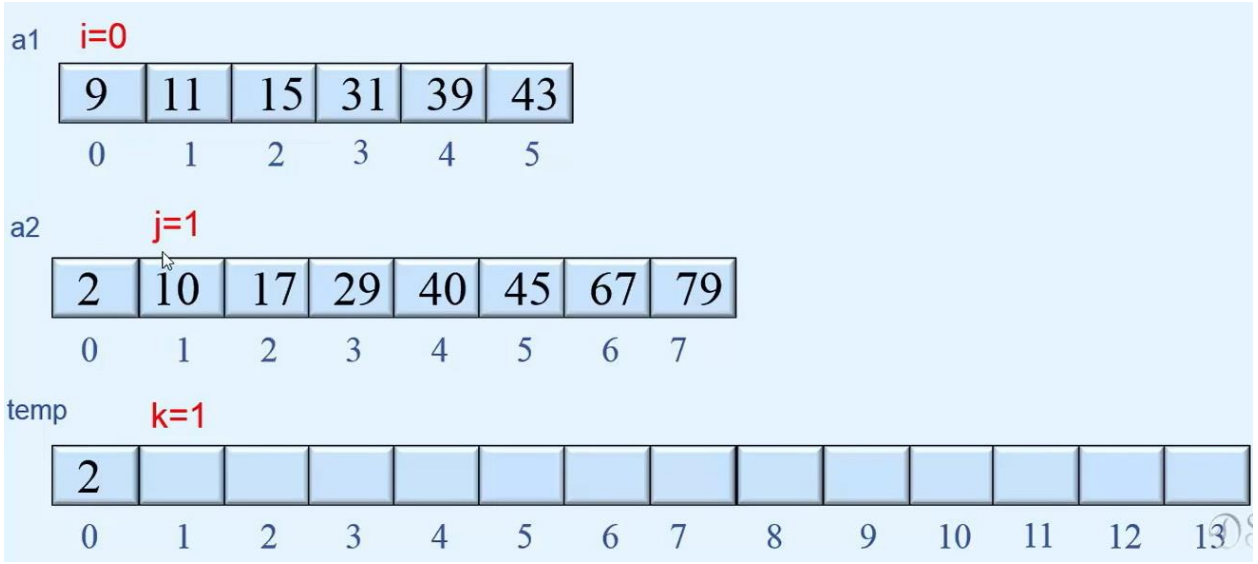
Spajanje sortiranih nizova u novi sortirani niz

