Sortiranje spajanjem (Merge Sort)

- Algoritam sortiranja spajanjem:

  ulaz je lista $a_1, \ldots, a_n$, izlaz sortirana lista

  1. podijeli listu na dva jednaka dijela
  2. sortiraj listu $a_1, \ldots, a_{n/2}$
  3. sortiraj listu $a_{n/2+1}, \ldots, a_n$
  4. spoji liste $a_1, \ldots, a_{n/2}$ i $a_{n/2+1}, \ldots, a_n$

- Algoritam spajanja:

  ulaz su dvije sortirane liste $b_1, \ldots, b_k$ i $c_1, \ldots, c_l$, izlaz je sortirana lista $a_1, \ldots, a_{k+l}$

  1. $i = 1, j = 1$
  2. ponavljaj korak 3 sve dok je $i \leq k$ i $j \leq l$
  3. ako je $b_i < c_j$ onda $a_{i+j-1} = b_i$, $i=i+1$, inače $a_{i+j-1} = c_j$, $j=j+1$
  4. ponavljaj korak 5 sve dok je $i \leq k$
  5. $a_{i+j-1} = b_i$, $i=i+1$
  6. ponavljaj korak 7 sve dok je $j \leq l$
  7. $a_{i+j-1} = c_j$, $j=j+1$

- Složenost ovog algoritma je $O(n \lg n)$
- Nedostatak: potrebno pomoćno polje

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
typedef int tip;

// udruzivanje LPoz:LijeviKraj i DPoz:DesniKraj
void Merge (tip A [], tip PomPolje [], int LPoz, int DPoz, int DesniKraj) {
  int i, LijeviKraj, BrojClanova, PomPoz;
  LijeviKraj = DPoz - 1;
  PomPoz = LPoz;
  BrojClanova = DesniKraj - LPoz + 1;
  while (LPoz <= LijeviKraj && DPoz <= DesniKraj) { // glavna petlja
    if (A [LPoz] <= A [DPoz])
      PomPolje [PomPoz++] = A [LPoz++];
    else
                          PomPolje [PomPoz++] = A [DPoz++];
  }
  while (LPoz <= LijeviKraj)    // Kopiraj ostatak prve polovice
    PomPolje [PomPoz++] = A [LPoz++];
  while (DPoz <= DesniKraj)    // Kopiraj ostatak druge polovice
     PomPolje [PomPoz++] = A [DPoz++];
  for (i = 0; i < BrojClanova; i++, DesniKraj--)    // Kopiraj PomPolje natrag
    A [DesniKraj] = PomPolje [DesniKraj];
}
```

```
// MergeSort - rekurzivno sortiranje podpolja
void MSort (tip A [], tip PomPolje[], int lijevo, int desno ) {
  int sredina;
  if (lijevo < desno) {
    sredina = (lijevo + desno) / 2;
    MSort (A, PomPolje, lijevo, sredina);
    MSort (A, PomPolje, sredina + 1, desno);
    Merge (A, PomPolje, lijevo, sredina + 1, desno);
  }
}

// MergeSort - sort udruzivanjem
void MergeSort (tip A [], int N) {
  tip *PomPolje;
  PomPolje = malloc (N * sizeof (tip));
  if (PomPolje != NULL) {
    MSort (A, PomPolje, 0, N - 1);
    free (PomPolje);
  } else {
            printf ("Nema mjesta za PomPolje!");
            exit(1);}
}
```

