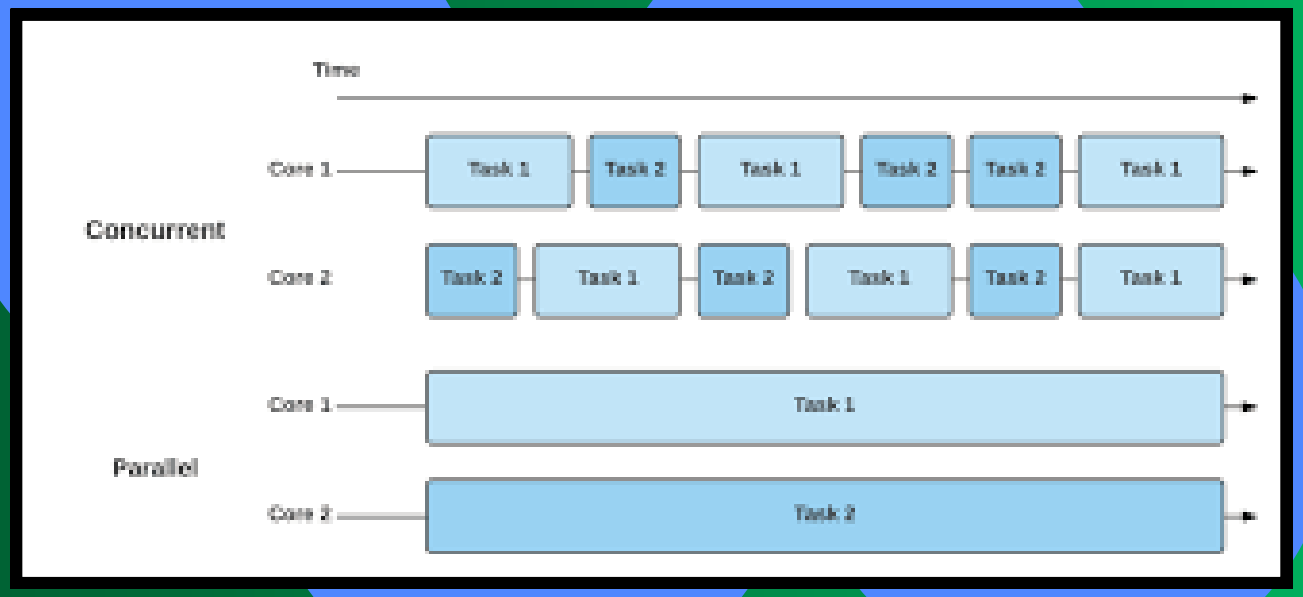
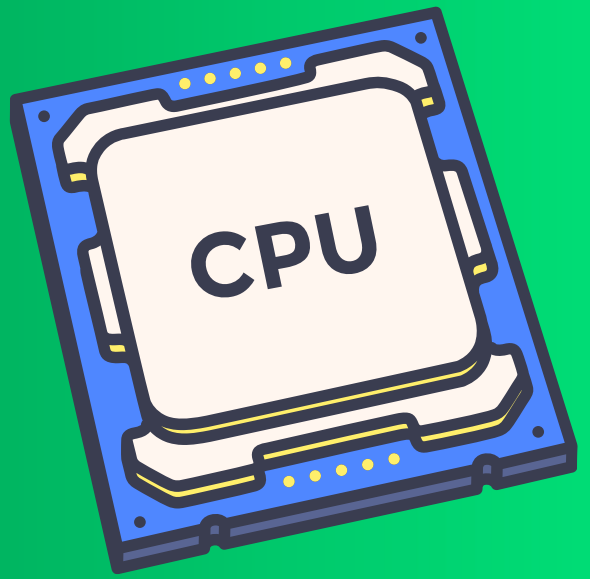
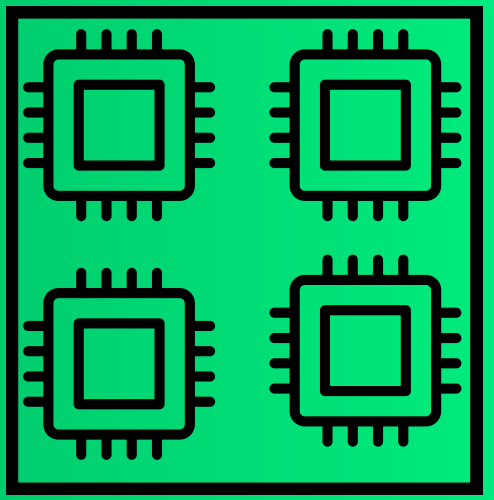
PARALELNI RACUNARSKI SISTEMI