- 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

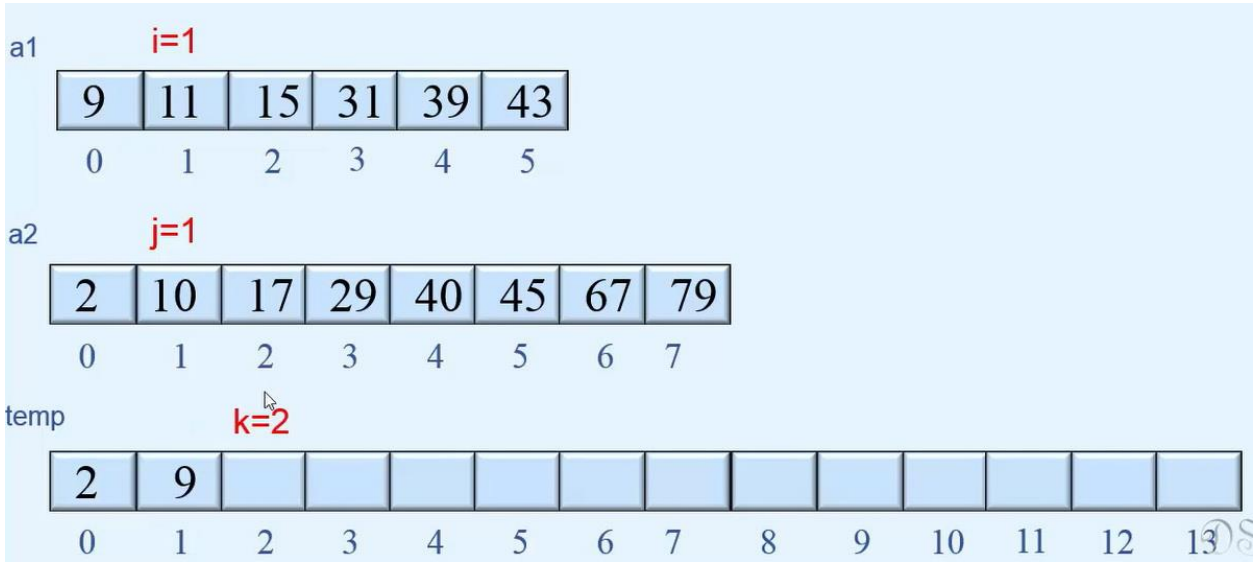


Spajanje sortiranih nizova -2



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
```

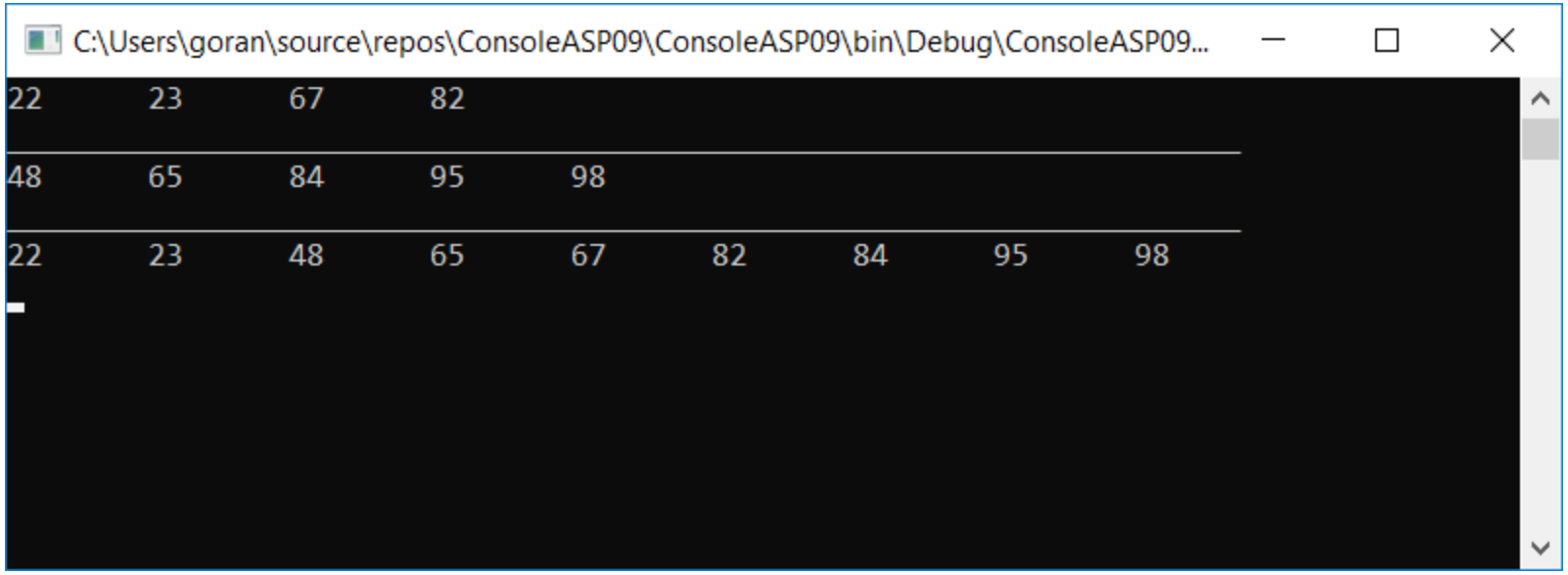
```
static void PisiNiz(int[] x)
{
    for (int i = 0; i < x.Length; i++)
    {
        Console.Write(x[i] + "\t");
    }
    Console.WriteLine();
}
```

