Pretraga i sortiranje – zadaci

Argumenti komandne linije, generatori slucajnih brojeva, interpolaciona pretraga

1.

Napisati iterativne funkcije za pretragu nizova. Svaka funkcija treba da vrati indeks pozicije na kojoj je pronađen traženi broj ili vrednost −1 ukoliko broj nije pronađen.

(a) Napisati funkciju linarna_pretraga koja vrši linearnu pretragu niza celih brojeva a, dužine n, tražeći u njemu broj x.

(b) Napisati funkciju binarna_pretraga koja vrši binarnu pretragu sortiranog niza a, dužine n, tražeći u njemu broj x.

(c) Napisati funkciju interpolaciona_pretraga koja vrši interpolacionu pretragu sortiranog niza a, dužine n, tražeći u njemu broj x.

Napisati i program koji generiše rastući niz slučajnih brojeva dimenzije n i pozivajući napisane funkcije traži broj x. Programu se kao prvi argument komandne linije prosleđuje prirodan broj n koji nije veći od 1000000 i broj x kao drugi argument komandne linije.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX 1000000

/* Naredne tri funkcije vrse pretragu. Ukoliko se trazeni element
   pronadje u nizu, one vracaju indeks pozicije na kojoj je element
   pronadjen. Ovaj indeks je uvek nenegativan. Ako element nije
   pronadjen u nizu, funkcije vracaju negativnu vrednost -1, kao
   indikator neuspesne pretrage. */

/* Linearna pretraga: Funkcija pretrazuje niz a[] celih brojeva
   duzine n, trazeci u njemu prvo pojavljivanje elementa x.
   Pretraga se vrsi prostom iteracijom kroz niz. */
int linearna_pretraga(int a[], int n, int x)
{
  int i;
  for (i = 0; i < n; i++)
```

```
    if (a[i] == x)
      return i;
  return -1;
}

/* Binarna pretraga: Funkcija trazi u sortiranom nizu a[] duzine n
   broj x. Pretraga koristi osobinu sortiranosti niza i u svakoj
   iteraciji polovi interval pretrage. */
int binarna_pretraga(int a[], int n, int x)
{
  int levi = 0;
  int desni = n - 1;
  int srednji;
  /* Dokle god je indeks levi levo od indeksa desni */
  while (levi <= desni) {
    /* Srednji indeks je njihova aritmeticka sredina */
    srednji = (levi + desni) / 2;
    /* Ako je element sa sredisnjim indeksom veci od x, tada se x
       mora nalaziti u levom delu niza */
    if (x < a[srednji])
      desni = srednji - 1;
    /* Ako je element sa sredisnjim indeksom manji od x, tada se x
       mora nalaziti u desnom delu niza */
    else if (x > a[srednji])
      levi = srednji + 1;
    else
      /* Ako je element sa sredisnjim indeksom jednak x, tada je
         broj x pronadjen na poziciji srednji */
      return srednji;
  }
  /* Ako element x nije pronadjen, vraca se -1 */
  return -1;
}

/* Interpolaciona pretraga: Funkcija trazi u sortiranom nizu a[]
   duzine n broj x. Pretraga koristi osobinu sortiranosti niza i
   zasniva se na linearnoj interpolaciji vrednosti koja se trazi
```

```
   vrednostima na krajevima prostora pretrage. */
int interpolaciona_pretraga(int a[], int n, int x)
{
  int levi = 0;
  int desni = n - 1;
  int srednji;
  /* Dokle god je indeks levi levo od indeksa desni...  */
  while (levi <= desni) {
    /* Ako je trazeni element manji od pocetnog ili veci od
       poslednjeg elementa u delu niza a[levi],...,a[desni], tada
       on nije u tom delu niza. Ova provera je neophodna, da se ne
       bi dogodilo da se prilikom izracunavanja indeksa srednji
       izadje izvan opsega indeksa [levi,desni] */
    if (x < a[levi] || x > a[desni])
      return -1;
    /* U suprotnom, x je izmedju a[levi] i a[desni], pa ako su
       a[levi] i a[desni] jednaki, tada je jasno da je trazeni broj
       x jednak ovim vrednostima, pa se vraca indeks levi (ili
       indeks desni). Ova provera je neophodna, jer bi se u
       suprotnom prilikom izracunavanja indeksa srednji pojavilo
       deljenje nulom. */
    else if (a[levi] == a[desni])
      return levi;
    /* Racuna se sredisnji indeks */
    srednji =
        levi +
        (int) ((double) (x - a[levi]) / (a[desni] - a[levi]) *
              (desni - levi));
    /* Napomena: Indeks srednji je uvek izmedju levi i desni, ali
       ce verovatno biti blize trazenoj vrednosti nego da je prosto
       uvek uzimana aritmiticka sredina indeksa levi i desni. Ovo
       se moze porediti sa pretragom recnika: ako neko trazi rec na
       slovo 'B', sigurno nece da otvori recnik na polovini, vec
       verovatno negde blize pocetku. */
    /* Ako je element sa indeksom srednji veci od trazenog, tada se
       trazeni element mora nalaziti u levoj polovini niza */
    if (x < a[srednji])
```