PARALELIZAM U ALGORITMIMA SORTIRANJA



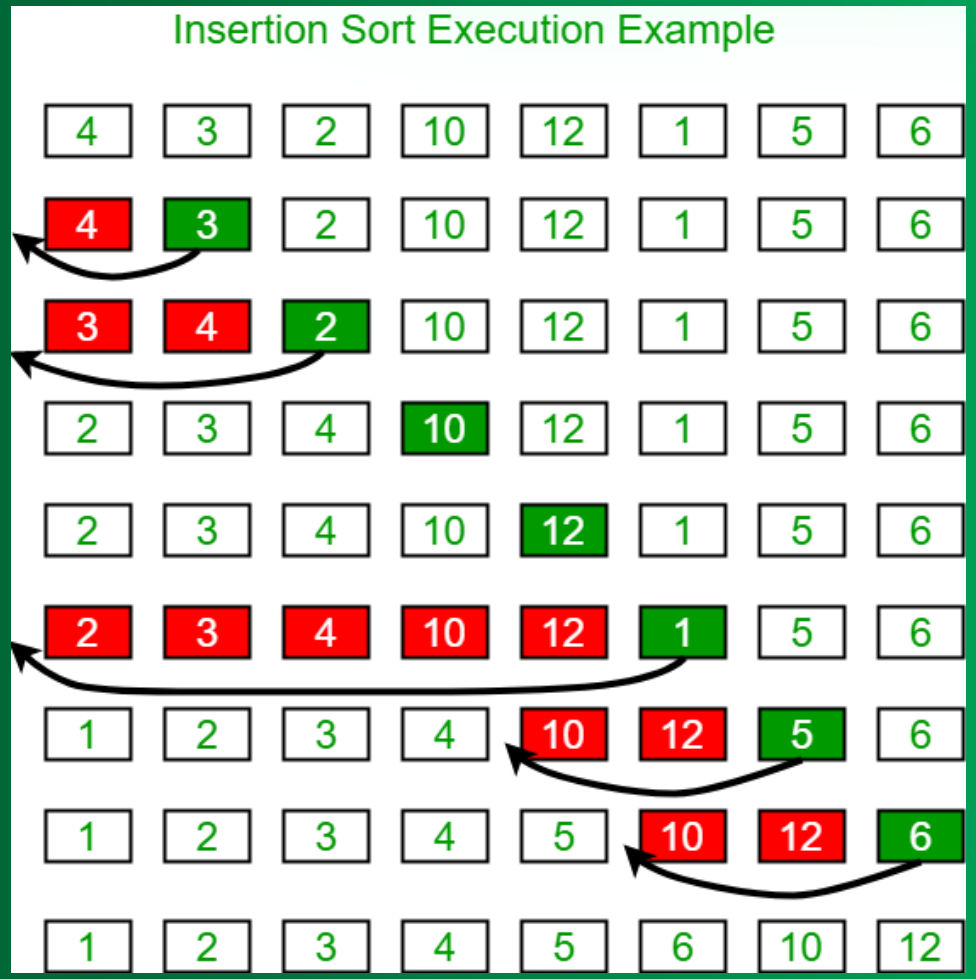




Paralelizam

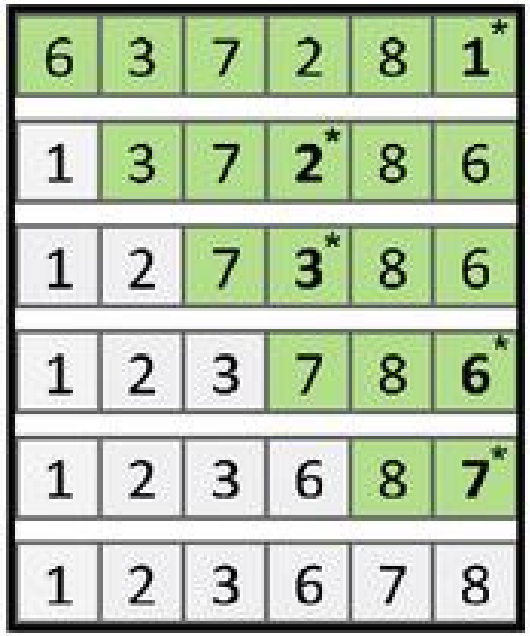
ALGORITMI SORTIRANJA

INSERTION SORT




```java
lic class InsertionSort {

 public static void insertionSort(int[] niz) {

     int i, j, priv;

     for(i=1;i<niz.length;i++){

         j=i;

         while(j>0 && niz[j-1]>niz[j]){

             priv=niz[j-1];
             niz[j-1]=niz[j];
             niz[j]=priv;
             j--;
         }
     }

 }
```

