Zadatak za zadaću

- Napišite program koji će za bilo koju vrijednost cijelog broja n provjeriti ispravnost sljedećih izraza:

$$\sum_{i=1}^{n} i = n(n+1)/2$$

$$\sum_{i=1}^{n} i^2 = n(n+1)(2n+1)/6$$

- Izračunavanje svake od gornjih suma je potrebno obaviti u posebnoj funkciji, a za svaku od suma je potrebno napraviti funkciju koja računa sumu rekurzivnim i nerekurzivnim postupkom.
- Rok za predaju zadatka: 8. 12 2006.

Algoritam binarnog pretraživanja

- Za pretraživanje uzlazno sortiranog polja koristimo algoritam koji je bitno efikasniji (brži) od linearnog pretraživanja. Njegov rad opisat ćemo na primjeru:
- Pretpostavimo da u *sortiranom* polju

  2, 5, 7, 8, 10, 17, 19, 23, 25, 27, 33, 35, 36, 40, 42, 45, 53

  tražimo element 17.
- Prvo pogledamo element koji se nalazi u sredini polja (to je element 25) i usporedimo ga s brojem kojeg tražimo (broj 17). Kako je 17 manji od 25, zaključujemo da se 17 nikako ne može nalaziti desno od 25 (jer je polje uzlazno sortirano!) pa elemente 27, 33, 35, 36, 40, 42, 45, 53 ne trebamo niti gledati – svi su oni veći od 17. Stoga u drugom koraku gledamo samo podniz koji se nalazi lijevo od 25:

  2, 5, 7, 8, 10, 17, 19, 23
- Sada pogledamo srednji element ovog podniza. Srednji element u ovom slučaju je 8. Kako je 17 veći od 8, zaključujemo da se 17 nikako ne može nalaziti lijevo od srednjeg elementa 8, pa elemente lijevo od 8 (2, 5 i 7) ne moramo ni gledati, nego u trećem koraku gledamo podniz desno od 8:

  10, 17, 19, 23
- Pogledamo srednji element ovog podniza. No, srednji element je upravo jednak traženom elementu 17, pa algoritam završava s radom jer je pronašao ono što je tražio.

```
#include <stdio.h>
#include <stdlib.h>
int BinarySearch(int a[], int length, int key);

void main(){
    int n, i, k;
    int *niz;
    int rezultat;
    printf("Unesi duljinu polja:\n");
    scanf("%d",&n);
    if ((niz = (int *)malloc(n*sizeof(int))) == NULL) {
        printf("Greska u alociranju memorije !");
        exit (1);
            }

    for(i = 0; i < n; i++)
            niz[i] = 2*i + 1;
    printf("Elementi niza su:\n");
    for(i = 0; i < n; i++)
            printf("%d ", niz[i]);

    printf("\nUnesi element pretrage: ");
    scanf("%d", &k);
    rezultat = BinarySearch(niz, n, k);
```

```
    if (rezultat == -1)
            printf("Uneseni element se ne nalazi u nizu!\n");
     else
            printf("Uneseni element se nalazi na poziciji %d", rezultat+1);
    system("PAUSE");
  }

int BinarySearch(int a[], int length, int key) {
     int left = 0;
     int right = length - 1;
     int middle;
     do     {
            middle = (left + right) / 2;
            if (a[middle] == key)
                    return middle;
            else if (a[middle] < key)
                    left = middle + 1;
            else
                    right = middle - 1;
     }  while (left <= right);
     return -1;
}
```

- Efikasnost algoritma pretraživanja:
- Da bi pronašli traženi element u sortiranom nizu duljine $n=2^m$ koristeći algoritam binarnog pretraživanja potrebno je u najgorem slučaju m+1 korak
- $2^m \rightarrow 2^{m-1} \rightarrow \ldots \rightarrow 2^1 \rightarrow 2^0$ koraka
- Za niz duljine $2^{m-1} < n < 2^m$ potrebno je također najviše m+1 koraka

- Za pronalaženje elementa u nizu duljine n upotrebom linearnog pretraživanja potrebno je u najgorem slučaju n koraka