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

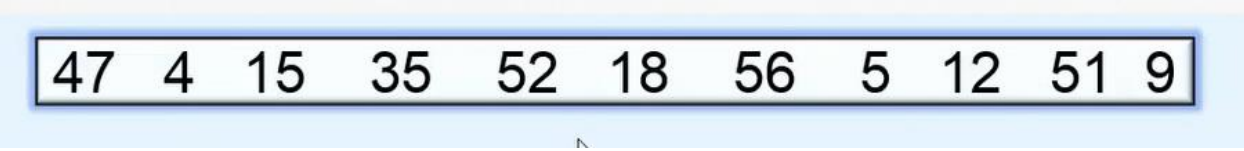
Poziv funkcije za spajanje nizova

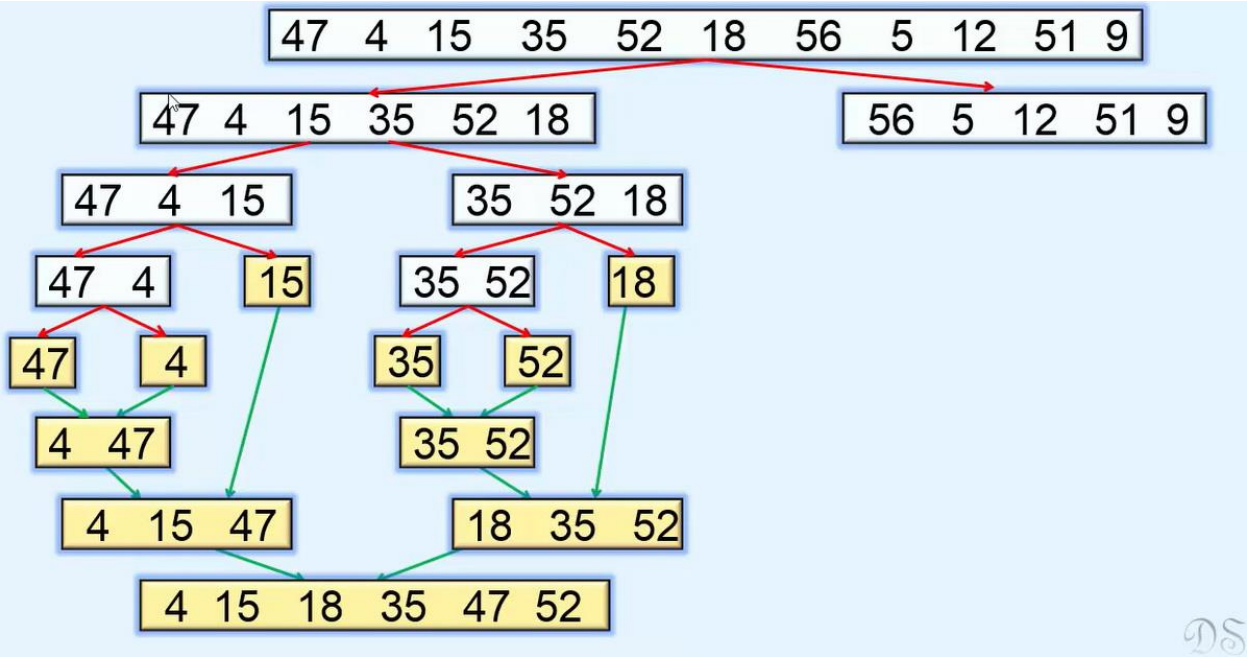
```
static void Main(string[] args)
{
    int[] a1 = KreirajSortiraniNiz(4);
    int[] a2 = KreirajSortiraniNiz(5);

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

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

    PisiNiz(temp);

