Strukture podataka i algoritmi

Sortiranje

dr. sc. Hrvoje Kalinić

Sortiranje

Sortiranje je postupak ponovnog raspoređivanja stvari prema nekoj njihovoj odrednici, na način da te stvari budu raspoređene uzlazno ili silazno.

Neke od najvažnijih primjena sortiranja:

rješenje problema zajedništva, odnosno grupiranja elemenata ovisno o nekoj njihovoj karakteristici (značajci, opisu)

ako je dano 10 tisuća elemenata u proizvoljnom poretku, pri čemu mnogi od njih imaju istu vrijednost, a potrebno je presložiti te elemente tako da svi elementi s jednakim vrijednostima budu jedan pored drugoga

uspoređivanje elemenata u jednoj ili više datoteka

ako je nekoliko datoteka sortirano na isti način (npr. uzlazno) moguće je pronaći sve podatke koji se poklapaju u samo jednom sekvencijalnom prolazu kroz njih, bez kretanja u suprotnom smjeru

potraga za informacijom prema vrijednosti ključa

telefonski imenik

Sortiranje (Povijest)

- Jedna od prvih primjena sortiranja u velikim informacijskim sustavima je bila 1960. u tvrtci Computer Sciences Corporation
- U tom desetljeću proizvođači računala su tvrdili kako više od 25% vremena u kojem njihova računala rade, otpada na sortiranje, a u nekim tvrtkama taj postotak je rastao i do 50%.
- Odavde su se kristalizirala 3 zaključka
  - sortiranje ima mnogo važnih primjena
  - ljudi sortiraju kada i ne bi trebali
  - u upotrebi su neučinkoviti algoritmi sortiranja
- Na prvi i drugi zaključak računarstvo nije moglo utjecati, ali razvojem novih i učinkovitijih algoritama sortiranja je značajno utjecalo na rješenje posljednjeg problema

Sortiranje

- Za bolje shvaćanje sortiranja potrebno je pobliže definirati problem kojeg mora riješiti
- Dano je n elemenata $R_1, R_2, \ldots, R_n$ koje treba sortirati, pri čemu se elementi često zovu zapisi (eng. **Record**)
- Svakom od zapisa R_j pripada odgovarajući **ključ** K_j koji usmjerava proces sortiranja.
  - Ključevi predstavljaju neku odrednicu zapisa po kojoj se sortira. Ako zapis predstavlja studenta s imenom, prezimenom, godinom rođenja i sl. onda jedan od ključeva može biti dob ili prezime

Sortiranje

- Zapise sortiranjem valja poredati na specifičan način, a ta relacija poretka (uređaja) koja se definira nad ključevima mora zadovoljavati sljedeće uvjete za bilo koja tri ključa *a,b,c*
  - **načelo trihotomije** - uvijek vrijedi točno jedan od izraza: $a<b$, $a=b$ *ili* $a>b$
  - **tranzitivnost** - iz odnosâ $a<b$ i $b<c$ slijedi odnos $a<c$
- Cilj sortiranja je pronaći permutaciju *p(1), p(2), ... , p(n)* nad indeksima {1,2, ..., n} koja će postaviti ključeve u nepadajući poredak

$$K_{p(1)} \leq K_{p(2)} \leq \ldots \leq K_{p(n)}$$

- Sortiranje se dodatno naziva **stabilnim** ako zapisi sa istim ključevima nakon sortiranja zadrže svoj isti relativni poredak u odnosu na početno stanje

Sortiranje

- Neka je dano 8 uzastopnih memorijskih lokacija M1, M2, M3, ..., M8, a na svakoj od njih po jedan cijeli broj. Napiši program koji će izmijeniti poredak brojeva na lokacijama, ako je to potrebno, tako da budu poredani uzlazno.

Sortiranje

- Neka je dano 8 uzastopnih memorijskih lokacija M1, M2, M3, ..., M8, a na svakoj od njih po jedan cijeli broj. Napiši program koji će izmijeniti poredak brojeva na lokacijama, ako je to potrebno, tako da budu poredani uzlazno.

- Za vježbu, pokušajte ovo riješiti na neki vaš način! Može i na papiru

Sortiranje

- Ako ste napravili takav algoritam on gotovo sigurno spada u jedan od sljedećih tipova:
  - sortiranje izborom
    - prvo se pronađe najmanji (najveći) element i na neki način izdvoji od ostatka, a potom se proces ponavlja za preostale elemente
  - sortiranje izmjenom
    - ako se za dva susjedna broja utvrdi da nisu u povoljnom međusobnom odnosu tada oni mijenjaju mjesta, a ovaj postupak se ponavlja sve dok zamjene više nisu potrebne
  - sortiranje umetanjem
    - brojevi se razmatraju jedan po jedan i svaki novorazmatrani broj se umeće na odgovarajuću poziciju ovisno o prethodno sortiranim elementima
  - sortiranje spajanjem
  - nešto totalno drugačije

Sortiranje izborom

- Eng. Selection sort

```
for i ← N to 2
    K ← max(K1...Kj)
    Zamjeni(K,Kj)
```

STRAIGHT SELECTION SORTING

503	087	512	061	**908**	170	**897**	275	653	426	154	509	612	677	**765**	**703** |
503	087	512	061	703	170	**897**	275	653	426	154	509	612	677	**765** |	908
503	087	512	061	703	170	**765**	275	653	426	154	509	612	**677** |	897	908
503	087	512	061	**703**	170	**677**	275	**653**	426	154	509	**612** |	765	897	908
503	087	512	061	612	170	**677**	275	**653**	426	154	**509** |	703	765	897	908
503	087	512	061	612	170	509	275	**653**	**426**	**154** |	677	703	765	897	908
...															
061 |	087	154	170	275	426	503	509	512	612	653	677	703	765	897	908

Sortiranje izborom

- Sortiranje izborom najmanjeg elementa

Početak:	16	30	5	44	12	21	9
1. korak:	5	30	16	44	12	21	9
2. korak:	5	9	16	44	12	21	30
3. korak:	5	9	12	44	16	21	30
4. korak:	5	9	12	16	44	21	30
5. korak:	5	9	12	16	21	44	30
6. korak:	5	9	12	16	21	44	30
7. korak:	5	9	12	16	21	30	44

Sortiranje izborom

```
static void Selection(int[] niz)
{
    int i, j, mmax;
    for (i = 0; i < niz.Count(); i++)
    {
        mmax = i;
        for (j = i; j < niz.Count(); j++)
            if (niz[mmax] > niz[j])
                mmax = j;
        Zamjena(ref niz[i], ref niz[mmax]);
    }
}
```

Sortiranje izborom

- Eng. Selection sort

```
for i ← N to 2
    K ← max(K1...Kj)
    Zamjeni(K,Kj)
```