```
      desni = srednji - 1;
    /* Ako je element sa indeksom srednji manji od trazenog, tada
       se trazeni element mora nalaziti u desnoj polovini niza */
    else if (x > a[srednji])
      levi = srednji + 1;
    else
      /* Ako je element sa indeksom srednji jednak trazenom, onda
         se pretraga zavrsava na poziciji srednji */
      return srednji;
  }
  /* U slucaju neuspesne pretrage vraca se -1 */
  return -1;
}

int a[MAX];

int main(int argc, char **argv)
{

  int n, i, x;
//  struct timespec vreme1, vreme2, vreme3, vreme4, vreme5, vreme6;

  /* Provera argumenata komandne linije */
  if (argc != 3) {
    fprintf(stderr,
            "Greska: Program se poziva sa %s dim_niza broj\n",
            argv[0]);
    exit(EXIT_FAILURE);
  }

  /* Dimenzija niza */
  n = atoi(argv[1]);
  if (n > MAX || n <= 0) {
    fprintf(stderr, "Greska: Dimenzija niza neodgovarajuca\n");
    exit(EXIT_FAILURE);
  }
```

```
/* Broj koji se trazi */
x = atoi(argv[2]);
/* Elementi niza se generisu slucajno, tako da je svaki sledeci
   veci od prethodnog. Funkcija srand() inicijalizuje pocetnu
   vrednost sa kojom se krece u izracunavanje sekvence
   pseudo-slucajnih brojeva. Kako generisani niz ne bi uvek bio
   isti, ova vrednost se postavlja na tekuce vreme u sekundama od
   Nove godine 1970, tako da je za svako sledece pokretanje
   programa (u vremenskim intervalima vecim od jedne sekunde) ove
   vrednost drugacija. rand()%100 vraca brojeve izmedju 0 i 99 */
srand(time(NULL));
for (i = 0; i < n; i++)
  a[i] = i == 0 ? rand() % 100 : a[i - 1] + rand() % 100;
/* Lineara pretraga */
printf("Linearna pretraga:\n");
/* Vreme proteklo od Nove godine 1970 */
//clock_gettime(CLOCK_REALTIME, &vreme1);
i = linearna_pretraga(a, n, x);
/* Novo vreme i razlika sa prvim predstavlja vreme utroseno za
   linearnu pretragu */
// clock_gettime(CLOCK_REALTIME, &vreme2);
if (i == -1)
  printf("Element nije u nizu\n");
else
  printf("Element je u nizu na poziciji %d\n", i);
/* Binarna pretraga */
printf("Binarna pretraga:\n");
//clock_gettime(CLOCK_REALTIME, &vreme3);
i = binarna_pretraga(a, n, x);
//clock_gettime(CLOCK_REALTIME, &vreme4);
if (i == -1)
  printf("Element nije u nizu\n");
else
  printf("Element je u nizu na poziciji %d\n", i);
/* Interpolaciona pretraga */
printf("Interpolaciona pretraga:\n");
//clock_gettime(CLOCK_REALTIME, &vreme5);
```