```
void main (void) {
  tip *Polje1,maxcl;
  int Duljina,brojac;
      printf ("Unesi broj clanova polja >");
      scanf ("%d", &Duljina);
      if ((Polje1 = (tip *) malloc (Duljina * sizeof (tip)))== NULL) {
              printf("\nNema dovoljno memorije!\n");
              exit(1); }
      srand(time(NULL));
      printf("\nUnesi maksimalnu vrijednost clanova>");
      scanf("%d", &maxcl);
      for (brojac=0; brojac<Duljina; brojac++){
              Polje1[brojac]=(tip) maxcl * ((float)rand() / (RAND_MAX + 1));  }
    printf("\n Prije sortiranja:\n");
    for (brojac=0; brojac<Duljina; brojac++)
      printf("Polje[%d]=%d  ",brojac,Polje1[brojac]);
    printf("\n");
      MergeSort(Polje1,Duljina);
      printf("\n Nakon sortiranja:\n");
      for (brojac=0; brojac<Duljina; brojac++)
              printf("Polje[%d]=%d  ",brojac,Polje1[brojac]);
      printf("\n");
     system("PAUSE");
      exit(0);
 }
```

Sortiranje Shellovim algoritmom (Shell Sort)

- Nazvan po D. L. Shellu koji ga je napravio 1959.
- Shellovo sortiranje je vrsta algoritma sortiranja koje, u prosjeku, treba manje od $O(n^2)$ operacija uspoređivanja i zamjene elemenata
- Jednostavan algoritam, ali vrlo kompliciranо određivanje složenosti algoritma
- Ovaj algoritam je poboljšana verzija sortiranja umetanjem (insertion sort) ili mjehuričastog sortiranja (bubble sort)
- Ta dva algoritma su efikasna ako su nizovi gotovo sortirani, a neefikasni jer pomiču elemente, uglavnom, za po jedno mjesto
- Shellovo sortiranje radi tako da sortira u većim koracima koji se postupno smanjuju na korak od jedne pozicije, ali do tada je niz već gotovo sortiran, pa je i sortiranje umetanjem/mjehuričasto sortiranje efikasno
- Ideja Shellovog sortiranja: prvo se niz podataka složi u dvodimenzionalno polje, a zatim se stupci sortiraju
- Time se dobije djelomično sortirano polje
- Proces se ponavlja tako da u svakom koraku polje ima sve manje stupaca, u zadnjem koraku ostaje samo jedan stupac u kojem su podaci sortirani

- Svakim korakom sortiranost niza je sve veća
- Broj operacija sortiranja u svakom koraku je ograničen, jer je niz već djelomično sortiranim u prethodnim koracima
- Primjer: zadan niz 3 7 9 0 5 1 6 8 4 2 0 6 1 5 7 3 4 9 8 2, složi se u npr. polje od 7 stupaca i stupci se sortiraju:

3 7 9 0 5 1 6		3 3 2 0 5 1 5
8 4 2 0 6 1 5	->	7 4 4 0 6 1 6
7 3 4 9 8 2		8 7 9 9 8 2

- U slijedećem koraku se podaci preslože u polje s 3 stupca:

3 3 2		0 0 1
0 5 1		1 2 2
5 7 4	- >	3 3 4
4 0 6		4 5 6
1 6 8		5 6 8
7 9 9		7 7 9
8 2		8 9

- U završnom koraku se dobije 1 stupac gotovo sortiranih podataka (prvih 11 elemenata je već sortirano) koji se lako sortira (pomiču se samo 3 elementa)

- U praktičnom algoritmu se koristi jednodimenzionalno polje, uz adekvatno indeksiranje
- Npr. elementi na pozicijama 0, 5, 10, 15, … tvore prvi red; svaki tako dobiveni stupac se sortira algoritmom sortiranja umetanjem
- Izbor koraka indeksiranja (broja stupaca u 2D polju koji se sortiraju) određuje ukupan broj operacija u sortiranju niza, dakle i vrijeme izvršavanja algoritma
- Primjer funkcije (originalna Shellova):

```
void ShellSort (int a[] , int n) {
 int i, j, k, h, v;
 int cols[] = {1391376, 463792, 198768, 86961, 33936, 13776, 4592, 1968, 861, 336, 112, 48, 21, 7,
     3, 1} ;
 for (k=0; k<16; k++) {
 h=cols[k];
  for (i=h; i<n; i++) {
       v=a[i]; j=i;
       while (j>=h && a[j-h]>v) {
           a[j]=a[j-h]; j=j-h; }
a[j]=v; } } }
```

- Drugi primjer izbora koraka: uzet ćemo da je broj stupaca N/2 i u svakom koraku raspolovimo taj broj:

```
void ShellSort2 (tip A [], int N) {
  int i, j, korak;
  tip pom;
  for (korak = N / 2; korak > 0; korak /= 2) {    // Insertion Sort s većim korakom
    for (i = korak; i < N; i++) {
      pom = A [i];
      for (j = i; j >= korak && A[j-korak] > pom; j -= korak) {
        A [j] = A [j - korak];      }
      A [j] = pom;    }
  }
}