Vremenska složenost za selection sort u najgorem slučaju je $O(n^2)$

Sortiranje izborom

- Eng. Selection sort

```
for i ← N to 2
    K ← max(K1...Kj)
    Zamjeni(K,Kj)
```

```
pom ← K
K ← Kj
Kj ← pom
```

Vremenska složenost za selection sort u najgorem slučaju je $O(n^2)$

Sortiranje izborom

- Eng. Selection sort

```
for i ← N to 2
    K ← max(K1...Kj)
    Zamjeni(K,Kj)
```

$K_1>K_2$
$K_2>K_3$
$K_2>K_4$
$K_4>K_5$

```
pom ← K
K ← Kj
Kj ← pom
```

Vremenska složenost za selection sort u najgorem slučaju je $O(n^2)$

Sortiranje izborom

- Eng. Selection sort

```
for i ← N to 2
    K ← max(K1...Kj)
    Zamjeni(K,Kj)
```

$K_1>K_2$
$K_2>K_3$
$K_2>K_4$
$K_4>K_5$

Sortiranje usporedbom?

```
pom ← K
K ← Kj
Kj ← pom
```

Vremenska složenost za selection sort u najgorem slučaju je $O(n^2)$

Sortiranje usporedbom

- Eng. Comparison sort
- Nekoliko najpoznatijih:
  - Bubble (ili exchange) sort - sortiranje izmjenom
  - Quicksort
  - Merge sort (sortiranje spajanjem)
  - Shell sort (Shellovo sortiranje)
  - Sortiranje umetanjem

Sortiranje usporedbom

- Eng. Comparison sort
- Nekoliko najpoznatijih:
  - Bubble (ili exchange) sort - sortiranje izmjenom
  - Quicksort
  - Merge sort (sortiranje spajanjem)
  - Shell sort (Shellovo sortiranje)
  - Sortiranje umetanjem

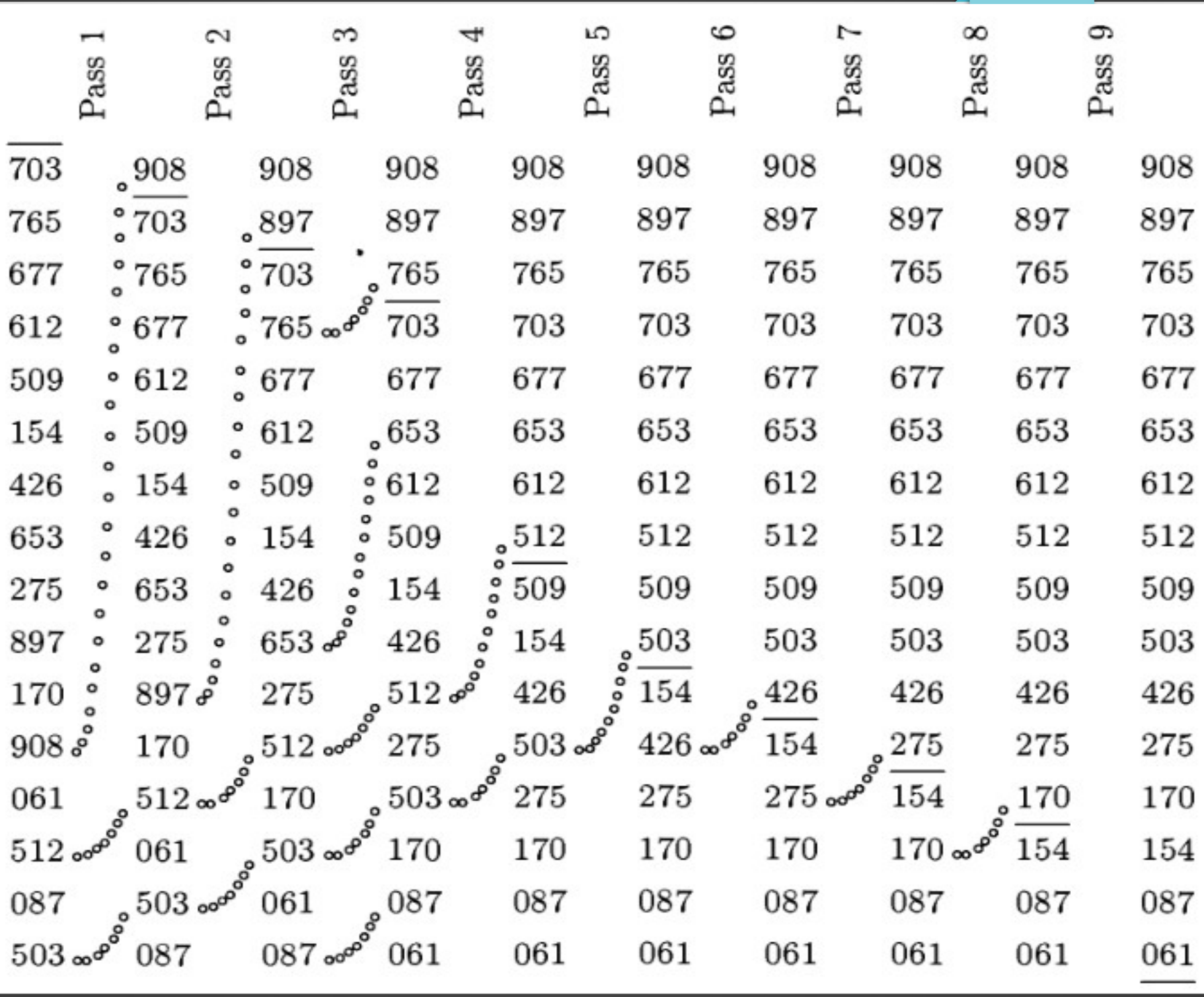
Sortiranje izmjenom

6 5 3 1 8 7 2 4

Sortiranje izmjenom

- Uspoređuje susjedne ključeve K1 i K2 te mijenja zapise R1 i R2 ako nisu u dobrom odnosu, a postupak nastaviti s parovima R2 i R3, R3 i R4 itd.
- A.K.A. Bubble sort - najveći elementi (ako sortiramo uzlazno) isplivaju na vrh poput mjehurića zraka
- U najboljem slučaju Bubble sort prolazi samo jednom kroz sve elemente, dok je načelno potrebno više prolaza
- Vremenska složenost u najgorem slučaju je $O(n^2)$

Sc


Pass 1
Pass 2
Pass 3
Pass 4
Pass 5
Pass 6
Pass 7
Pass 8
Pass 9
703 765 677 612 509 154 426 653 275 897 170 908 061 512 087 503
908 703 765 677 612 509 154 426 653 275 897 170 512 061 503 087
908 897 703 765 677 612 509 154 426 653 275 512 170 503 061 087
908 897 765 703 677 653 612 509 154 426 275 512 503 170 087 061
908 897 765 703 677 653 612 512 509 154 503 426 275 170 087 061
908 897 765 703 677 653 612 512 509 503 154 426 275 170 087 061
908 897 765 703 677 653 612 512 509 503 426 154 275 170 087 061
908 897 765 703 677 653 612 512 509 503 426 275 154 170 087 061
908 897 765 703 677 653 612 512 509 503 426 275 170 154 087 061
908 897 765 703 677 653 612 512 509 503 426 275 170 154 087 061