```
 i = interpolaciona_pretraga(a, n, x);
 //clock_gettime(CLOCK_REALTIME, &vreme6);
 if (i == -1)
   printf("Element nije u nizu\n");
 else
   printf("Element je u nizu na poziciji %d\n", i);


 exit(EXIT_SUCCESS);
}
```

2.

Napraviti biblioteku koja implementira algoritme sortiranja nizova celih brojeva. Biblioteka treba da sadrži algoritam sortiranja izborom (engl. selection sort), sortiranja spajanjem (engl. merge sort), brzog sortiranja (engl. quick sort), mehurastog sortiranja (engl. bubble sort), sortiranja direktnim umetanjem (engl. insertion sort) i sortiranja umetanjem sa inkrementom (engl. shell sort). Upotrebiti biblioteku kako bi se napravilo pozivanje različitih algoritama sortiranja.

SORT.H

SORT.C

MAIN.C

SORT.H

```
#ifndef _SORT_H_
#define _SORT_H_ 1

/* Selection sort: Funkcija sortira niz celih brojeva metodom
   sortiranja izborom.  Ideja algoritma je sledeca: U svakoj
   iteraciji pronalazi se najmanji element i premesta se na pocetak
   niza. Dakle, u prvoj iteraciji, pronalazi se najmanji element, i
   dovodi na nulto mesto u nizu. U i-toj iteraciji najmanjih i-1
   elemenata su vec na svojim pozicijama, pa se od elemenata sa
   indeksima od i do n-1 trazi najmanji, koji se dovodi na i-tu
   poziciju. */
void selection_sort(int a[], int n);

/* Insertion sort: Funkcija sortira niz celih brojeva metodom
   sortiranja umetanjem.  Ideja algoritma je sledeca: neka je na
   pocetku i-te iteracije niz prvih i elemenata
   (a[0],a[1],...,a[i-1]) sortirano. U i-toj iteraciji treba
   element a[i] umetnuti na pravu poziciju medju prvih i elemenata
   tako da se dobije niz duzine i+1 koji je sortiran. Ovo se radi
```

```
   tako sto se i-ti element najpre uporedi sa njegovim prvim levim
   susedom (a[i-1]). Ako je a[i] vece, tada je on vec na pravom
   mestu, i niz a[0],a[1],...,a[i] je sortiran, pa se moze preci na
   sledecu iteraciju. Ako je a[i-1] vece, tada se zamenjuju a[i] i
   a[i-1], a zatim se proverava da li je potrebno dalje
   potiskivanje elementa u levo, poredeci ga sa njegovim novim
   levim susedom. Ovim uzastopnim premestanjem se a[i] umece na
   pravo mesto u nizu. */
void insertion_sort(int a[], int n);

/* Bubble sort: Funkcija sortira niz celih brojeva metodom
   mehurica. Ideja algoritma je sledeca: prolazi se kroz niz redom
   poredeci susedne elemente, i pri tom ih zamenjujuci ako su u
   pogresnom poretku.  Ovim se najveci element poput mehurica
   istiskuje na "povrsinu", tj. na krajnju desnu poziciju. Nakon
   toga je potrebno ovaj postupak ponoviti nad nizom
   a[0],...,a[n-2], tj. nad prvih n-1 elemenata niza bez poslednjeg
   koji je postavljen na pravu poziciju. Nakon toga se isti
   postupak ponavlja nad sve kracim i kracim prefiksima niza, cime
   se jedan po jedan istiskuju elementi na svoje prave pozicije. */
void bubble_sort(int a[], int n);

/* Selsort: Ovaj algoritam je jednostavno prosirenje sortiranja
   umetanjem koje dopusta direktnu razmenu udaljenih elemenata.
   Prosirenje se sastoji u tome da se kroz algoritam umetanja
   prolazi vise puta. U prvom prolazu, umesto koraka 1 uzima se
   neki korak h koji je manji od n (sto omogucuje razmenu udaljenih
   elemenata) i tako se dobija h-sortiran niz, tj. niz u kome su
   elementi na rastojanju h sortirani, mada susedni elementi to ne
   moraju biti. U drugom prolazu kroz isti algoritam sprovodi se
   isti postupak ali za manji korak h. Sa prolazima se nastavlja
   sve do koraka h = 1, u kome se dobija potpuno sortirani niz.
   Izbor pocetne vrednosti za h, i nacina njegovog smanjivanja
   menja u nekim slucajevima brzinu algoritma, ali bilo koja
   vrednost ce rezultovati ispravnim sortiranjem, pod uslovom da je
   u poslednjoj iteraciji h imalo vrednost 1. */
void shell_sort(int a[], int n);

