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

```
void bubble_sort(int a[], int n){
    int flag,temp, i;
    do {
        flag = 0;
        for(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, i;
	int prosti[MAXSIZE] = { 0 };
	printf("Unesi broj manji ili jednak %d: ", MAXSIZE - 1);   scanf("%d", &max);
	printf("\n");
	Eratosten(prosti, max);
	printf("Svi prim brojevi manji ili jednaki %d su:\n", max);
	for (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++;}  }  }
}
```

# Primjer za različite složenosti istog problema

- Zadano je polje cijelih brojeva $A_0$, $A_1$,...,$A_{n-1}$. Brojevi mogu biti i negativni. Potrebno je pronaći najveću vrijednost sume niza brojeva. Pretpostavit će se da je najveća suma 0 ako su svi brojevi negativni. Problem je razmatran na predavanjima.
- Kubna složenost: Ispituju se svi mogući podnizovi. U vanjskoj petlji se varira prvi član podniza, u srednjoj petlji varira se zadnji član podniza, u unutrašnjoj petlji varira se duljina niza od prvog člana do zadnjeg člana. Sve 3 petlje se za najgori slučaj obavljaju n puta: apriorna složenost $O(n^3)$.
- Kvadratna složenost: ako uočimo da je treća petlja nepotrebna (ne treba se suma svaki put računati iznova), složenost se može reducirati na $O(n^2)$.
- Linearna * logaritamska složenost: $O(n \log_2 n)$ - relativno složeni rekurzivni postupak. Ako se ulazno polje podijeli približno po sredini, rješenje može biti takvo da je maksimalna suma u lijevom dijelu polja, ili je u desnom dijelu polja ili prolazi kroz oba dijela. Prva dva slučaja mogu biti riješena rekurzivno. Zadnji slučaj se može realizirati tako da se nađe najveća suma u lijevom dijelu koja uključuje njegov zadnji član i najveća suma u desnom dijelu koja uključuje njegov prvi član. Te se dvije sume zbroje i uspoređuju s one prve dvije.
- Linearna složenost: zbrajaju se svi članovi polja redom, a pamti se ona suma koja je u cijelom tijeku tog postupka bila najveća, pa je složenost algoritma $O(n)$

```
#include <stdio.h>
#include <stdlib.h>

// vraca niz znakova c u zadanoj duljini n
char* nc (int c, int n) {
      static char s[80+1];
      s[n] = '\0';                    // prirubi
      while (--n >= 0) s[n] = c; // popuni
      return s;
}

// ispis polja
void ispisi(int A[], int n) {
      int i;
      printf("\n");
      for (i = 0; i < n; i++) printf(" A[%d]",i);
      printf("\n");
      for (i = 0; i < n; i++) printf("%5d", A[i]);
      printf("\n");
}
```

```
// Kubna slozenost
int MaxPodSumaNiza3 (int A[], int N) {
      int OvaSuma, MaxSuma, i, j, k;
      int iteracija = 0;

      MaxSuma = 0;
      for (i = 0; i < N; i++) {
            printf ("i=%d\n", i);

            for (j = i; j < N; j++) {
                  OvaSuma = 0;
                  for (k = i; k <= j; k++) {
                        OvaSuma += A [k];
                        ++iteracija;
                  }
       if (OvaSuma > MaxSuma)
            MaxSuma = OvaSuma;
            printf ("Suma clanova [%d, %d] = %d, a najveca = %d\n",i, j, OvaSuma, MaxSuma);
            }
      }
      printf ("Broj iteracija: %d\n", iteracija);
      return MaxSuma;
}
```

```
// Kvadratna slozenost
int MaxPodSumaNiza2 (int A[ ], int N) {
      int OvaSuma, MaxSuma, i, j;
      int iteracija = 0;

      MaxSuma = 0;
      for (i = 0; i < N; i++) {
            printf ("i=%d\n", i);

            OvaSuma = 0;
            for (j = i; j < N; j++) {
                  OvaSuma += A[ j ];
                  ++iteracija;
                  if (OvaSuma > MaxSuma)
                     MaxSuma = OvaSuma;
                 printf ("Suma clanova [%d, %d] = %d, a najveca = %d\n",i, j, OvaSuma, MaxSuma);