- Npr. n= 128, algoritam binarnog pretraživanja treba maksimalno 8 koraka, a algoritam linearnog pretraživanja 128 koraka

Pronalaženje najvećeg elementa u nesortiranom polju

- Napisati program koji će pronaći najveći broj u polju slučajno generiranih cijelih brojeva. Generirane brojeve ispisati u datoteku.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <limits.h>
#define N 100

int main() {
    short int a[N], max, i;
    FILE *fpout;

    if ((fpout=fopen("proba.txt","w"))==NULL) {
                    printf("Greska u otvaranju datoteke!\n");
                      exit(1);
          }
```

```
printf ("Generirat cu %d brojeva izmedju 0 i %d\n",N,SHRT_MAX/N);

srand(time(NULL));
for(i = 0; i < N; i++) {
        srand(time(NULL)/rand());
        a[i]= SHRT_MAX/N *((float) rand()/(RAND_MAX + 1));
        fprintf(fpout,"%d\n",a[i]);
}

max = a[0];
for(i = 1; i < N; i++)
        if (a[i] > max)
                max = a[i];

printf("Najveci uneseni broj je %d\n", max);
system("PAUSE");
return 0;
}
```

Sortiranje metodom umetanja (Insertion Sort)

- Zadano je polje nesortiranih cijelih brojeva koje treba sortirati.
- Ideja algoritma: članovi polja se međusobno uspoređuju i manji član se uvijek stavi ispred većeg, procedura se napravi za sve članove polja
- Primjer: zadan niz 4, 8, 1, 2
- Počnemo s drugim elementom 8 koji se spremi u privremenu varijablu i krenemo u proceduru za prvi element 4: 8 je veći od 4, pa ne zamjenjuju mjesta
- Sad izaberemo treći element 1 i uspoređujemo ga s drugim elementom 8: on je veći od 1, pa drugi element ubacujemo na mjesto trećeg; zatim ponovimo za prvi element: on je također veći od 1, pa se prebacuje na drugo mjesto, čime dobijemo niz 1, 4, 8, 2
- Istu proceduru ponovimo za četvrti element: on je manji od trećeg i drugog, pa se oni pomiču za jedno mjesto gore i 2 dolazi na drugo mjesto; usporedna s prvim elementom pokazuje da su oni dobro sortirani i ne mijenjaju se njihove pozicije, pa je rezultat drugog prolaska sortirani niz 1, 2, 4, 8