Sortiranje izmjenom

```
n = length(K)
repeat
  swapped = false
  for i = 1 to n-1
    /* ako parovi nisu poredani */
    if K[i-1] > K[i] then
      /* zamijeni i pamti zamjenu */
      swap( K[i-1], K[i] )
      swapped = true
    end if
  end for
until not swapped
```

Sortiranje izmjenom

- Algoritam:

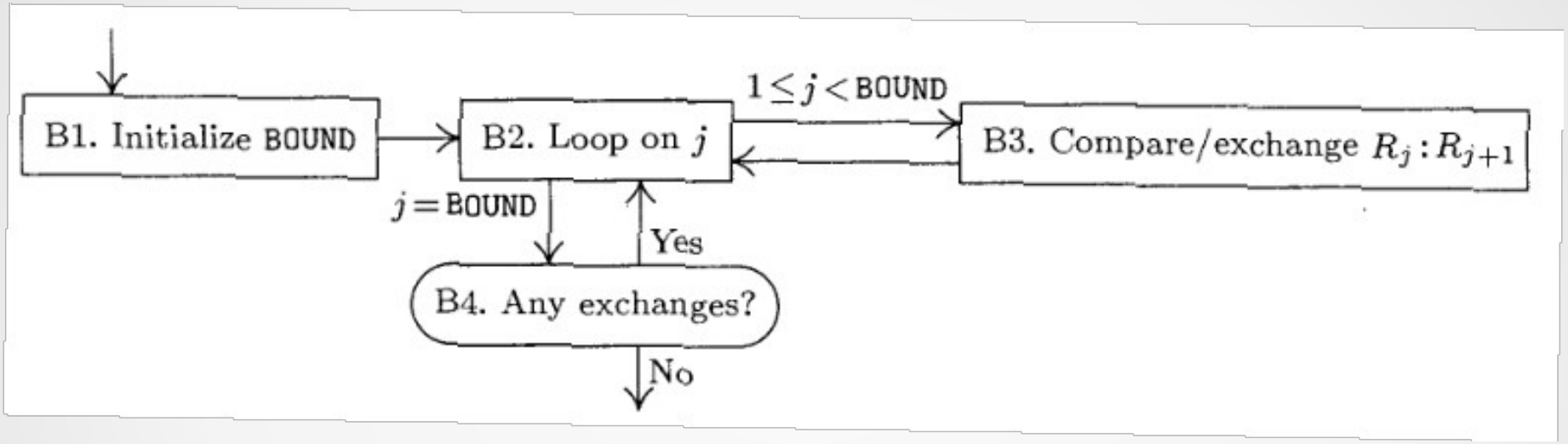
B1 [Postavi GRANICU] GRANICA = n

B2 [Vrti se u petlji po j] Postavi *t = 0*. Obavi korak B3 za j=1,2,...,Granica-1; zatim idi na korak B4

B3 [Usporedi/Zamijeni Rj i Rj+1] Ako je $K_j \geq K_{j+1}$ zamijeni R_j i R_{j+1} te postavi t=j

B4 [Je li bilo zamjena?] Ako je *t=0* završi algoritam. U protivnom GRANICA = t i vrati se na B2

Sortiranje izmjenom



Sortiranje izmjenom

```
static void Bubble(int[] niz)
{
    int i, j, t = 0;
    for (i = 0; i < niz.Count() - 1; i++)
    {
        for (j = i + 1; j < niz.Count(); j++)
            if (niz[i] < niz[j])
            {
                Zamjena(ref niz[i], ref niz[j]);
                t++;
            }
        if (t == 0)
            break;
    }
}
```

Sortiranje izmjenom

```
static void Bubble(int[] niz)
{
    int i, j, t = 0;
    for (i = 0; i < niz.Count() - 1; i++)
    {
        for (j = i + 1; j < niz.Count(); j++)
            if (niz[i] < niz[j])
            {
                Zamjena(ref niz[i], ref niz[j]);
                t++;
            }
        if (t == 0)
            break;
    }
}
```

Postoji li greška?

Sortiranje usporedbom

- Eng. Comparison sort
- Nekoliko najpoznatijih:
  - Bubble (ili exchange) sort - sortiranje izmjenom
  - Quicksort
  - Merge sort (sortiranje spajanjem)
  - Shell sort (Shellovo sortiranje)
  - Sortiranje umetanjem

Quicksort

- A.K.A. Sortiranje rekurzivnom izmjenom
- A.K.A. Sortiranje izmjenom djelova (ne elemenata)
  - eng. partition-exchange sort
- Nekada "quick sort", danas "quicksort"

Quicksort

- Unaprijeđenje bubble sorta je algoritam Quicksort kojeg je izumio C.A.R. Hoare 1960. kao gostujući student na Moskovskom Sveučilištu.
- Algoritam koristi rekurziju i dijeli niz na dva podniza (eng. partition) za koja vrijedi da su svi elementi jednog niza veći do svih elemenata drugog niza*
- Važan odabir "pivota"
  - prvi, zadnji, srednji(?)
- Pivot – od franc.: središnja ili centralna točka, referenca, ishodište

Quicksort

- Nakon odabora pivota provjeravaju se svi preostali elementi tog niza i ako su manji od njega, a desno ili veći od njega, a lijevo, mijenjaju mjesta s pivotom
- Algoritam koristi paradigmu podijeli-pa-vladaj
- Algoritam je brz, ali složen za analizu
- Vremenska složenost quicksort algoritma u najgorem slučaju je $O(n^2)$ ali brzina algoritma se strahovito ubrzava u preostalim slučajevima gdje mu je složenost reda $O(n\ ln(n))$