            }
      }
      printf ("Broj iteracija: %d\n", iteracija);
      return MaxSuma;
}
```

```
// NlogN slozenost - koristi funkcije Max3 i MaxPodSuma
// racuna najveci od 3 broja
int Max3 (int A, int B, int C) {
      return A > B ? A > C ? A : C : B > C ? B : C;
}
// trazi najvecu podsumu clanova od Lijeva do Desna
int MaxPodSuma (int A[], int Lijeva, int Desna, int dubina) {
      int MaxLijevaSuma, MaxDesnaSuma;
      int MaxLijevaRubnaSuma, MaxDesnaRubnaSuma;
      int LijevaRubnaSuma, DesnaRubnaSuma;
      int Sredina, i, ret;

  printf ("%s> MaxPodSuma(%d, %d) ...\n", nc(' ', dubina*2), Lijeva, Desna);
      if (Lijeva == Desna) { // Osnovni slucaj
        if (A [Lijeva] > 0)
                ret = A [Lijeva]; // podniz od clana A[Lijeva]
        else
                ret = 0; // suma je 0 ako su svi brojevi negativni
     printf ("%s< MaxPodSuma(%d, %d) = %d\n",nc(' ', dubina*2), Lijeva, Desna, ret);
              return ret;
      }
// racun lijeve i desne podsume s obzirom na Sredina
      Sredina = (Lijeva + Desna) / 2;
      MaxLijevaSuma = MaxPodSuma (A, Lijeva, Sredina, dubina+1);
      MaxDesnaSuma = MaxPodSuma (A,Sredina + 1, Desna, dubina+1);
```

```
// najveca gledano ulijevo od sredine
    MaxLijevaRubnaSuma = 0; LijevaRubnaSuma = 0;
    for (i = Sredina; i >= Lijeva; i--) {
            LijevaRubnaSuma += A [i];
            if (LijevaRubnaSuma > MaxLijevaRubnaSuma)
                        MaxLijevaRubnaSuma = LijevaRubnaSuma;
    }
    // najveca gledano udesno od sredine
  MaxDesnaRubnaSuma = 0; DesnaRubnaSuma = 0;
  for (i = Sredina + 1; i <= Desna; i++) {
    DesnaRubnaSuma += A [i];
    if (DesnaRubnaSuma > MaxDesnaRubnaSuma)
      MaxDesnaRubnaSuma = DesnaRubnaSuma;
  }
    printf ("%s  Lijeva=%d Desna=%d Rubna=%d\n",
                nc (' ', dubina*2), MaxLijevaSuma, MaxDesnaSuma,
                MaxLijevaRubnaSuma + MaxDesnaRubnaSuma);

    // najveca od lijeva, desna, rubna
  ret = Max3 (MaxLijevaSuma, MaxDesnaSuma,
                    MaxLijevaRubnaSuma + MaxDesnaRubnaSuma);
    printf ("%s< MaxPodSuma(%d, %d) = %d\n",
                            nc(' ', dubina*2), Lijeva, Desna, ret);
    return ret;
}
```

```
// NlogN slozenost
int MaxPodSumaNizaLog (int A [], int N) {
    return MaxPodSuma (A, 0, N - 1, 0);
}

// Linearna složenost
int  MaxPodSumaNiza1 (int A[], int N) {
    int OvaSuma, MaxSuma, j;

    OvaSuma = MaxSuma = 0;
    for (j = 0; j < N; j++) {
        OvaSuma += A[ j ];
        if (OvaSuma > MaxSuma)
            MaxSuma = OvaSuma;
        else if (OvaSuma < 0)
            OvaSuma = 0;        // povecanje izgleda sljedeceg podniza
        printf ("j=%d OvaSuma=%2d MaxSuma=%2d\n", j, OvaSuma, MaxSuma);
    }
    return MaxSuma;
}
```

```
void main (void) {
  int A [] = {2,5,-8,4,-2,5,-5,4,3,-2};
  int rez;
      printf("\n\nKubna slozenost\n");
      ispisi(A, sizeof (A) / sizeof (A [0]));
      rez = MaxPodSumaNiza3 (A, sizeof (A) / sizeof (A [0]));
       printf("\nMaxSuma3 = %d", rez);
       getchar();
      printf("\n\nKvadratna slozenost\n");
      ispisi(A, sizeof (A) / sizeof (A [0]));
      rez = MaxPodSumaNiza2 (A, sizeof (A) / sizeof (A [0]));
       printf("\nMaxSuma2 = %d", rez);
      getchar();
      printf("\n\nLogaritamska slozenost\n");
      ispisi(A, sizeof (A) / sizeof (A [0]));
      rez = MaxPodSumaNizaLog (A, sizeof (A) / sizeof (A [0]));
       printf("\nMaxSumaLog = %d", rez);
       getchar();
      printf("\n\nLinearna slozenost\n");
      ispisi(A, sizeof (A) / sizeof (A [0]));
      rez = MaxPodSumaNiza1 (A, sizeof (A) / sizeof (A [0]));
       printf("\nMaxSuma1 = %d\n", rez);
       system("PAUSE");
      exit (0);
}
```