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
typedef int tip;
```

```
void main(){
     tip *polje1, *polje2;
     int bel, ind;
     printf ("Unesi broj clanova polja:");
     scanf ("%d", &bel);
     if (!(polje1 = (tip *) malloc (bel * sizeof (tip))))
              exit(1);
     if (!(polje2 = (tip *) malloc (bel * sizeof (tip))))
              exit(1);
     srand ((unsigned) time (NULL));
     for( ind = 0; ind < bel; ind++ ) {
        polje1[ind]=rand()%1000;
         polje2[ind]=polje1[ind];
        printf("%d  ",polje1[ind]); }
        printf("\n\n");
   ShellSort(polje1, bel);
   for( ind = 0; ind < bel; ind++ )
        printf("%d  ",polje1[ind]);
   printf("\n");
   ShellSort2(polje2, bel);
   for( ind = 0; ind < bel; ind++ )
        printf("%d  ",polje2[ind]);
  printf("\n");
   exit(0);}
```

Algoritam brzog sortiranja (Quicksort)

- Najbrži poznati algoritam sortiranja, koji u prosjeku treba $\Theta(n \lg n)$ operacija uspoređivanja za sortiranje niza duljine n
- U najgorem slučaju složenost mu je $\Theta(n^2)$
- To je algoritam strategije podijeli pa vladaj, koji dijeli niz u dva podniza tako da izabere poseban element pivot (stožer) i onda preuredi niz tako da se svi elementi manji od pivota prebace na pozicije prije njega, a svi veći se nalaze iza pivot elementa, nakon ovog koraka pivot se nalazi na mjestu na kojem mora biti u sortiranom nizu
- Postupak se rekurzivno ponavlja za ta dva podniza (manji i veći od pivota), sve dok se ne dođe do podniza od 1 elementa
- Algoritam brzog sortiranja uspoređuje elemente u nizu pa spada u algoritme sortiranja uspoređivanjem
- rezultat je pokušaja ubrzavanja faze spajanja u algoritmu sortiranja spajanjem (Merge Sort); ovdje je spajanje u potpunosti izbjegnuto, jer nakon što su elementi u podnizovima sortirani, zbog uređenosti polja, svi su elementi iz drugog podniza veći od svakog elementa iz prvog podniza
- Izvedba algoritma ovisi o izboru pivot (stožer) elementa: najjednostavnija varijanta uzima prvi element za pivot, no on se može odrediti na bilo koji način koji se može izračunati u O(1) vremenu

- Razmotrimo slučaj kad je pivot prvi element u nizu. Za premještanje elemenata niza potrebna su dva kursora, prvi je iniciran na drugi element u nizu i raste, a drugi na zadnji i pada
- Kreće se s prvim kursorom koji se pomiče sve dok se ne nađe element veći od pivota, tada se kreće od drugog kursora koji se pomiče dok se ne nađe element manji od kursora
- Zamijene se mjesta ta dva elementa i nastavlja se traženje od prvog kursora
- Pretraživanje završava kad je prvi kursor iza drugoga, tada drugi kursor pokazuje na poziciju na kojoj će u sortiranom nizu biti pivot element, pa se zamijene mjesta tog elementa i pivot elementa
- Nakon toga se ponavlja postupak na dva podniza: za elemente prije pivot elementa i one iza pivot elementa
- Primjer: niz 7 3 5 1 8 4 2 9 6 , pivot je 7
- Kreće se od početka, prvi element veći od 7 je 8, sa stražnje strane prvi manji je 6, pa im se zamijene mjesta:

  7 3 5 1 6 4 2 9 8
- Sljedeći element veći od 7 je 9, odostraga prvi manji je 2. No drugi kursor je ispred prvog pa ovaj korak završava. Niz se dijeli na dva dijela, od početka do drugog kursora, te od prvog kursora do kraja

7 3 5 1 6 4 2 || 9 8

- Pivot i element na koji pokazuje drugi kursor zamijene mjesta, kako su svi elementi u prvoj listi manji od njega, on je na pravom mjestu i više ga se ne dira

  2 3 5 1 6 4 || 7 || 9 8

- Ponavljanje postupka na prvom podnizu: pivot je 2, sprijeda prvi veći od njega je 3, straga prvi manji je 1

  2 1 5 3 6 4 || 7 || 9 8

- Nastavljamo: sprijeda 5, straga 1, ali je prvi kursor iza drugoga, pa se mijenjaju pivot i element na koji pokazuje drugi kursor

  1 || 2 || 5 3 6 4 || 7 || 9 8

- Drugi podniz (zadnja 2 elementa): pivot je 9 prvi kursor ne nalazi veći od njega nego prelazi drugi kursor, pa 9 i 8 mijenjaju mjesta:

  1 || 2 || 5 3 6 4 || 7 || 8 || 9

- Jedini preostao niz je u sredini: pivot je 5, sprijeda nalazimo 6, straga 4

  1 || 2 || 5 3 4 6 || 7 || 8 || 9

- Sprijeda nalazimo 6 i sad je prvi kursor iza drugog: 5 i 4 zamijene mjesta

1 || 2 || 4 3 || 5 || 6 || 7 || 8 || 9

- Zadnji korak: niz od 2 elementa, pivot je 4, pretraga sprijeda ne nalazi veći od njega nego se prvi kursor pomiče iza drugog:

  1 || 2 || 3 || 4 || 5 || 6 || 7 || 8 || 9

- Algoritam brzog sortiranja: ulaz je lista a_1, … a_n i kursori *i* (početak) i *j* (kraj),a izlaz sortirana lista

  1. k = i+1, l = j
  2. sve dok je k <= l radi korake 3 – 5
  3. dok je ((k<=l) i (a_i >= a_k)) k = k+1
  4. dok je ((k<=l) i (a_i <= a_l)) l = l-1
  5. ako je k < l, zamjeni(a_k, a_l)
  6. ako je l > i zamjeni(a_i, a_l)
  7. ako je l > i quicksort(a, i, l-1)
  8. ako je k < j quicksort(a, k, j)

- Najgori slučaj za ovaj algoritam: već sortirana lista i lista sortirana obrnutim rasporedom. Tada se u svakom koraku lista dijeli na podliste od jednog i n-1 elemenata