/* Merge sort: Funkcija sortira niz celih brojeva a[]
   ucesljavanjem. Sortiranje se vrsi od elementa na poziciji l do
   onog na poziciji d. Na pocetku, da bi niz bio kompletno
   sortiran, l mora biti 0, a d je jednako poslednjem validnom
   indeksu u nizu. Funkcija niz podeli na dve polovine, levu i
   desnu, koje zatim rekurzivno sortira. Od ova dva sortirana
   podniza, sortiran niz se dobija ucesljavanjem, tj. istovremenim
   prolaskom kroz oba niza i izborom trenutnog manjeg elementa koji
   se smesta u pomocni niz. Na kraju algoritma, sortirani elementi
   su u pomocnom nizu, koji se kopira u originalni niz. */
void merge_sort(int a[], int l, int d);

/* Quick sort: Funkcija sortira deo niza brojeva a izmedju pozicija
   l i d. Njena ideja sortiranja je izbor jednog elementa niza, koji
   se naziva pivot, i koji se dovodi na svoje mesto. Posle ovog
   koraka, svi elementi levo od njega bice manji, a svi desno bice
   veci od njega. Kako je pivot doveden na svoje mesto, da bi niz
   bio kompletno sortiran, potrebno je sortirati elemente levo
   (manje) od njega, i elemente desno (vece). Kako su dimenzije ova
   dva podniza manje od dimenzije pocetnog niza koji je trebalo
   sortirati, ovaj deo moze se uraditi rekurzivno. */
void quick_sort(int a[], int l, int d);

#endif
```

SORT.C

```
#include "sort.h"

#define MAX 1000000

void selection_sort(int a[], int n)
{
  int i, j;
  int min;
  int pom;

  /* U svakoj iteraciji ove petlje pronalazi se najmanji element
     medju elementima a[i], a[i+1],...,a[n-1], i postavlja se na
     poziciju i, dok se element na pozciji i premesta na poziciju
     min, na kojoj se nalazio najmanji od navedenih elemenata. */
  for (i = 0; i < n - 1; i++) {
    /* Unutrasnja petlja pronalazi poziciju min, na kojoj se nalazi
       najmanji od elemenata a[i],...,a[n-1]. */
    min = i;
    for (j = i + 1; j < n; j++)
      if (a[j] < a[min])
        min = j;

    /* Zamena elemenata na pozicijama (i) i min. Ovo se radi samo
       ako su (i) i min razliciti, inace je nepotrebno. */
    if (min != i) {
      pom = a[i];
      a[i] = a[min];
      a[min] = pom;
    }
  }
}

void insertion_sort(int a[], int n)
{
  int i, j;

  /* Na pocetku iteracije pretpostavlja se da je niz
     a[0],...,a[i-1] sortiran */
  for (i = 1; i < n; i++) {
    /* U ovoj petlji se redom potiskuje element a[i] ulevo koliko
       je potrebno, dok ne zauzme pravo mesto, tako da niz
       a[0],...a[i] bude sortiran. Indeks j je trenutna pozicija na
       kojoj se element koji se umece nalazi. Petlja se zavrsava
       ili kada element dodje do levog kraja (j==0) ili kada se
       naidje na element a[j-1] koji je manji od a[j]. */
    int temp = a[i];
    for (j = i; j > 0 && temp < a[j - 1]; j--)
      a[j] = a[j - 1];
    a[j] = temp;
  }
}

void bubble_sort(int a[], int n)
{
  int i, j;
  int ind;

  for (i = n, ind = 1; i > 1 && ind; i--)
    /* Poput "mehurica" potiskuje se najveci element medju
```

```
       elementima od a[0] do a[i-1] na poziciju i-1 uporedjujuci
       susedne elemente niza i potiskujuci veci u desno */
    for (j = 0, ind = 0; j < i - 1; j++)
      if (a[j] > a[j + 1]) {
        int temp = a[j];
        a[j] = a[j + 1];
        a[j + 1] = temp;
        /* Promenljiva ind registruje da je bilo premestanja. Samo
           u tom slucaju ima smisla ici na sledecu iteraciju, jer
           ako nije bilo premestanja, znaci da su svi elementi vec
           u dobrom poretku, pa nema potrebe prelaziti na kraci
           prefiks niza. Algoritam bi bio korektan i bez ovoga.
           Sortiranje bi bilo ispravno, ali manje efikasno, jer bi
           se cesto nepotrebno vrsila mnoga uporedjivanja, kada je
           vec jasno da je sortiranje zavrseno. */
        ind = 1;
      }
}