```
#include <stdio.h>
#define N 30
void InsertionSort(int a[], int n);
int main(){
	int i;
	int a[N] ={21,99,10,4,7,12,3,66,9,0,55,27,18,1,6,8,15,2,14,5,19,76,17,13,82,11,16,71,50,30};
	printf("Nesortirano polje:\n");
	for(i = 0; i < N; i++)	printf("%d ", a[i]);	printf("\n");
	InsertionSort(a, N);
	printf("Sortirano polje:\n");
	for(i = 0; i < N; i++)	printf("%d ", a[i]);	printf("\n");
	system("PAUSE");
	return 0;
}
void InsertionSort(int a[], int n) {
	int i,j,k;
	for(j = 1; j < n; j++) {
		k = a[j];
		i = j - 1;
		while(a[i] > k && i >= 0) {
			a[i + 1] = a[i];
			i--;
		}  a[i + 1] = k;
	}
}
```

Mjehuričasto sortiranje – Bubble sort

- Ideja algoritma:
- Pretpostavimo da na ulazu imamo sljedeće polje od 6 elemenata: 5, 3, 10, 2, 6, 4
- Prvo uspoređujemo drugi element polja (3) s prvim elementom (3): **5, 3**, 10, 2, 6, 4
- Ukoliko je prvi uspoređivani element veći od drugog, a ovdje jest, oni međusobno zamijene svoja mjesta. U suprotnom slučaju se zamjena ne obavlja. Nakon zamjene dobivamo sljedeću situaciju:

  3, 5, 10, 2, 6, 4
- Zatim se uspoređuju drugi i treći element: 3, **5, 10**, 2, 6, 4
- 5 je manji od 10, pa nema zamjene: 3, 5, 10, 2, 6, 4
- Ovaj postupak se nastavlja dok se ne dođe do zadnjeg elementa:

  U našem slučaju, rad algoritma nastavio bi se ovako:

Uspoređivanje:	3, 5, **10**, **2**, 6, 4
Nakon (eventualne) zamjene:	3, 5, 2, 10, 6, 4
Uspoređivanje :	3, 5, 2, **10**, **6**, 4
Nakon (eventualne) zamjene:	3, 5, 2, 6, 10, 4
Uspoređivanje :	3, 5, 2, 6, **10**, **4**
Nakon (eventualne) zamjene:	3, 5, 2, 6, 4, 10
- Ovim postupkom najveći element došao na svoje mjesto, na kraj niza.
- Sada "skraćujemo" niz za jedan element i ponavljamo ponovno gornji postupak uspoređivanja, ponovno počinjući od prvog elementa polja.

- Algoritam nastavlja ovako (oznakom **|** odvojili smo naše "skraćeno" polje od ostatka koji je već sortiran):

Uspoređivanje :	**3**, **5**, 2, 6, 4 | 10
Nakon (eventualne) zamjene:	3, 5, 2, 6, 4 | 10
Uspoređivanje :	3, **5**, **2**, 6, 4 | 10
Nakon (eventualne) zamjene:	3, 2, 5, 6, 4 | 10
Uspoređivanje:	3, 2, **5**, **6**, 4 | 10
Nakon (eventualne) zamjene:	3, 2, 5, 6, 4 | 10
Uspoređivanje :	3, 2, 5, **6**, **4** | 10
Nakon (eventualne) zamjene:	3, 2, 5, 4, 6 | 10

- Nakon ovog kruga zamjena opet je (ali ovaj put u "skraćenom" polju, bez elementa 10) na njegov kraj došao najveći element (broj 6). Sada ponovo skraćujemo polje i započinjemo novi krug zamjena:

Uspoređivanje :	**3**, **2**, 5, 4 | 6, 10
Nakon (eventualne) zamjene:	2, 3, 5, 4 | 6, 10
Uspoređivanje :	2, **3**, **5**, 4 | 6, 10
Nakon (eventualne) zamjene:	2, 3, 5, 4 | 6, 10
Uspoređivanje :	2, 3, **4**, **5** | 6, 10

- Element 5 došao je nakon ovog kruga zamjena na svoje pravo mjesto, pa opet skraćujemo polje za jedan element i započinjemo novi krug usporedbi:

Uspoređivanje :	**2**, **3**, 4 | 5, 6, 10
Nakon (eventualne) zamjene:	2, 3, 4 | 5, 6, 10
Uspoređivanje :	2, **3**, **4** | 5, 6, 10

- Vidimo da u ovom krugu ni jednom nismo morali zamijeniti neka dva elementa. To znači da je niz sortiran, pa algoritam završava s radom.

```
#include <stdio.h>
#include <stdlib.h>
#define N 30
void ispis(int[], int);
void bubble_sort(int[], int);

int main(){
int polje[N]={31,51,14,99,0,54,19,76,66,78,23,41,50,2,41,5,52,9,63,4,74,17,88,8,11,7,1,91,3,6};

     printf("Nesortirano polje:\n");
     ispis(polje, N);
     bubble_sort(polje, N);
     printf("Sortirano polje:\n");
     ispis(polje, N);
     system("PAUSE");
     return 0;
}

void ispis(int a[], int n){
     for(int i = 0; i < n; i++)
              printf("%d ", a[i]);
     printf("\n");
}
```