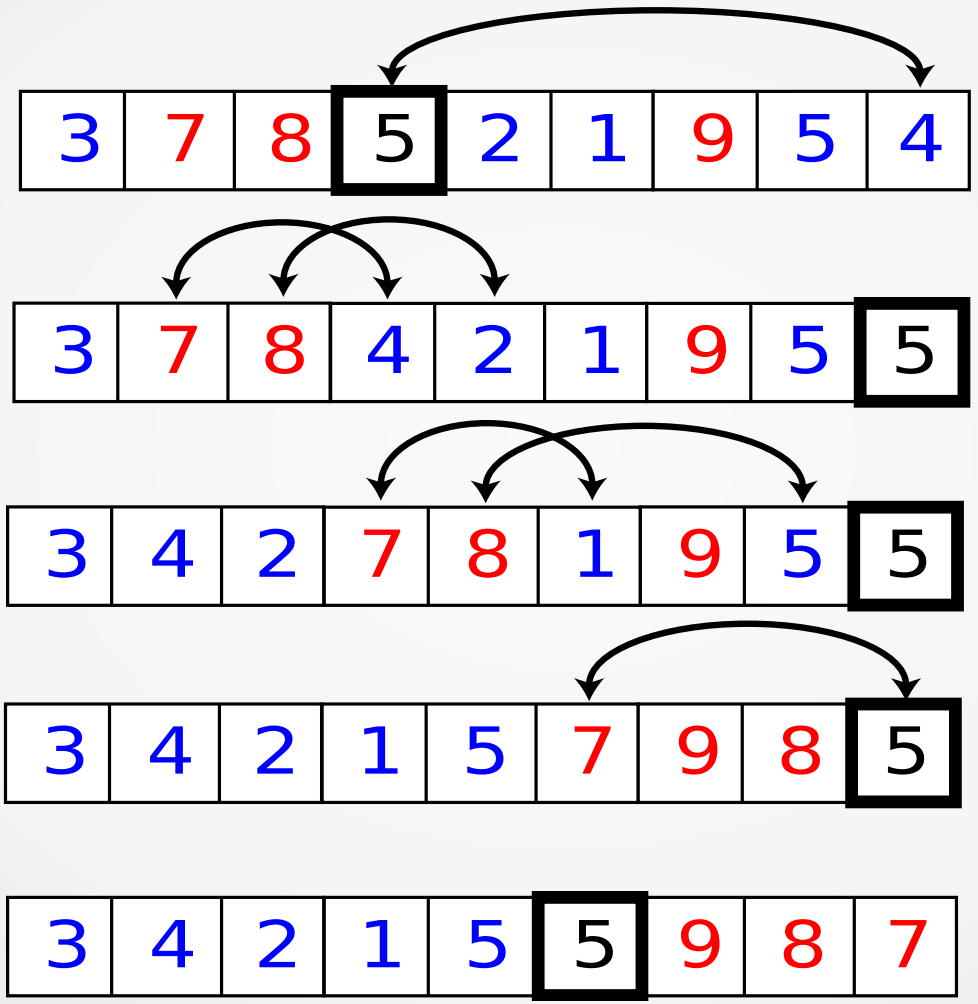
Quicksort

```
quicksort(A, i, k):
  if i < k:
    p := partition(A, i, k)
    quicksort(A, i, p - 1)
    quicksort(A, p + 1, k)
```

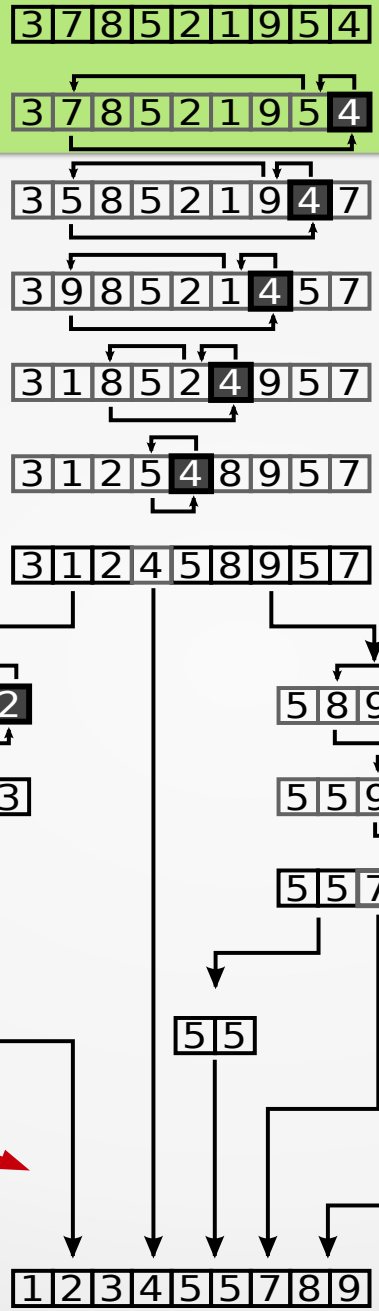
Quicksort

```
// L - indeks najlievijeg elementa
// D - indeks najdesnijeg elementa
// br. el. u nizu D-L+1
partition(niz, L, D)
    pivotIndex := odaberiPivota(niz, L, D) // pivotIndex := D
    pivotValue := niz[pivotIndex]
    swap niz[pivotIndex] and niz[D]
    storeIndex := L
    for i from L to D – 1
        // sve manje od pivota stavi na početak
        if niz[i] < pivotValue
            swap niz[i] and niz[storeIndex]
            storeIndex := storeIndex + 1
    swap niz[storeIndex] and niz[D]  // pomakni pivota
    return storeIndex
```

Quicksort



Quicksort



Podijeli-pa-vladaj

Drugačija "partition" funkcija!!!

Spajanje (merge)

Quicksort

```
static void QuickSort(int[] niz, int n)
{
    QuickPomocna(niz, 0, n - 1);
}

static void QuickPomocna(int[] niz, int prvi, int zadnji)
{
    if (prvi < zadnji)
    {
        int pivot = Particija(niz, prvi, zadnji);
        QuickPomocna(niz, prvi, pivot - 1);
        QuickPomocna(niz, pivot + 1, zadnji);
    }
}
```

Quicksort

```
static int Particija(int[] niz, int prvi, int zadnji)
{
    int pivot = prvi;
    while (prvi < zadnji)
    {
        if (niz[prvi] <= niz[zadnji])
            Zamjena(ref niz[pivot++], ref niz[prvi]);
        ++prvi;
    }
    Zamjena(ref niz[pivot], ref niz[zadnji]);
    return pivot;
}
```

Sortiranje usporedbom

- Eng. Comparison sort
- Nekoliko najpoznatijih:
  - Bubble (ili exchange) sort - sortiranje izmjenom
  - Quicksort
  - Merge sort (sortiranje spajanjem)
  - Shell sort (Shellovo sortiranje)
  - Sortiranje umetanjem

Sortiranje spajanjem

Sortiranje spajanjem

- Spajanje podrazumijeva kombiniranje 2 ili više nizova u jedan uređeni niz.
- Na primjer spajanje nizova 503 703 765 i 087 512 677 u uređeni niz 087 503 512 677 703 765
- Najjednostavniji način za ovo je uspoređivati dva najmanja elementa iz tih nizova te kao rezultat izbaciti manji od njih dva

Sortiranje spajanjem

$$\left\{\begin{matrix} 503 & 703 & 765 \\ 087 & 512 & 677 \end{matrix}\right.$$

$$087 \left\{\begin{matrix} 503 & 703 & 765 \\ 512 & 677 & \end{matrix}\right.$$

$$087\ \ 503 \left\{\begin{matrix} 703 & 765 \\ 512 & 677 \end{matrix}\right.$$

$$087\ \ 503\ \ 512 \left\{\begin{matrix} 703 & 765 \\ 677 & \end{matrix}\right.$$