```
void quick(tip polje[], int l, int d) {
    int i,j;
    tip pom;
    i = l+1;    j = d;
    if (l>=d) return;
    while ((i <= j) && (i<=d) && (j>l)) {
        while (polje[i] <= polje[l]) i++;
        while (polje[j] > polje[l]) j--;
        if (i<j) {
            pom = polje[i];          polje[i] = polje[j];          polje[j] = pom;       }
  }
    if (i>d) { // stozer je najveci u polju
        pom = polje[d];        polje[d] = polje[l];       polje[l] = pom;
        quick(polje, l, d-1);    }
    else if (j<=l) { // stozer je najmanji u polju
        quick(polje, l+1, d);    }
    else { // stozer je negdje u sredini
        pom = polje[j];        polje[j] = polje[l];       polje[l] = pom;
        quick(polje, l, j-1);
        quick(polje, j+1, d);    }
 }
```

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
typedef int tip;

void main(){
      tip *polje;
      int bel, ind;
      printf ("Unesi broj clanova polja:");
      scanf ("%d", &bel);
      if (!(polje = (tip *) malloc (bel * sizeof (tip))))
              exit(1);
      srand ((unsigned) time (NULL));
      for( ind = 0; ind < bel; ind++ ) {
         polje[ind]=rand()%100;
         printf("%d  ",polje[ind]); }
         printf("\n\n");
    quick(polje,0,bel-1);
    for( ind = 0; ind < bel; ind++ )
         printf("%d  ",polje[ind]);
    system("PAUSE");
    exit(0);
    }
```

- Optimizacija algoritma: bolji izbor pivot elementa
- Može se uzeti medijan od prvog, srednjeg i zadnjeg elementa (usporede se ta tri i izabere se srednji među njima za pivot)
- Najbolja izvedba algoritma: za indeks pivot elementa se uzima slučajni broj, to je algoritam slučajnog brzog sortiranja (randomized quicksort)
- Druga mogućnost optimizacije je upotreba nekog drugog algoritma sortiranja za kratke podnizove, npr. kad se u dijeljenju niza dođe do podnizova od recimo 5 elemenata, njihovo sortiranje se izvede sortiranjem umetanjem (ili mjehuričastim sortiranjem)
- Primjer za složeniju verziju koda: za pivot se uzima srednji po veličini od elemenata koji su na prvom, srednjem i zadnjem mjestu u nizu, a sortiranje podnizova kraćih od četiri elementa se provodi algoritmom sortiranja umetanjem