```
void bubble_sort(int a[], int n){
    int flag,temp;
    do {
        flag = 0;
        for(int i = 0; i < n - 1; i++) {
            if (a[i] > a[i + 1]) {
                temp = a[i];
                a[i] = a[i + 1];
                a[i + 1] = temp;
                flag = 1;
            }
        }
        n--;
    }  while (flag != 0);
}
```

Eratostenov algoritam za nalaženje prostih brojeva

- Za prirodan broj p veći od 1 kažemo da je *prost (prim)* ako nema drugih djelitelja osim 1 i samog broja p.
- Ukoliko želimo naći sve proste brojeve između 2 i n, obično se služimo postupkom kojeg je prvi opisao antički matematičar Eratosten.
- Primjer: naći sve proste brojeve između 2 i 29. Najprije napišemo sve te brojeve jedan iza drugoga:
- 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29
- Zatim u gornjem nizu označimo sve višekratnike broja 2 (ali ne i sam broj 2):
- 2, 3, **4**, 5, **6**, 7, **8**, 9, **10**, 11, **12**, 13, **14**, 15, **16**, 17, **18**, 19, **20**, 21, **22**, 23, **24**, 25, **26**, 27, **28**, 29
- Sada se s prvog elementa polja (element 2) pomaknemo na sljedeći *neoznačeni* element. To je broj 3. Označimo sada sve višekratnike broja 3 (ali ne i sam broj 3). Dobijemo ovo:
- 2, 3, **4**, 5, **6**, 7, **8**, **9**, **10**, 11, **12**, 13, **14**, **15**, **16**, 17, **18**, 19, **20**, **21**, **22**, 23, **24**, 25, **26**, **27**, **28**, 29
- Sada se s elementa 3 pomičemo na sljedeći neoznačeni element. To je broj 5.
- 2, 3, **4**, 5, **6**, 7, **8**, **9**, **10**, 11, **12**, 13, **14**, **15**, **16**, 17, **18**, 19, **20**, **21**, **22**, 23, **24**, **25**, **26**, **27**, **28**, 29
- Sljedeći neoznačeni element bio bi broj 7. Međutim njegove višekratnike ne moramo označavati!
- Dovoljno je označiti višekratnike samo onih prirodnih brojeva koji su manji od drugog korijena najvećeg elementa u nizu. Kako je sqrt(29) ≈ 5.3851, označeni su svi potrebni višekratnici.
- Nakon završenog postupka, u nizu će neoznačeni ostati *jedino prosti brojevi*.
- Napisati program koji Eratostenovim algoritmom pronalazi i ispisuje sve proste brojeve manje od 100000.

```
#include <stdio.h>
#include<stdlib.h>
#include<math.h>
#define MAXSIZE 100001
void Eratosten(int a[], int n);
void main(){
    int max;
    int prosti[MAXSIZE] = { 0 };
    printf("Unesi broj manji ili jednak %d: ", MAXSIZE - 1);   scanf("%d", &max);
    printf("\n");
    Eratosten(prosti, max);
    printf("Svi prosti brojevi manji ili jednaki %d su:\n", max);
    for (int i = 2; i <= max; i++)
        if (prosti[i] == 0)    printf("%d ", i);
    printf("\n");
    system("PAUSE");
}
void Eratosten(int a[], int n){
    for(int i = 2; i <= sqrt(n); i++) {
        if (a[i] == 0) {
            int k = 2;
            while (k * i <= n){
                a[k * i] = 1;
                k++;}  }  }
}
```

Zadaci za zadaću

- Napisati program za sortiranje metodom umetanja (Insertion sort) i mjehuričastim sortiranjem (Bubble sort) koji sortiraju elemente silazno (od najvećeg prema najmanjem, obrnuto od primjera napravljenog na vježbama). Sortiraju se polja generiranih slučajnih cijelih brojeva. Pronaći u koliko se koraka sortira polje u najgorem slučaju za oba algoritma i pronaći koji je brži za vrlo dugačke nizove brojeva.

- Koristeći Eratostenov algoritam s vježbi, napraviti program koji će za bilo koji upisani prirodan broj (maksimalan broj nije unaprijed zadan) odrediti da li je prim broj i ako nije prikazat ga kao umnožak prim brojeva (bez ostatka, tj. u obliku $x=p_1*p_2*\ldots p_k$).