Sortiranje spajanjem

- Merge sort koristi navedeni princip
  - odatle mu i ime
  - Promotrimo kako izgleda implementacija spajanja
    - Spojimo (merge) dva niza

Sortiranje spajanjem

```
static void merge ( int[] niz, int first, int mid, int last )
{
    int i = first, j = mid, k = 0;
    int[] save = new int[last-first];
    while ( i < mid && j < last )
    {
        if ( niz[i] <= niz[j] )
            save[k++] = niz[i++];
        else
            save[k++] = niz[j++];
    }
    while ( i < mid )
        save[k++] = niz[i++];
    while ( j < last )
        save[k++] = niz[j++];
    for ( i = 0; i < ( last - first ); i++ )
        niz[first + i] = save[i];
}
```

Sortiranje spajanjem

- Merge sort koristi i tehniku podijeli-pa-vladaj
- Polazni niz se rekurzivno komada na dva niza po pola, sve dok duljina nastalog niza nije 1, što je trivijalno za sortirati
- Nakon toga, spajanjem se formiraju novi podnizovi – naravno, sortirani
- Promotrimo kako izgleda implementacija toga (rekurzivna)

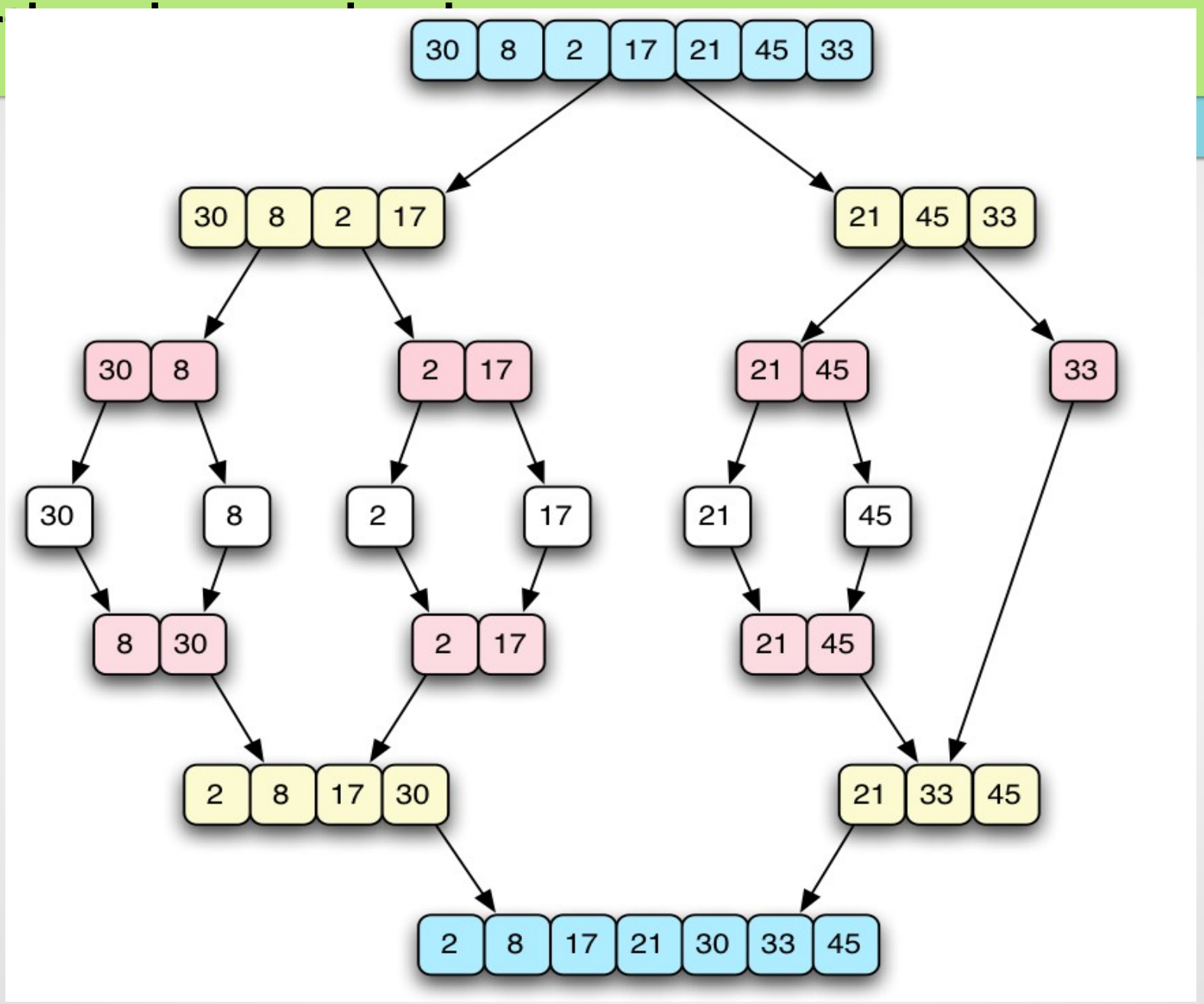
Sortiranje spajanjem

```
static void merge_sort(int[] niz, int first, int last)
{
    if (first < last - 1)
    {
        int mid = (first + last) / 2;
        merge_sort(niz, first, mid);
        merge_sort(niz, mid, last);
        merge(niz, first, mid, last);
    }
}
```

Sortiranje spajanjem

- Vremenska složenost u najgorem slučaju za merge sort je *O(n ln(n))*
- Merge sort je prilično brz algoritam, ali mu je najveći nedostatak to što stalno stvara nove podnizove i na taj način zauzima puno memorije
- Zbog korištenja podijeli-pa-vladaj pristupa, iznimno je prikladan za paraleleno programiranje

Sor


30 8 2 17 21 45 33
30 8 2 17
21 45 33
30 8
2 17
21 45
33
30
8
2
17
21
45
8 30
2 17
21 45
2 8 17 30
21 33 45
2 8 17 21 30 33 45

Sortiranje usporedbom

- Eng. Comparison sort
- Nekoliko najpoznatijih:
  - Bubble (ili exchange) sort - sortiranje izmjenom
  - Quicksort
  - Merge sort (sortiranje spajanjem)
  - Shell sort (Shellovo sortiranje)
  - Sortiranje umetanjem