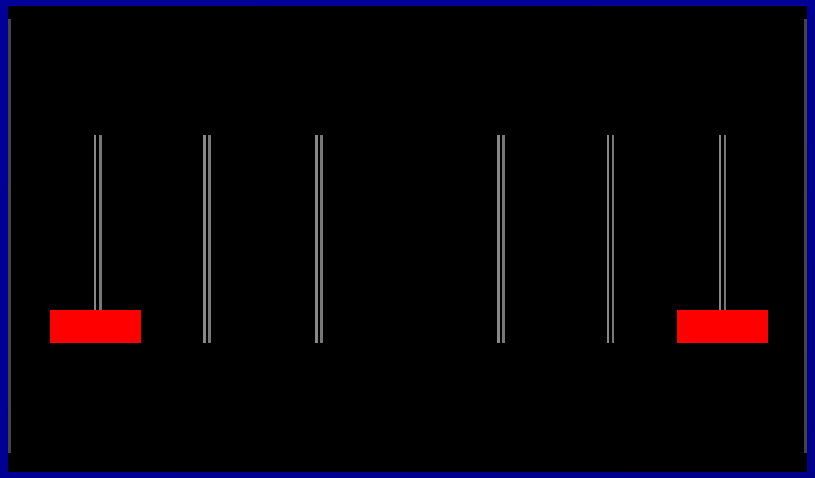
```
#define Cutoff (3)

 void Zamijeni (tip *lijevo, tip *desno) {// zamjena vrijednosti *lijevo i *desno
  tip pom = *lijevo;
  *lijevo = *desno;
  *desno = pom;
}

// QuickSort - medijan i stozer,Vrati medijan od lijevo, sredina i desno, poredaj ih i sakrij stozer
tip medijan3 (tip A [], int lijevo, int desno) {
  int sredina = (lijevo + desno) / 2;
  if (A [lijevo] > A  [sredina])
    Zamijeni (&A[lijevo], &A[sredina]);
  if (A [lijevo] > A [desno])
    Zamijeni (&A [lijevo], &A [desno]);
  if (A [sredina] > A [desno])
    Zamijeni (&A [sredina], &A [desno]);
  // Sada je: A[lijevo]<=A[sredina]<=A[desno]  , Sakrij stozer
  Zamijeni (&A [sredina], &A [desno - 1]);  // Vrati stozer
  return A [desno - 1];
}
```

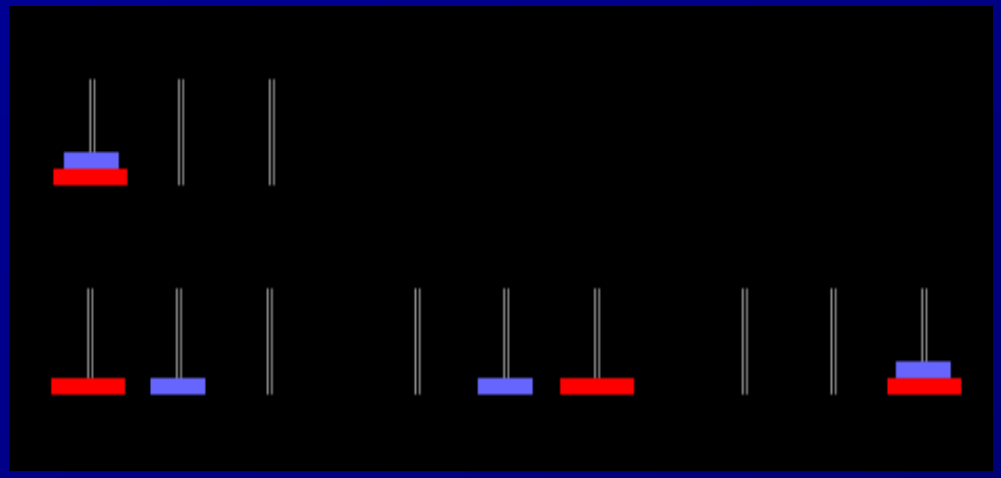
```
void Qsort (tip A [], int lijevo, int desno) {// QuickSort - rekurzivno sortiranje podpolja
  int i, j;
  tip stozer;
  if (lijevo + Cutoff <= desno) {
    stozer = medijan3 (A, lijevo, desno);
    i = lijevo; j = desno - 1;
    while (1) {
      while (A [++i] < stozer);
      while (A [--j] > stozer);
      if (i < j)
        Zamijeni (&A [i], &A [j]);
      else
        break;    }
Zamijeni (&A [i], &A [desno - 1]); // Obnovi stozer
    Qsort (A, lijevo, i - 1);
    Qsort (A, i + 1, desno);
  } else {    // Sortiraj podpolje
    InsertionSort (A + lijevo, desno - lijevo + 1);
  }}

// QuickSort
void QuickSort (tip A [], int N) {
  Qsort (A, 0, N - 1); }
```

