

```

package LinearneStrukture;

/**
 * Sadrži algoritme za pretraživanje niza celih brojeva.
 *
 * @author Kostandin Dimitrijević, Strukture podataka i algoritmi, FON, 2010
 */
public class Pretrazivanje {

    /**
     * Sekvencijalno prtetraživanje niza celih brojeva.
     * Algoritam prolazi kroz ceo niz od početka do kraja i upoređuje
     * trenutni element sa elementom koji se traži.
     * Ako je trenutni element onaj koji se traži, vraća indeks elementa.
     * Ako nije, prelazi na sledeći element.
     *
     * @param podatak Element koji se traži
     * @param n Niz za pretraživanje.
     * @return Indeks traženog elementa ili -1 ukoliko traženi element ne postoji u nizu
     */
    public static int sekvencijalno(int podatak, int[] n){
        for (int i = 0; i < n.length; i++) {
            if (n[i] == podatak)
                return i;
        }
        // element nije pronađen
        return -1;
    }

    /**
     * Rekurzivna varijanta sekvencijalnog pretraživanja.
     */
    public static int sekvencijalnoR(int podatak, int[] n, int i){
        if (i > -1 && i < n.length){
            if (n[i] == podatak)
                return i;
            return Pretrazivanje.sekvencijalnoR(podatak, n, ++i);
        }
        return -1;
    }

    /**
     * Algoritam za binarno pretraživanje niza celih brojeva
     * sortiranog u rastićem redosledu.
     * Pretražuje segment niza tako što upoređuje element koji se traži
     * sa elementom na sredini segmenta.
     *
     * @param podatak Element koji se traži
     * @param n Niz za pretraživanje.
     * @return Indeks traženog elementa ili -1 ukoliko traženi element ne postoji u nizu
     */
    public static int binarno(int podatak, int[] n){
        // leva granica intervala. Na početku je to indeks prvog elementa.
        int l = 0;

        // desna granica intervala. Na početku je to indeks poslednjeg elementa.
        int d = n.length - 1;

        // ako element ne postoji u nizu, u nekom trenutku će se desiti
        // da je desna granica intervala manja od leve,
        // pa tada treba prekinuti algoritam
        while(l <= d){
            // sredina intervala

```

```

        int s = (l + d)/2;
        if (n[s] == podatak)
            return s; // pronađen je element
        if (n[s] > podatak)
            // element koji tražimo se možda nalazi u levoj polovini intervala,
            // pa zato pomeramo desnu granicu
            d = s - 1;
        else
            // element koji tražimo se možda nalazi u desnoj polovini intervala,
            // pa zato pomeramo levu granicu
            l = s + 1;
    }
    // element nije pronađen
    return -1;
}

/**
 * Rekurzivna varijanta binarnog pretraživanja.
 */
public static int binarnoR(int podatak, int[] n, int l, int d){
    if (l <= d){
        int s = (l + d)/2;
        if (n[s] == podatak)
            return s;
        if (n[s] > podatak)
            return binarnoR(podatak, n, l, s - 1);
        else
            return binarnoR(podatak, n, s + 1, d);
    }
    return -1;
}

/**
 * Algoritam za interpolaciono pretraživanje niza celih brojeva
 * sortiranog u rastućem redosledu.
 * Pretražuje segment niza tako što izračuna ("predvidi") indeks
 * gde bi element trebalo da se nađe i zatim proverava da li se
 * element koji se traži nalazi na tom mestu.
 *
 * @param podatak Element koji se traži.
 * @param n Niz za pretraživanje.
 * @return Indeks traženog elementa ili -1 ukoliko traženi element ne postoji u nizu.
 */
public static int interpolaciono(int podatak, int[] n){
    // leva granica intervala. Na početku je to indeks prvog elementa.
    int l = 0;

    // desna granica intervala. Na početku je to indeks poslednjeg elementa.
    int d = n.length - 1;

    // ako element ne postoji u nizu, u nekom trenutku će se desiti
    // da je desna granica intervala manja od leve,
    // pa tada treba prekinuti algoritam
    while(l <= d){
        // formula za izračunavanje mesta gde bi u segmentu
        // trebalo da se nalazi element koji tražimo
        int index = l + (d - l)/(n[d] - n[l])*(podatak - n[l]);
        if (n[index] == podatak)
            return index;
        if (n[index] > podatak)
            // element koji tražimo se možda nalazi u levoj polovini intervala,
            // pa zato pomeramo desnu granicu
            d = index - 1;
    }

```

```

        else
            // element koji tražimo se možda nalazi u desnoj polovini intervala,
            // pa zato pomeramo levu granicu
            l = index + 1;
        }
        return -1;
    }

/**
 * Rekurzivna varijanta interpolacionog pretraživanja.
 */
public static int interpolacionoR(int podatak, int[] n, int l, int d){
    if (l <= d){
        int index = l + (d - l)/(n[d] - n[l])*(podatak - n[l]);
        if (n[index] == podatak)
            return index;
        if (n[index] > podatak)
            return interpolacionoR(podatak, n, l, index - 1);
        else
            return interpolacionoR(podatak, n, index + 1, d);
    }
    return -1;
}
}

```