Shellovo sortiranje

- Generalizacija sortiranja umetanjem (??)
  - Vratit ćemo se na to

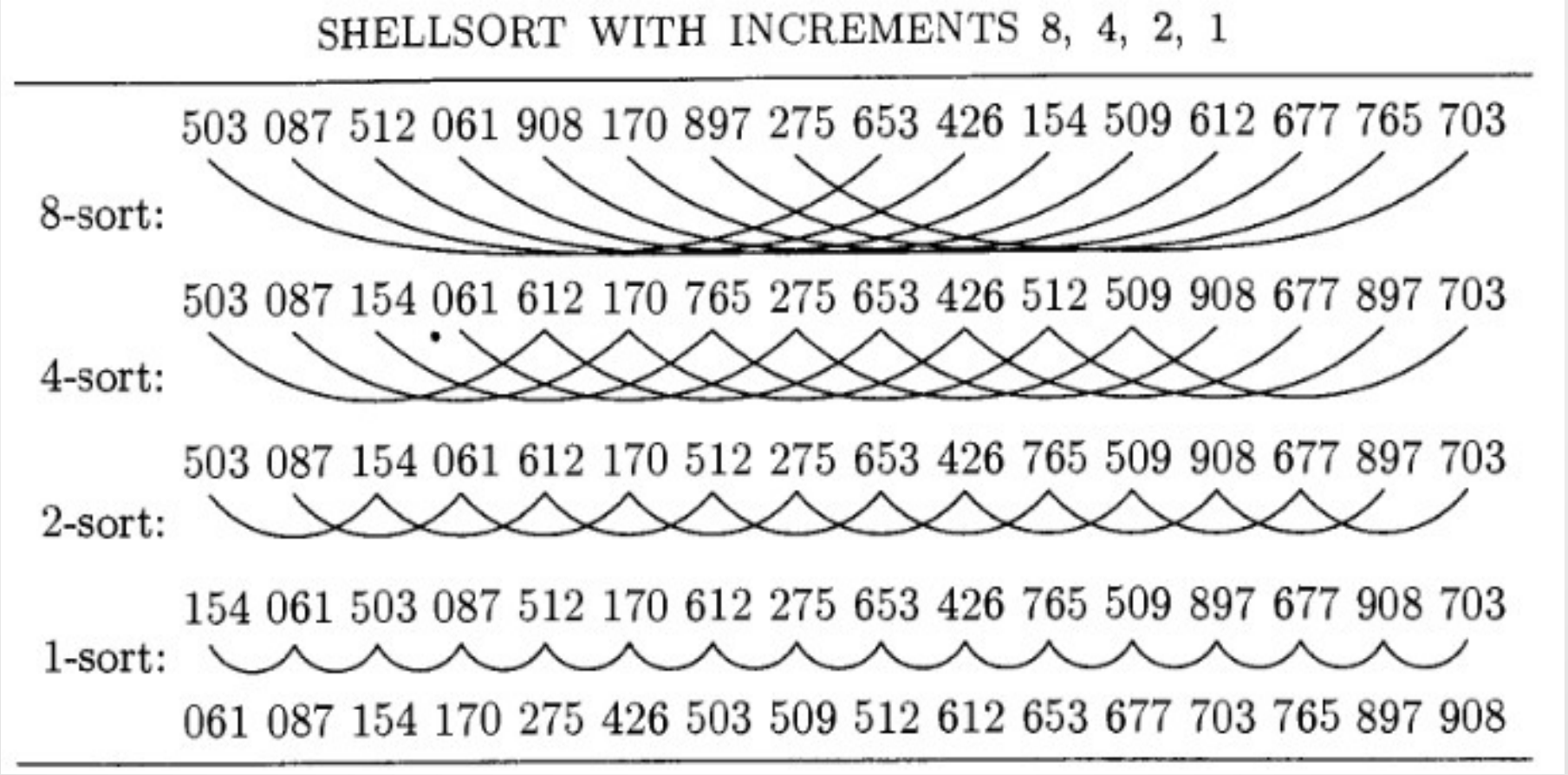
Shellovo sortiranje

- Generalizacija sortiranja umetanjem (??)
  - Vratit ćemo se na to
- Generalizacija sortiranja izmjenom (Bubble sort)
  - Ako je dan algoritam sortiranja koji pomiče elemente samo jednu po jednu poziciju tada će njegovo prosječno vrijeme izvođenja u najbolju ruku biti $O(n^2)$
  - Kako to poboljšati?
    - srpanj 1959. Donald L. Shell

Shellovo sortiranje

- Za unaprijeđenje su potrebni određeni mehanizmi koji pomiču elemente koji su međusobno udaljeni za puno više od jedne pozicije
- Ideja algoritma je jednostavna:
  - promatrati svako k-ti element u nizu
  - tako nastale k podnizove sortirati insertion sort algoritmom
  - smanjiti k i ponoviti postupak
  - kada je k = 1, cijeli niz sortirati
    - Bubble sortom ili
    - Umetanjem (Insertion sort)

Shellovo sortiranje



Shellovo sortiranje

- Očito je da će za izvršenje algoritma trebati manje ili više vremena ovisno o odabiru koraka k kojima se stvaraju podnizovi.
- Iscrpna istraživanja ovog problema su pokazala da je trenutno najbolji niz koraka onaj kojeg definira poljski matematičar Marcin Ciura:

  {1,4,10,23,57,132,301,701,...}

- Ukoliko je potrebno povećati korak, svaki sljedeći je za otprilike 2.3 puta veći od prethodnog.

Shellovo sortiranje

- Algoritam

S1 [Vrti se u petlji po s] Obavi korak S2 za s= t-1;t-2,...,0 a onda završi algoritam

S2 [Vrti se u petlji po j] Postavi $h=h_s$ i obavljaj korake S3 do S6 za $h < j < n$

S3 [Postavi i, K, R] Postavi $i = j-h$; $K=K_j$; $R=R_j$

S4 [Usporedi K sa Ki] Ako je $K \geq K_i$ idi na S6

S5 [Pomakni Ri, smanji i] Postavi $R_{i+h}=R_i$; $i=i-h$ a ako je $i>0$ idi natrag na S4.

S6 [Smjesti R] Postavi $R_{i+h}=R$

Shellovo sortiranje

```
static void ShellSortPomocna(int[] niz, int duljina, int korak)
{
    int i; for (i = 0; i < duljina; i++)
    { /* Umetnimo niz[i] u sortirani podniz */
        int j, v = niz[i];
        for (j = i - korak; j >= 0; j -= korak)
        {
            if (niz[j] <= v) break;
            niz[j + korak] = niz[j];
        }
        niz[j + korak] = v;
    }
}
```

Shellovo sortiranje

```
static void ShellSort(int[] niz, int duljina)
       {
           int[] CiurinNiz = new int[8] { 701, 301, 132, 57, 23, 10, 4, 1 };
           double prosirenje = 2.3;
           int korak_idx = 0;
           int korak = CiurinNiz[0];
           if (duljina > korak)
           {
               while (duljina > korak)
               {
                   korak_idx--;
                   korak = (int)(korak * prosirenje);
               }
           }
           else
           {
               while (duljina < korak)
               {
                   korak_idx++;
                   korak = CiurinNiz[korak_idx];
               }
           }
           while (korak > 1)
           {
               korak_idx++;
               if (korak_idx >= 0)
                   korak = CiurinNiz[korak_idx];
               else
                   korak = (int)(korak / prosirenje);
               ShellSortPomocna(niz, duljina, korak);
           }
       }
```