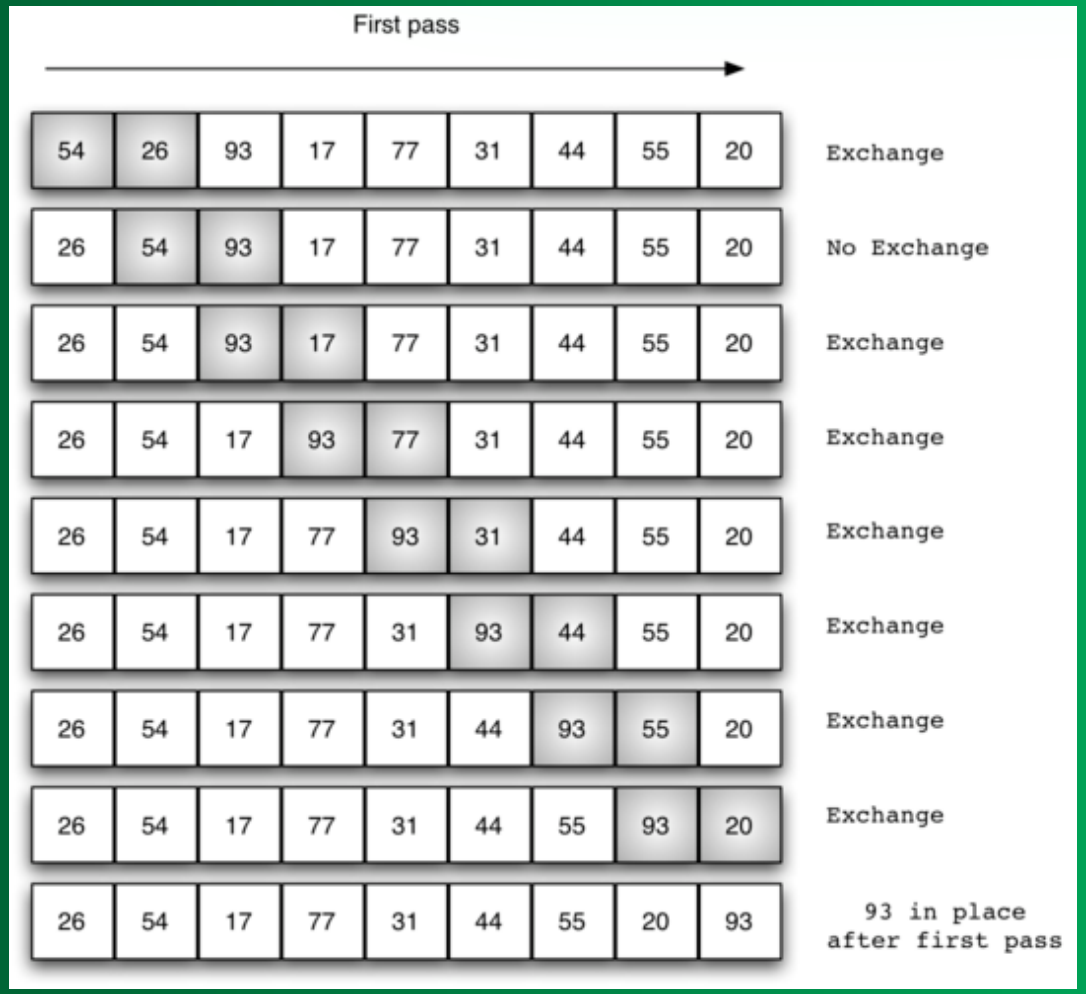
SELECTION SORT



6	3	7	2	8	1*
1	3	7	2*	8	6
1	2	7	3*	8	6
1	2	3	7	8	6*
1	2	3	6	8	7*
1	2	3	6	7	8

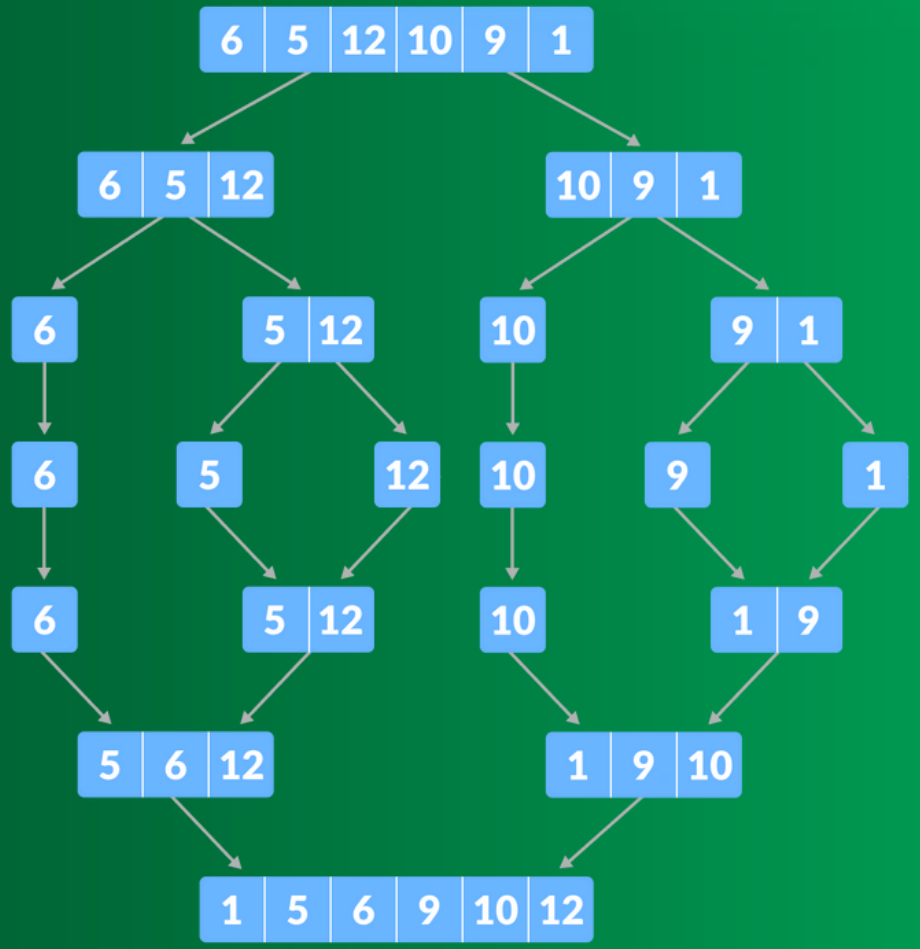
```java
public class SelectionSort {

    public static void selectionSort(int[] niz) {

        int i, min,j,pom;

        for(i=0;i<niz.length-1;i++){

            min=i;

            for(j=i+1;j<niz.length;j++){
                if(niz[j]<niz[min]){
                        min=j;
                        }
                    }

                pom=niz[min];
                niz[min] = niz[i];
                niz[i]=pom;
        }
    }
}
```