    Console.ReadLine();
}
```

Rezultat spajanja nizova



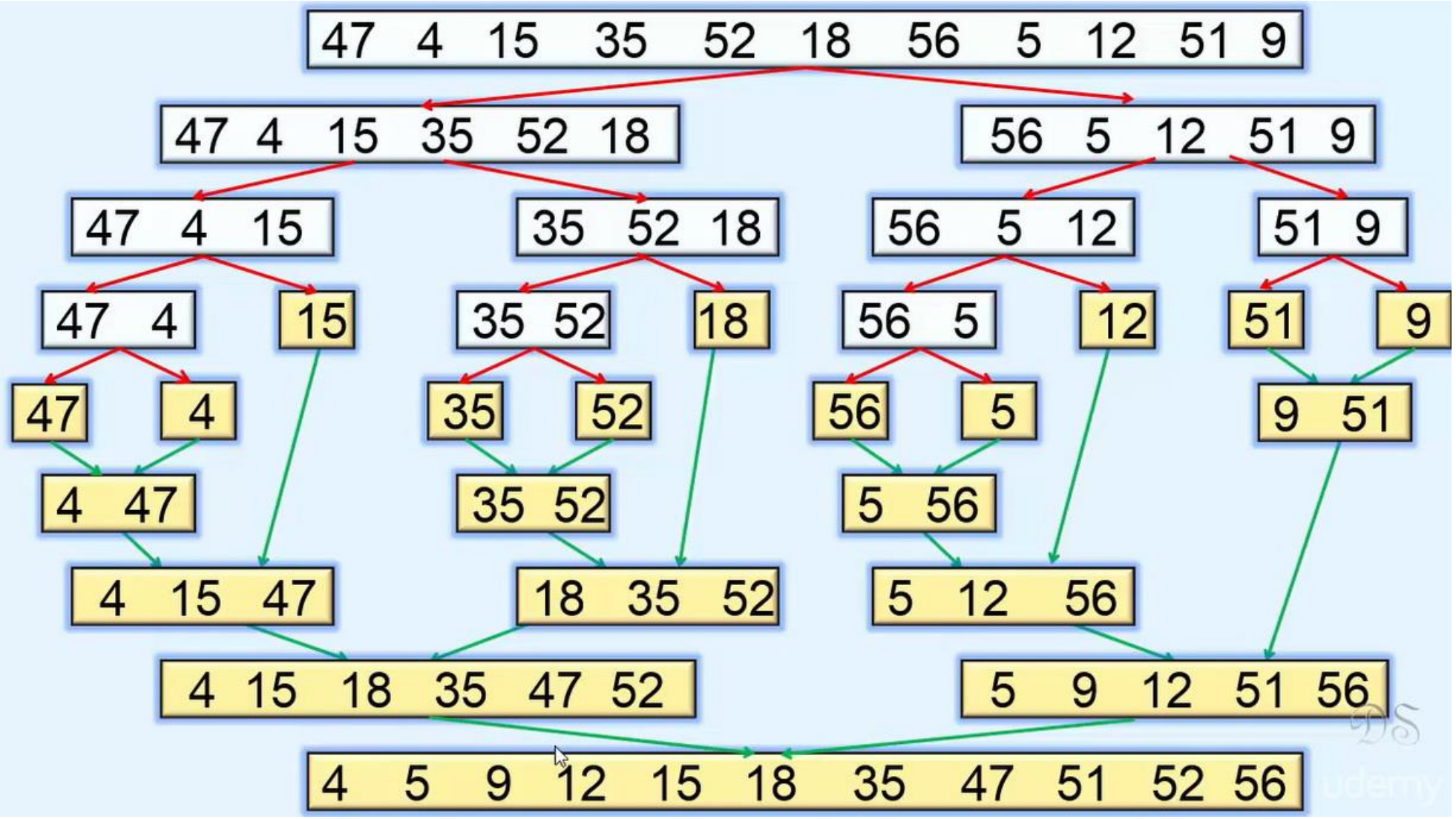
Algoritam Merge Sort





- 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()

```
public static void Merge(int[] a, int[] temp, int low1, int up1, int low2, int up2)
{
    int i = low1;
    int j = low2;
    int k = low1;

    while (i<=up1 && j<=up2)
    {
        if (a[i] <a[j])
        {
            temp[k++] = a[i++];
        }
        else
        {
            temp[k++] = a[j++];
        }
    }

    while (i <=up1)
    {
        temp[k++] = a[i++];
    }

    while (j <=up2)
    {
        temp[k++] = a[j++];
    }
}
```

- Od niza a kreira se niz temp
- low1, up1 – granice prvog podniza niza a
- low2, up2 –granice drugog podniza niza a

Funkcija za kopiranje dela niza temp[] u deo niza a[]