void shell_sort(int a[], int n)
{
  int h = n / 2, i, j;
  while (h > 0) {
    /* Insertion sort sa korakom h */
    for (i = h; i < n; i++) {
      int temp = a[i];
      j = i;
      while (j >= h && a[j - h] > temp) {
        a[j] = a[j - h];
        j -= h;
      }
      a[j] = temp;
    }
    h = h / 2;
  }
}

void merge_sort(int a[], int l, int d)
{
  int s;
  static int b[MAX];            /* Pomocni niz */
  int i, j, k;

  /* Izlaz iz rekurzije */
  if (l >= d)
    return;

  /* Odredjivanje sredisnjeg indeksa */
  s = (l + d) / 2;

  /* Rekurzivni pozivi */
  merge_sort(a, l, s);
  merge_sort(a, s + 1, d);

  /* Inicijalizacija indeksa. Indeks i prolazi krozi levu polovinu
     niza, dok indeks j prolazi kroz desnu polovinu niza. Indeks k
     prolazi kroz pomocni niz b[] */
  i = l;
  j = s + 1;
  k = 0;

  /* "Ucesljavanje" koriscenjem pomocnog niza b[] */
  while (i <= s && j <= d) {
    if (a[i] < a[j])
```

```
      b[k++] = a[i++];
    else
      b[k++] = a[j++];
  }

  /* U slucaju da se prethodna petlja zavrsila izlaskom promenljive
     j iz dopustenog opsega u pomocni niz se prepisuje ostatak leve
     polovine niza */
  while (i <= s)
    b[k++] = a[i++];

  /* U slucaju da se prethodna petlja zavrsila izlaskom promenljive
     i iz dopustenog opsega u pomocni niz se prepisuje ostatak desne
     polovine niza */
  while (j <= d)
    b[k++] = a[j++];

  /* Prepisuje se "ucesljani" niz u originalni niz */
  for (k = 0, i = l; i <= d; i++, k++)
    a[i] = b[k];
}

/* Pomocna funkcija koja menja mesto i-tom i j-tom elementu niza a */
void swap(int a[], int i, int j)
{
  int tmp = a[i];
  a[i] = a[j];
  a[j] = tmp;
}

void quick_sort(int a[], int l, int d)
{
  int i, pivot_pozicija;

  /* Izlaz iz rekurzije -- prazan niz */
  if (l >= d)
    return;

  /* Particionisanje niza. Svi elementi na pozicijama levo od
     pivot_pozicija (izuzev same pozicije l) su strogo manji od
     pivota. Kada se pronadje neki element manji od pivota, uvecava
     se promenljiva pivot_pozicija i na tu poziciju se premesta
     nadjeni element. Na kraju ce pivot_pozicija zaista biti
     pozicija na koju treba smestiti pivot, jer ce svi elementi levo
     od te pozicije biti manji a desno biti veci ili jednaki od
     pivota. */
  pivot_pozicija = l;
  for (i = l + 1; i <= d; i++)
    if (a[i] < a[l])
      swap(a, ++pivot_pozicija, i);

  /* Postavljanje pivota na svoje mesto */
  swap(a, l, pivot_pozicija);

  /* Rekurzivno sortiranje elementa manjih od pivota */
  quick_sort(a, l, pivot_pozicija - 1);
  /* Rekurzivno sortiranje elementa vecih od pivota */
  quick_sort(a, pivot_pozicija + 1, d);
}
```

MAIN.C

```c
/* Program testira funkcije koje sortiraju niz celih brojeva */
#include <stdio.h>
#include "sort.h"

int main()
{
    int a[] = { 11,7,3,14,1,17,81,18,24,59,52,31,44,26,9,37 };
    int n = sizeof(a) / sizeof(int);
    int i;

        //  selection_sort(a,n);
        //  insertion_sort(a,n);
          bubble_sort(a,n);
        //merge_sort(a,0,n-1);
        //quick_sort(a,0,n-1);


    printf("\n----- Sortiran niz -----\n ");
    for( i =0; i<n ; i++ )
        printf(" %d ", a[i]);

    putchar('\n');

    return 0;
}
```