BUBBLE SORT



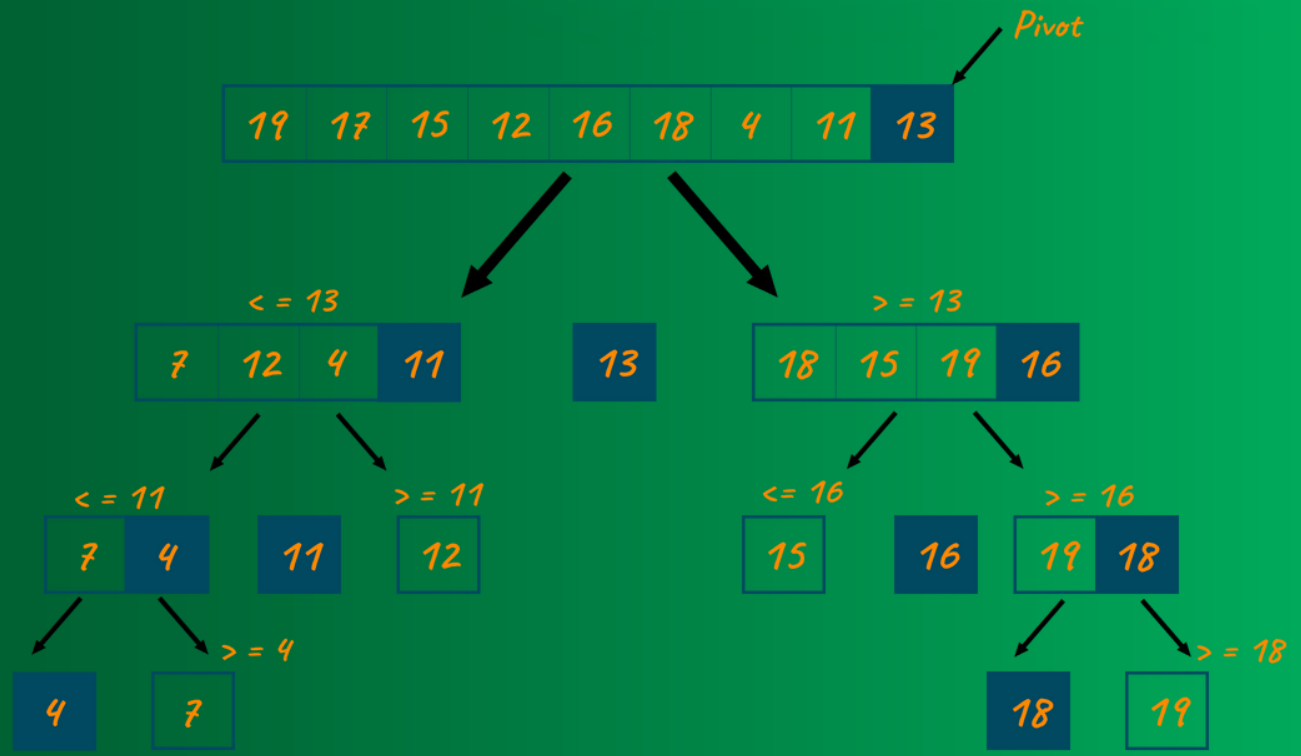

```
blic class BubbleSort {
  public static void bubbleSort(int[] niz) {

      int i, j, pom;
      for(i=1;i<niz.length;i++){
          for(j=0;j<niz.length-1;j++){

              if(niz[j]>niz[j+1]){

                  pom=niz[j];
                  niz[j] = niz[j+1];
                  niz[j+1]=pom;
                 }
          }
      }

  }
```

MERGE SORT




```
public static void mergeSort(int[] niz, int lijevi, int desni) {
if (lijevi < desni) {

int srednji = (lijevi + desni) / 2;

// Sortiramo prvu i drugu polovicu
mergeSort(niz, lijevi, srednji);
mergeSort(niz, srednji + 1, desni);

// Spajamo sortirane polovice
merge(niz, lijevi, srednji, desni);
}
}
```

QUICK SORT




```java
package algorithams;

public class QuickSort {

    public static void sort(int[] niz, int lijevi, int desni) {
        if (lijevi < desni) {
            int pivotIndex = partition(niz, lijevi, desni);
            sort(niz, lijevi, pivotIndex - 1);
            sort(niz, pivotIndex + 1, desni);
        }
    }

    private static int partition(int[] niz, int lijevi, int desni) {
        int pivot = niz[desni];
        int i = lijevi - 1;
        for (int j = lijevi; j < desni; j++) {
            if (niz[j] < pivot) {
                i++;
                int temp = niz[i];
                niz[i] = niz[j];
                niz[j] = temp;
            }
        }
        int temp = niz[i + 1];
        niz[i + 1] = niz[desni];
        niz[desni] = temp;
        return i + 1;
    }
}
```

PARALELNI ALGORITMI SORTIRANJA

SVAKI PARALELNO SORTIRANJE SE ZAPOČINJE NA OVAJ NAČIN

```
public static void parallelMergeSort(int[] niz, int lijevi, int desni) {
    SortTask task = new SortTask(niz, lijevi, desni);
    ForkJoinPool pool = new ForkJoinPool();
    pool.invoke(task);
}
```

Instanciramo task.
Dodijelimo ga pool-u
Okinemo ga pomoću invoke()