Shellovo sortiranje

- Nestabilan algortiam sortiranja
- Složenost:
  - U najboljem slučaju O(n log(n))
  - U najgorem slučaju $O(n^2)$
- Zahtjeva nešto više operacija i ima veći "cache miss ratio" od Quicksorta
- Rijetko se koristi danas
  - Iznimka neki ugradbeni (embedded) sustavi

Sortiranje usporedbom

- Eng. Comparison sort
- Nekoliko najpoznatijih:
  - Bubble (ili exchange) sort - sortiranje izmjenom
  - Quicksort
  - Merge sort (sortiranje spajanjem)
  - Shell sort (Shellovo sortiranje)
  - Sortiranje umetanjem

Sortiranje umetanjem

Sortiranje umetanjem

- Algoritmi iz ove skupine podsjećaju na algoritme korištene pri slaganju karata u kartaškim igrama
- Prije razmatranja zapisa R_j
  - pretpostavljamo da su prethodni zapisi R_1, R_2 , ... , R_{j-1} sortirani
  - potom ga umećemo na odgovarajuće mjesto među prethodno sortirane zapise

Sortiranje umetanjem

```
for i ← 1 to length(A)
    j ← i
    while j > 0 and A[j-1] > A[j]
        swap A[j] and A[j-1]
        j ← j - 1
```

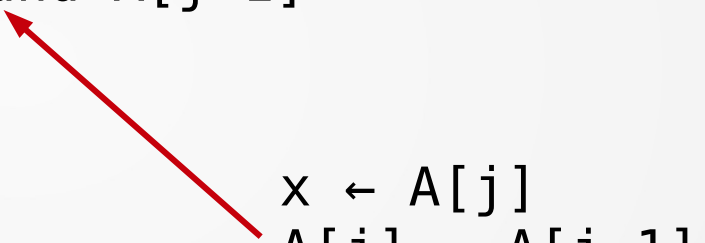
Sortiranje umetanjem

```
for i ← 1 to length(A)
    j ← i
    while j > 0 and A[j-1] > A[j]
        swap A[j] and A[j-1]
        j ← j - 1
```

Zar to nije Bubble s novim elementom?

Sortiranje umetanjem

```
for i ← 1 to length(A)
    j ← i
    while j > 0 and A[j-1] > A[j]
        swap A[j] and A[j-1]
        j ← j - 1
```



```
x ← A[j]
A[j] ← A[j-1]
A[j-1] ← x
```

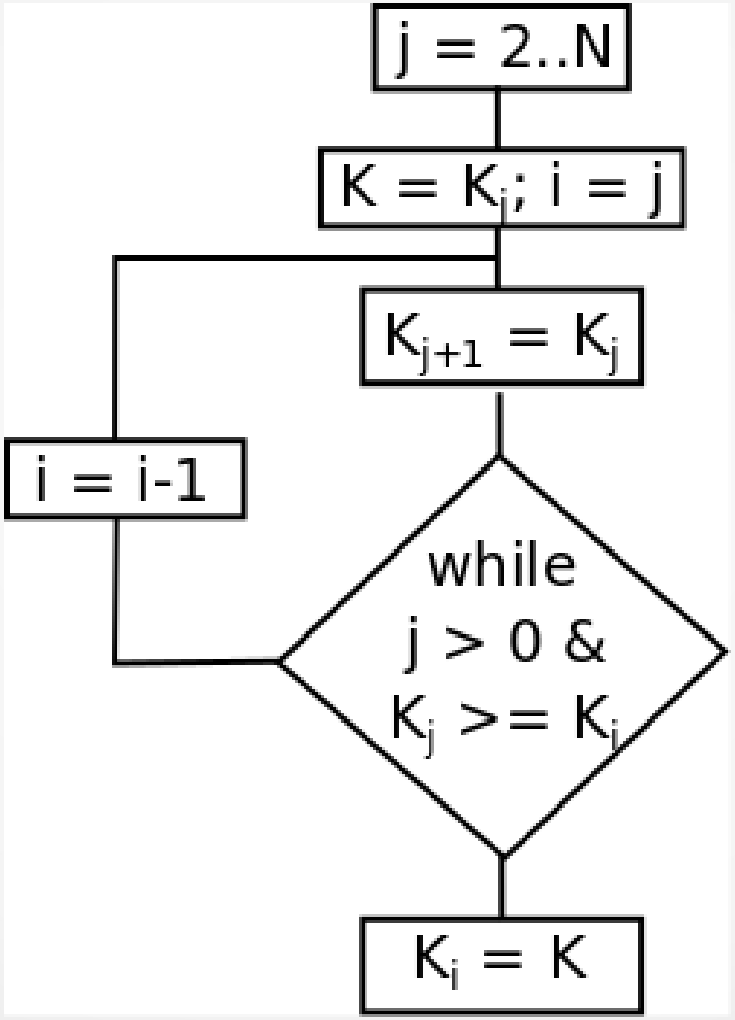
Sortiranje umetanjem

```
for i ← 1 to length(A)
    j ← i
    while j > 0 and A[j-1] > A[j]
        x ← A[j]
        A[j] ← A[j-1]
        A[j-1] ← x
        j ← j - 1
```

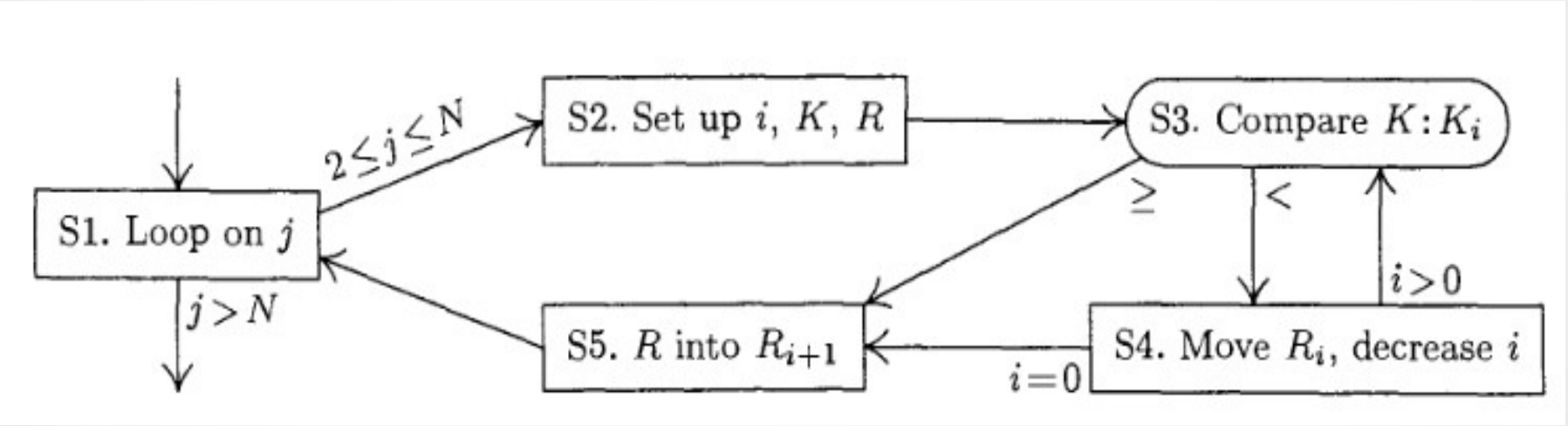
Sortiranje umetanjem

```
for i ← 1 to length(A)
    x ← A[i]
    j ← i
    while j > 0 and A[j-1] > x
        A[j] ← A[j-1]
        j ← j - 1
    A[j] ← x
```