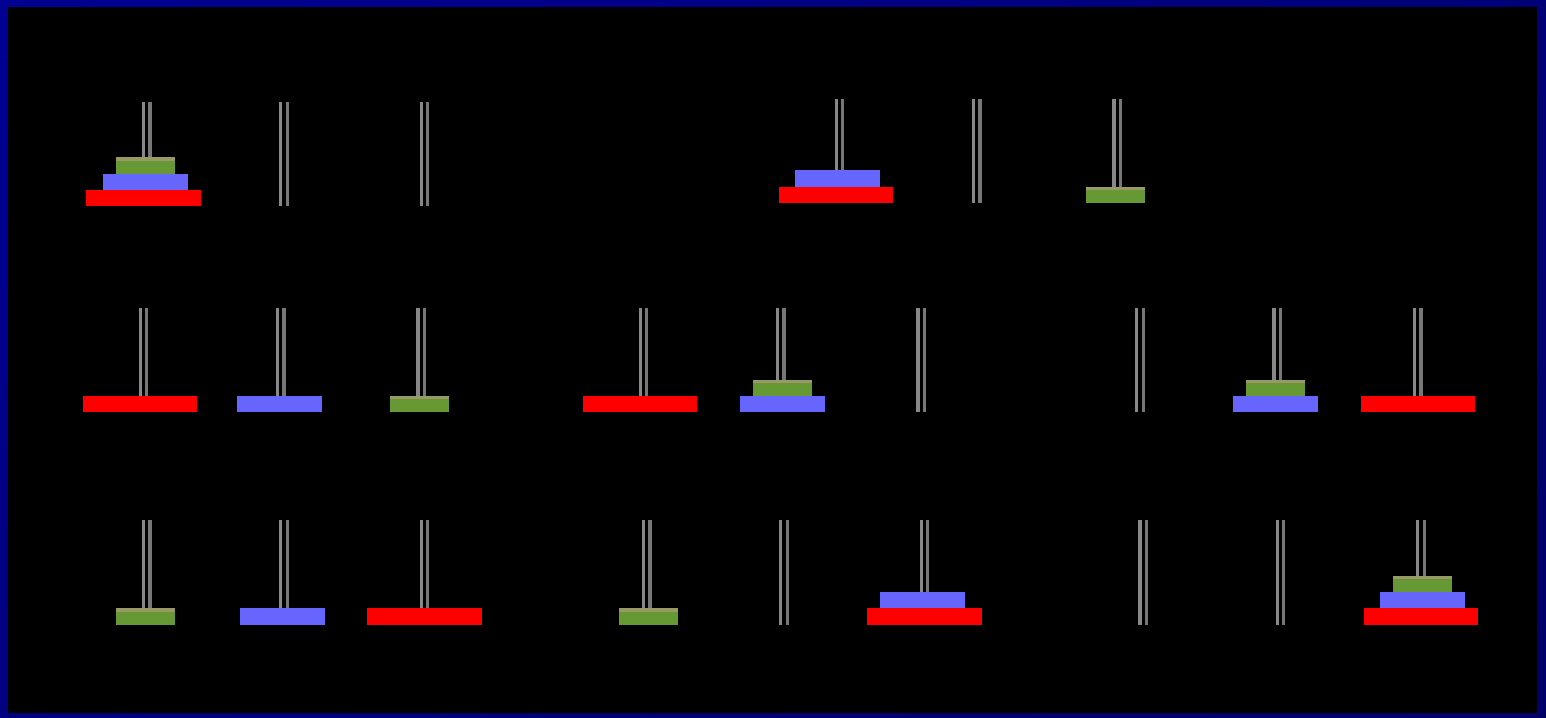
```
void main(){
      tip *polje;
      int bel, ind;
      printf ("Unesi broj clanova polja:"); scanf ("%d", &bel);
      if (!(polje = (tip *) malloc (bel * sizeof (tip))))  exit(1);
      srand ((unsigned) time (NULL));
      for( ind = 0; ind < bel; ind++ ) {
         polje[ind]=rand()%100;
         printf("%d  ",polje[ind]); }
         printf("\n\n");
    QuickSort(polje,bel);
    for( ind = 0; ind < bel; ind++ )
         printf("%d  ",polje[ind]);
    exit(0);   }

void InsertionSort (tip A [], int N) {// sort ubacivanjem
  int i, j;
  tip pom;
  for (i = 1; i < N; i++) {    pom = A[i];
    for (j = i; j >= 1 && A[j-1] > pom; j--)     A[j] = A[j-1];    A[j] = pom;  }
}
```