MERGESORT

SEKVENCIJALNI

```
if (lijevi < desni) {

    int srednji = (lijevi + desni) / 2;

    // Sortiramo prvu i drugu polovicu
    mergeSort(niz, lijevi, srednji);
    mergeSort(niz, srednji + 1, desni);

    // Spajamo sortirane polovice
    merge(niz, lijevi, srednji, desni);
}
```

AKO POREDIMO LINIJU PO LINIJU, OVI ALGORITMI VRLO SLIČE

PARALELNI

```
@Override
protected void compute() {

    if(lijevi < desni) {

        int srednji = (lijevi + desni) / 2;

        SortTask prvaPolovina = new SortTask(niz, lijevi, srednji);
        SortTask drugaPolovina = new SortTask(niz, srednji + 1, desni);

        invokeAll(prvaPolovina, drugaPolovina);

        MergeSort.merge(niz, lijevi, srednji, desni);
    }
}
```

QUICKSORT

SEKVENCIJALNI

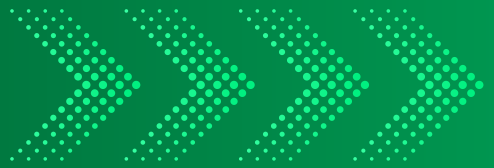
```java
@Override
public <T extends Comparable<T>> void sort(T[] niz, int lijevi, int desni) {
        if (lijevi < desni) {
            int pivotIndex = partition(niz, lijevi, desni);
            sort(niz, lijevi, pivotIndex - 1);
            sort(niz, pivotIndex + 1, desni);
        }
    }
```

PARALELNI

```java
@Override
protected void compute() {

    if(lijevi < desni) {

        int pivot = QuickSort.partition(niz, lijevi, desni);

        SortTask<?> lijevaStrana = new SortTask<T>(niz, lijevi, pivot - 1);
        SortTask<?> desnaStrana = new SortTask<T>(niz, pivot + 1, desni);

        invokeAll(lijevaStrana, desnaStrana);
    }
}
```

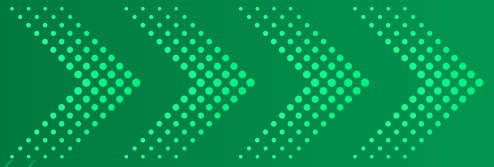
SELECTIONSORT

SEKVENCIJALNI



ELSE

PARALELNI



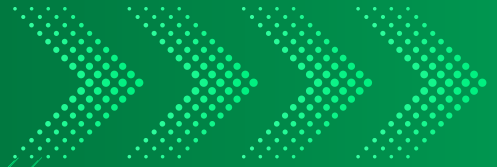
```java
@Override
protected void compute() {
    if(desni - lijevi <= GRANICA) {
        //Sortiranje podniza selection sortom
         for (int i = lijevi; i < desni; i++) {
             int min = i;
               for(int j = i + 1; j < desni + 1; j++) {
                   if(niz[j].compareTo(niz[min]) < 0)
                       min = j;
               }

               T pom = niz[min];
               niz[min] = niz[i];
               niz[i] = pom;
           }
    }
    else {
        //Podjela zadatka na podzadatke
        int srednji = lijevi + (desni - lijevi) / 2;

        SortTask<?> prvaPolovina = new SortTask<T>(niz, lijevi, srednji);
        SortTask<?> drugaPolovina = new SortTask<T>(niz, srednji + 1, desni);

        invokeAll(prvaPolovina, drugaPolovina);
        //Spajanje sortiranih nizova
        MergeSort.merge(niz, lijevi, srednji, desni);
    }

}
```

INSERTIONSORT

SEKVENCIJALNI

ELSE

PARALELNI

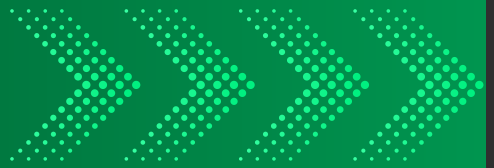


```
@Override
protected void compute() {
    if(desni - lijevi <= GRANICA) {

        //Sortiranje podniza insertion sortom
         for (int i = lijevi + 1; i < desni + 1; i++) {

             int j = i;

             while(j > lijevi && niz[j - 1].compareTo(niz[j]) > 0) {
                   T pom = niz[j - 1];
                   niz[j - 1] = niz[j];
                   niz[j] = pom;
                   j--;
                }
            }
    }
    else {
        //Podjela zadatka na podzadatke
        int srednji = lijevi + (desni - lijevi) / 2;

        SortTask<?> prvaPolovina = new SortTask<T>(niz, lijevi, srednji);
        SortTask<?> drugaPolovina = new SortTask<T>(niz, srednji + 1, desni);

        invokeAll(prvaPolovina, drugaPolovina);
        //Spajanje sortiranih nizova
        MergeSort.merge(niz, lijevi, srednji, desni);
    }

}
```

BUBBLESORT

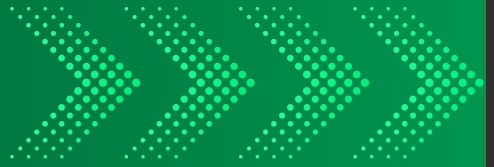
Dijeljenje nizova na cjeline, te njihovo ponovno spajanje isto je kao kod Mergesorta.