```
public static void Copy(int[] a, int[] temp, int low, int up)
{
    for (int i = low; i <= up; i++)
    {
        a[i] = temp[i];
    }
}
```

Rekurzivna funkcija Sort sa parametrima

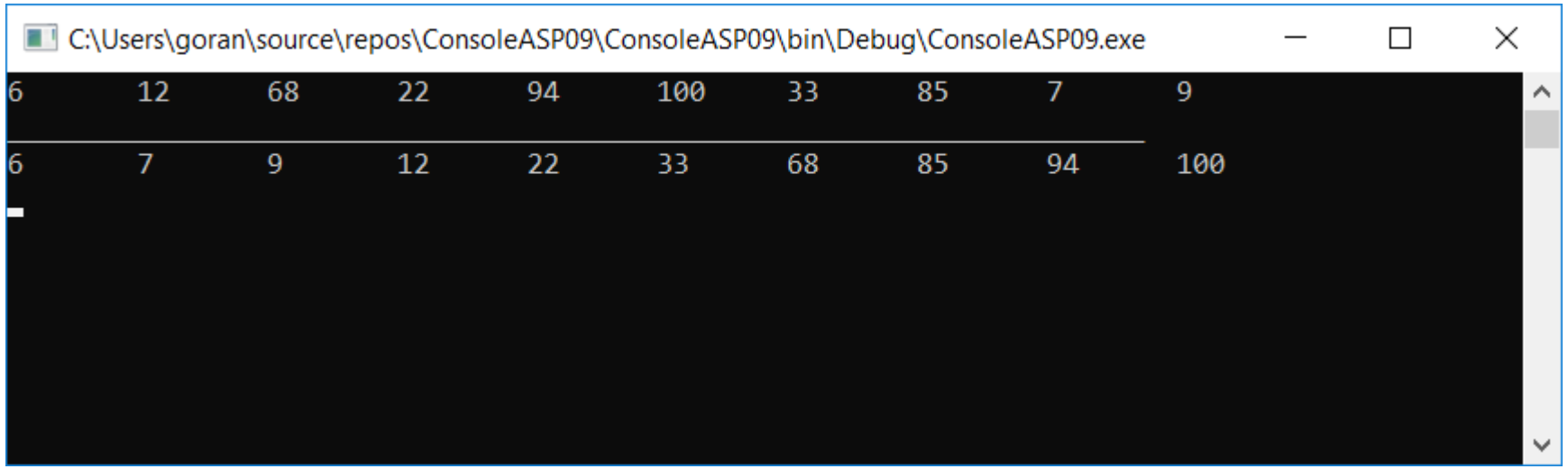
```
public static void Sort(int[] a, int[] temp, int low, int up)
{
    if (low == up)
    {
        // samo jedan element
        return;
    }

    int mid = (low + up) / 2;

    Sort(a, temp, low, mid);
    Sort(a, temp, mid + 1, up);

    Merge(a, temp, low, mid, mid + 1, up);
    Copy(a, temp, low, up);
}
```

Funkcija MergeSort()

```
public static void MergeSort(int[] a)
{
    int n = a.Length;

    int[] temp = new int[n];

    Sort(a, temp, 0, n - 1);
}
```

Poziv funkcije MergeSort()

```
static void Main(string[] args)
{
    int[] a = KreirajNiz(10);
    PisiNiz(a);
    Linija(70);
    MergeSort(a);
    PisiNiz(a);
    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