Sortiranje umetanjem



Sortiranje umetanjem



Sortiranje umetanjem

503 087 512 061 908 170 897 275 653 426 154 509 612 677 765 703

EXAMPLE OF STRAIGHT INSERTION

```
^503 : 087
 087  503 :^512
^087  503  512 : 061
 061  087  503  512 :^908
 061  087 ^503  512  908 : 170
 061  087  170  503  512 ^908 : 897
   . . . . . .
 061  087  154  170  275  426  503  509  512  612  653  677 ^765  897  908 : 703
 061  087  154  170  275  426  503  509  512  612  653  677  703  765  897  908
```

Sortiranje umetanjem

- Algoritam

I1 [Vrti se u petlji po j] Obavljaj korake I2 do I5 za $j = 2, 3, K, n$ a potom završi algoritam

I2 [Postavi i, K, R] Postavi $i = j - 1, K = K_j, R = R_j$

I3 [Usporedi K sa Ki] Ako je $K \geq K_i$ idi na I5

I4 [Pomakni Ri, smanji i] Postavi $R_{i+1} = R_i, i = i - 1$ a

I ako je $i > 0$ idi na I3

I5 [Smjesti R] Postavi $R_{i+1} = R$

u sljedećim koracima pokušat ćemo smjestiti R na pravo mjesto, uspoređujući K s Ki za sve padajuće vrijednosti od i

Pronađeno mjesto za R

Ako je i = 0, tada je K najmanji do sada pronađeni ključ, tako da zapis R pripada na mjestu 1

Sortiranje umetanjem

- Ovaj program se izvršava 9B + 10N -3A -9 vremenskih jedinica
  - N = broj zapisa koje treba sortirati
  - A = koliko puta se i smanjuje do 0 u koraku S4
  - B = broj micanja elemenata

Sortiranje umetanjem

- Ovaj program se izvršava 9B + 10N -3A -9 vremenskih jedinica

```
01  START ENT1 2-N
02  2H    LDA  INPUT+N,1
03        ENT2 N-1,1
04  3H    CMPA INPUT,2
05        JGE  5F
06  4H    LDX  INPUT,2
07        STX  INPUT+1,2
08        DEC2 1
09        J2P  3B
10  5H    STA  INPUT+1,2
11        INC1 1
12        J1NP 2B
```

1	*S1. Loop on j.* $j \leftarrow 2$.
$N-1$	*S2. Set up i, K, R.*
$N-1$	$i \leftarrow j-1$.
$B+N-1-A$	*S3. Compare* $K : K_i$.
$B+N-1-A$	To S5 if $K \geq K_i$.
B	*S4. Move* R_i*, decrease* i.
B	$R_{i+1} \leftarrow R_i$.
B	$i \leftarrow i-1$.
B	To S3 if $i > 0$.
$N-1$	*S5.* R *into* R_{i+1}.
$N-1$	
$N-1$	$2 \leq j \leq N$. ∎

Sortiranje umetanjem

```
static void Insertion(int[] niz)
{
      int i;
      for (i = 0; i < niz.Count(); i++)
      {
          /* Umetnimo niz[i] u sortirani podniz */
          int j, v = niz[i];
          for (j = i - 1; j >= 0; j--)
          {
              if (niz[j] <= v) break;
              niz[j + 1] = niz[j];
          }
          niz[j + 1] = v;
      }
}
```

Sortiranje umetanjem

- Vremenska složenost ovog algoritma je $O(n^2)$ za najgori slučaj

Sortiranje usporedbom

- Eng. Comparison sort
- Nekoliko najpoznatijih:
  - Bubble (ili exchange) sort - sortiranje izmjenom
  - Quicksort
  - Merge sort (sortiranje spajanjem)
  - Shell sort (Shellovo sortiranje)
  - Sortiranje umetanjem

Sortiranje bez usporedbe

- Eng. non-comparison sort
- Nekoliko najpoznatijih
  - Radix sort
  - Bucket sort
  - Counting sort

Sortiranje bez usporedbe

- Eng. non-comparison sort
- Nekoliko najpoznatijih
  - Radix sort
  - Bucket sort
  - Counting sort

Radix sort

- Ne uspoređuje elemente već njihove "znamenke"
  - Grupira elemente
  - Počinje s usporedbom jedinica, pa desetica, pa tisućica, ...
- "Znamenka" u digitalnom (računalnom svijetu)
- Vremenska složenost u najgorem slučaju za radix sort je $O(kn)$ gdje je k broj znamenaka najvećeg broja

Radix sort

- Ne uspoređuje elemente već njihove "znamenke"
  - Grupira elemente
  - Počinje s usporedbom jedinica, pa desetica, pa tisućica, ...
- "Znamenka" u digitalnom (računalnom svijetu)
- Vremenska složenost u najgorem slučaju za radix sort je *O(kn)* gdje je k broj znamenaka najvećeg broja
  - k nije konstanta? k=log(n)?

Radix sort

```
radix
odredi koliko znamenaka ima max broj = k
petlja po i od 1 do k
     formiraj niz znamenaka po  evši s jedinicama (1) pa s
     deseticama (2) do k
     sortiraj taj niz znamenaka
     petlja po j od 0 do n-1 po nizu brojeva
       šetaj paralelno po nizu znamenaka i nizu brojeva
       traži broj sa odgovaraju  om znamenkom i zamijeni ga s
       trenutnom (1, 2, 3...)
```

Radix sort

Početak	1. prolaz	2. prolaz	3. prolaz	Kraj
503	170	503	061	061
087	061	908	087	087
512	512	509	154	154
061	612	703	170	170
908	503	512	275	275
170	653	612	426	426
897	703	426	503	503
275	154	653	509	509
653	275	154	512	512
426	765	061	612	612
154	426	765	653	653
509	087	170	677	677
612	897	275	703	703
677	677	677	765	765
765	908	087	897	897
703	509	897	908	908

Pitanja

ime.prezime@izor.hr