SEKVENCIJALNI



ELSE

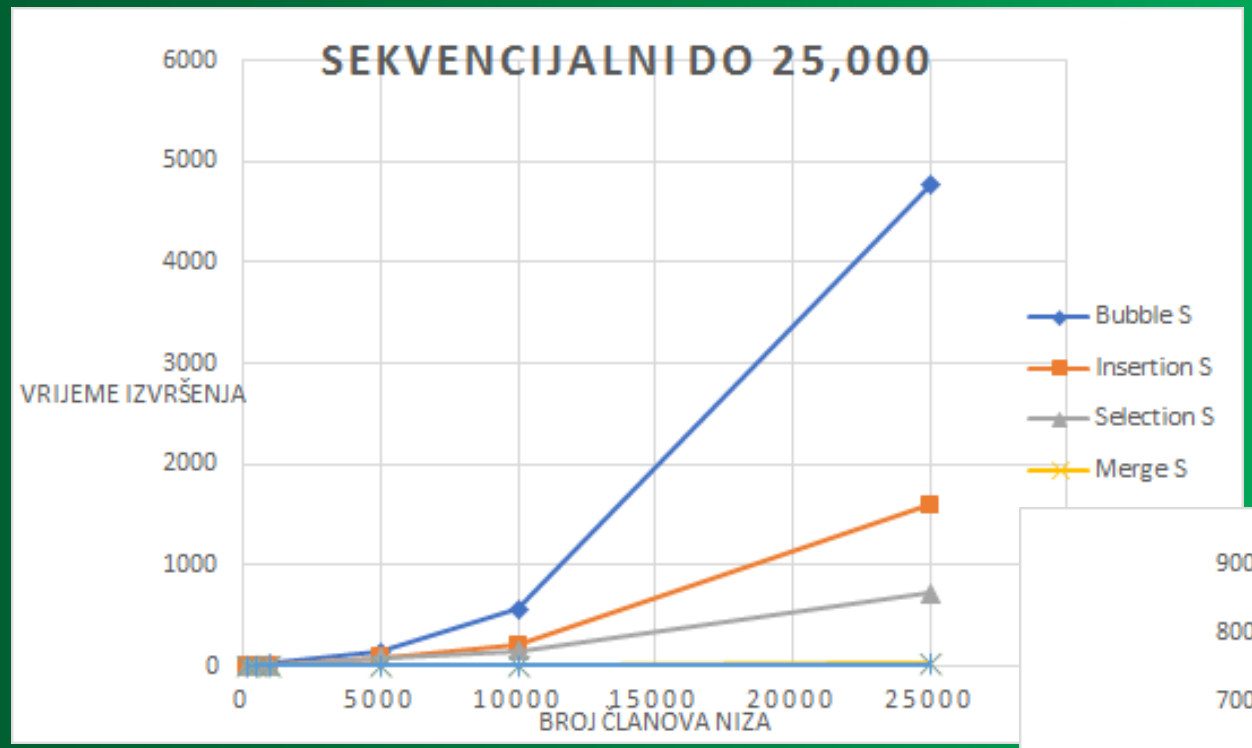
PARALELNI



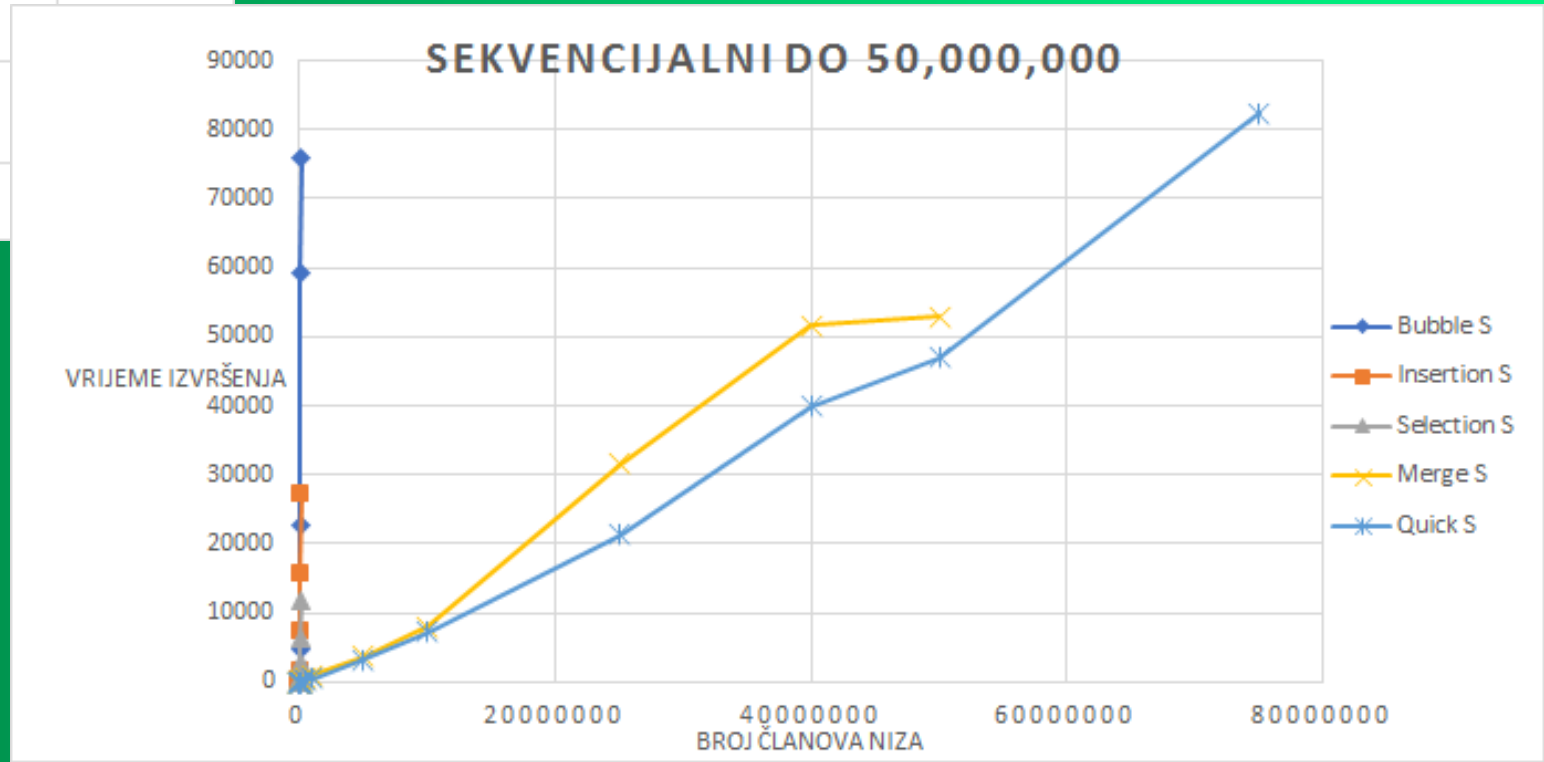
Možemo reći da su zadnja tri algoritma ustvari grublje granulisani i preprocesirani Mergesort!

```
@Override
protected void compute() {
    if(desni - lijevi <= GRANICA) {
        //Sortiranje podniza bubble sortom
         for (int i = lijevi + 1; i < desni + 1; i++) {
                for (int j = lijevi; j < desni; j++) {

                    if (niz[j].compareTo(niz[j+1]) > 0) {
                     T temp = niz[j];
                     niz[j] = niz[j + 1];
                     niz[j + 1] = temp;
                    }
                }
            }
    }
    else {
        //Podjela zadatka na podzadatke
        int srednji = lijevi + (desni - lijevi) / 2;

        SortTask<?> prvaPolovina = new SortTask<T>(niz, lijevi, srednji);
        SortTask<?> drugaPolovina = new SortTask<T>(niz, srednji + 1, desni);

        invokeAll(prvaPolovina, drugaPolovina);
        //Spajanje sortiranih nizova
        MergeSort.merge(niz, lijevi, srednji, desni);
    }

}
```