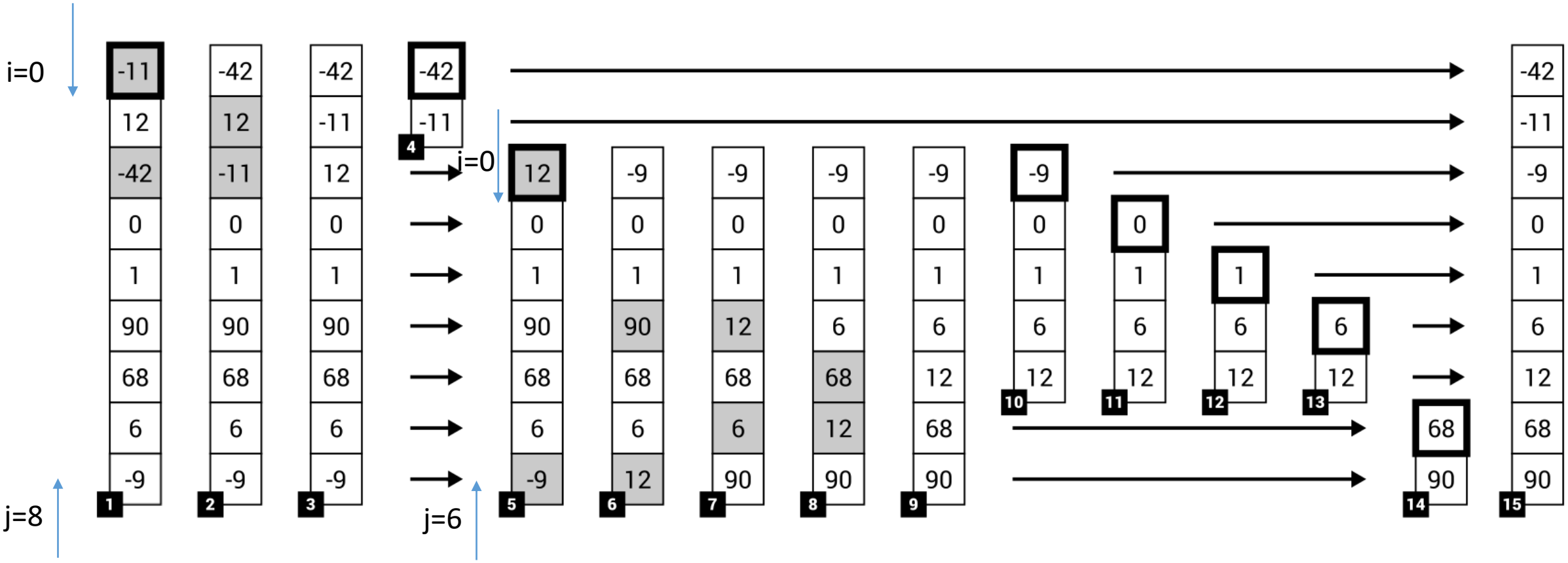
- Efikasan algoritam sortiranja
- Tehnika podeli pa vladaj
- Ceo niz a[] se deli na dve particije: levu i desnu
- Svi elementi leve particije su manji od elementa koji se zove pivot
- Svi elementi desne grupe su veći ili jednaki od pivot elementa
- Inicijalno se za pivot postavlja bilo koji element niza. npr prvi:
  pivot= a[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[] a, int levi, int desni)
```

- Neka je levi, levi indeks podniza koji treba da se podeli na particije
- Neka je desni, desni indeks podniza koji treba da se podeli na particije
- Inicijalno levi = 0, desni = n-1, kada se od celog niza kreiraju dve particije
- Da bi se ostvarila podela na grupe uvodi se:
  - indeks i inicijalizovan na vrednost levi koji služi za kretanje kroz niz sa leva na desno
  - indeks j inicijalizovan na vrednost desni za kretanje kroz niz sa desne na levu stranu
- Funkcija vraća tačnu poziciju pivot elementa nakon podele niza na particije
- Funkcija menja niz tako da su levo od tačne pozicije pivot elementa vrednosti manje od njega, a desno veće od njega

Funkcionisanje Quick Sort algoritma



Objasnjenje primera

- Brojač i ide sa leva na desno (na slici odozgo na dole) a brojč j ide sa desna na levo (na slici odozdo na gore)
- Tačka 1. pivot = -11
- Tačka 1. Brojač j: Pokazuje na prvi element koji je manji od pivota(-11), a to je element sa vrednošću (-42)
- Tačka 2. Elementi sa vrednošću (-11) i (-42) menjaju mesto
- Tačka 2. Brojač i: Pokazuje na prvi element koji je veći od pivota a to je 12
- Tačka 3. (12) i (-11) -pivot menjaju mesto i pivot se pozicionira na pravu poziciju tj. poziciju 1
- Svi elementi levo od (-11) su manji od njega a desno su veći od njega
- Tačka 4. leva particija je sortirana tj. njen pivot (-42) je na pravom mestu
- Tačka 5. Pretpostavlja se da je 12 pivot desne particije
- Tačka 5. Brojač j: pokazuje na prvi element koji je manji od pivota tj.(-9)
- Tačka 6. Brojevi (-9) i pivot=12 menjaju mesta
- Tačka 6. Brojač i: pokazuje na prvi element koji je veći od pivota a to je 90
- Tačka 7. Brojevi 12 i 90 menjaju mesto a brojac j pokazuje na prvi broj koji je manji od pivota =12 a to je 6.
- itd

Funkcija za particionisanje niza

```
static int Particija(int[] a, int levi, int desni)
{
    int i, j;
    i = levi;
    j = desni;
    int pivot = a[levi];
    while (true)
    {

        while (a[i] < pivot)
        {
            i++;
        }

        while (a[j] > pivot)
        {
            j--;
        }

        if (i < j)
        {
            if (a[i] == a[j])
            {
                return j;
            }

            int temp = a[i];
            a[i] = a[j];
            a[j] = temp;
        }
        else
        {
            return j;
        }
    }
}
```

Quick Sort algoritam

- Niz se rearanžira tako da se pivot pozicionira na odgovarajuće mesto
- Levi podniz se sortira rekurzivno primenom quick sort algoritma
- Desni podniz se sortira rekurzivno primenom quick sort algoritma
- Rekurzija se završava kada podniz sadrži jedan element ili je prazan

Funkcija za rekurzivno sortiranje

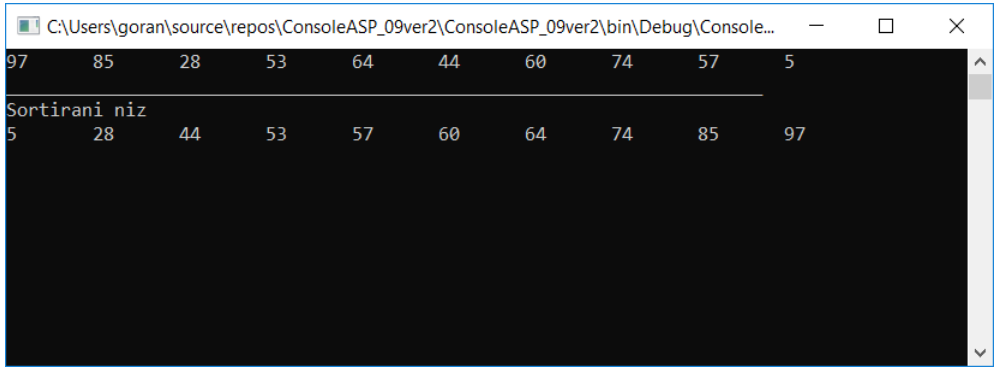
```
static void Sort(int[] a, int levi, int desni)
{
    if (levi >= desni)
    {
        return;
    }

    int p = Particija(a, levi, desni);
    if (p>1)
    {
        Sort(a, levi, p - 1); // sortiraj levi podniz
    }

    if (p+1 <desni)
    {
        Sort(a, p + 1, desni);  // sortiraj desni podniz
    }
}
```

Funkcija QuickSort()

```
static void QuickSort(int[] a)
{
    int n = a.Length;
    Sort(a, 0, n - 1);
}
```

Poziv funkcije QuickSort()

```
static void Main(string[] args)
{
    int[] a = KreirajNiz(10);
    PisiNiz(a);
    Linija(70);
    QuickSort(a);

    Console.WriteLine("Sortirani niz");
    PisiNiz(a);

    Console.ReadLine();
}
```



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