TESTIRANJE

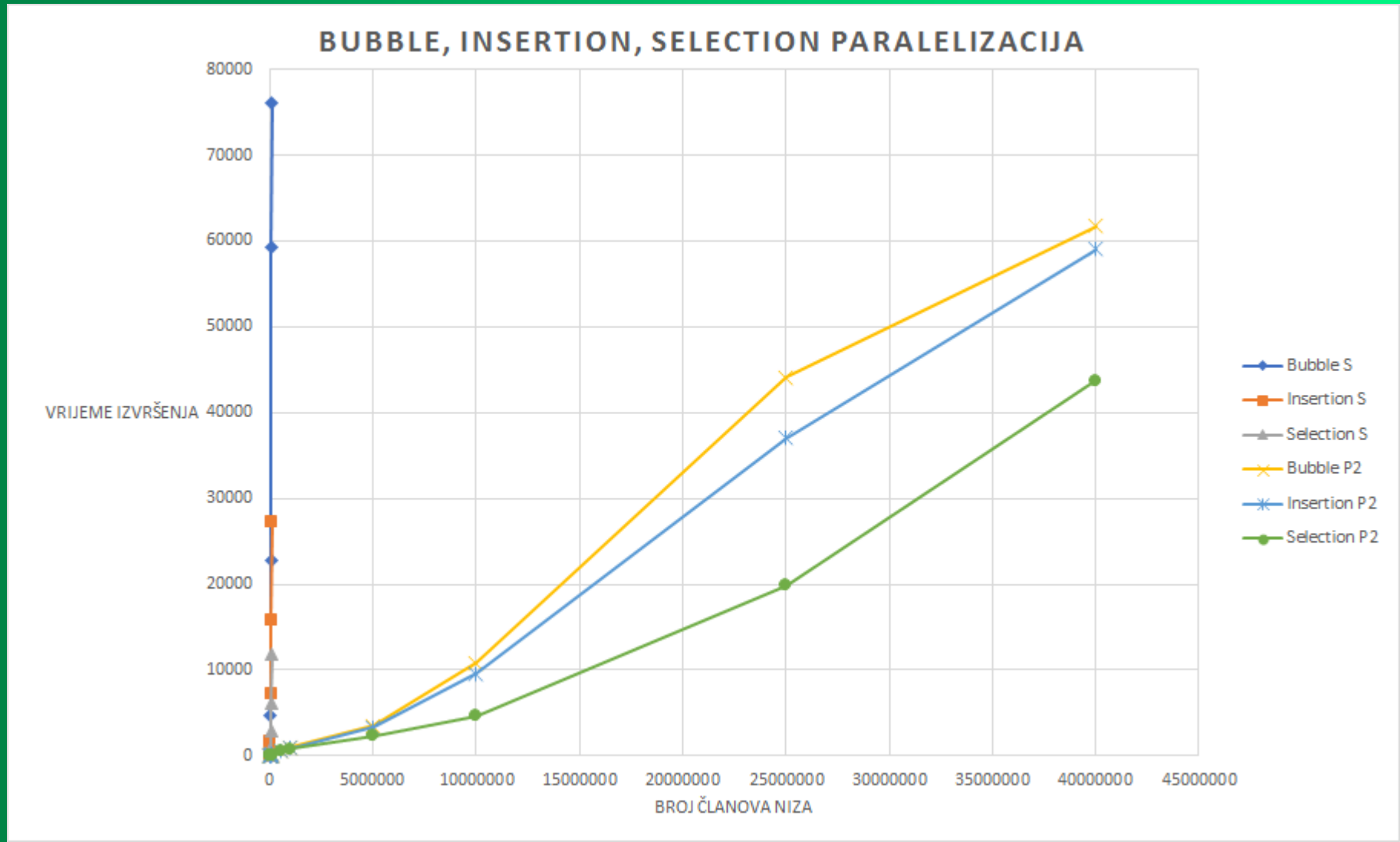


Merge i quicksort su daleko brži sa veće nizove.

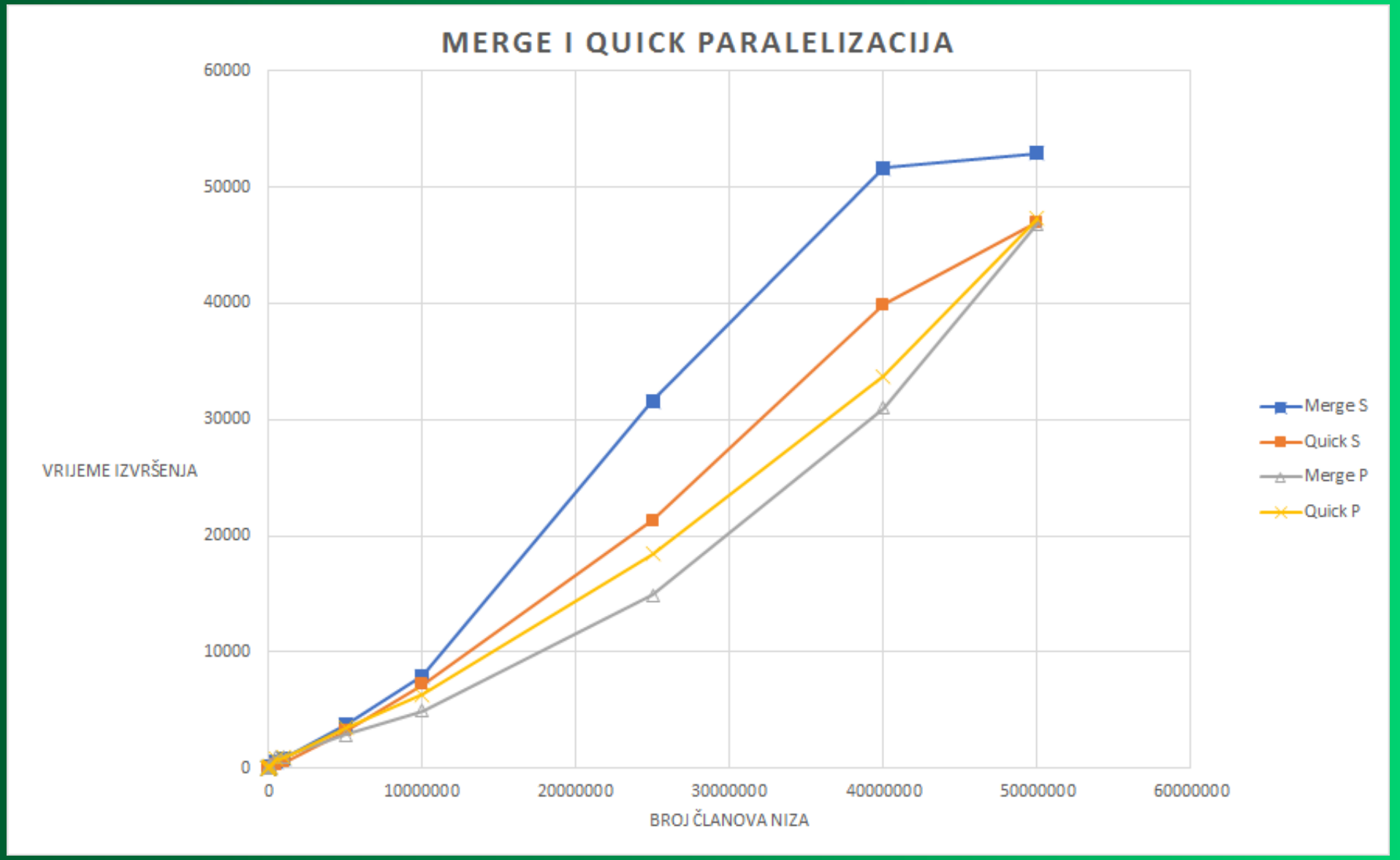


PARALELIZACIJA N^2 ALGORITAMA

Slično, za ova tri algoritma, postignute su daleko bolje performanse.



PARALELIZACIJA NLOGN ALGORITAMA



Za ove algoritme primjećujemo otprilike linearno smanjenje krivih, što je za očekivati.