Problem hanojskih tornjeva

- Napisati program koji rješava problem hanojskih tornjeva:
- Štapovi S (*source*, izvor), D (*destination*, odredište), T(*temp*, pomoćni)
- Na prvom štapu (S) ima n diskova različite veličine postavljenih tako da veći nikad ne dolazi iznad manjeg. Preseliti sve diskove na D, **jedan po jedan**, uvijek postavljajući manji na veći
- Problem je zadao francuski matematičar Edouard Lucas 1883 godine
- Školski primjer uspjeha rekurzivnog postupka
- **Algoritam rješenja:**
- Ignorirati donji (najveći) disk i riješiti problem za n-1 disk, ali sa štapa S na štap T koristeći D kao pomoćni.
- Sada se najveći disk nalazi na S, a ostalih n-1 na T.
- Preseliti najveći disk sa S na D.
- Preseliti n-1 disk sa T na D koristeći S kao pomoćni (problem je već riješen za n-1 disk).







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

void hanoii(char src, char dest, char tmp, int n) {
   if (n > 0) {
      hanoii(src, tmp, dest, n - 1);
      printf("\nPrebacujem element %d s tornja %c na toranj %c", n, src, dest);
      hanoii(tmp, dest, src, n - 1);
   }
}

int main() {
   int n ;
   printf("\nHanojski tornjevi:");
   printf("\n Unesi broj elemenata na tornjevima>");
   scanf("%d",&n);
   hanoii('S', 'D', 'T', n);
   system("PAUSE");
   return 0;
}
```

- Analiza algoritma: označimo s T_N minimalan broj poteza potreban da se riješi problem s N diskova
- za N=3, T_3=7; za N=2, T_2=3; N=1, T_1=1 (T_0=0)
- Izloženi rekurzivni algoritam uključuje dva puta po N-1 pomicanje s jednog štapa na drugi i još jedno završno pomicanje diska. Ovo prebacivanje nije moguće obaviti u manje koraka, jer do trenutka kad je na štapu D ostao samo najdonji disk, potrebno je prebaciti N-1 diskova sa štapa S na štap T, a to se može obaviti u najmanje N-1 prebacivanja. Zatim se u jednom prebacivanju najdonji disk složi na štap D, a da bi se prebacilo N-1 diskova na štap D potrebno je opet najmanje N-1 koraka. Dakle vrijedi:

$$T_N = 2T_{N-1}+1,\ T_0=0,\ T_1=1$$

- uvedemo supstituciju $S_n = T_N + 1$ i slijedi rekurzivna relacija

$$S_n = 2S_{n-1},\ S_0=0$$

- Karakteristična jdba je $x - 2 = 0$ i njeno rješenje je $x = 2$, a rješenje homogene rekurzivne jdbe je

$$S_n = 2^n,\ \text{pa je}$$

$$T_N = 2^N - 1,\ N \geq 0$$

- Dakle, našli smo da vrijeme izvršavanja ovog algoritma raste eksponencijalno s brojem diskova

- Stara legenda: u indijskom hramu stoje 3 stupa s 64 zlatna diska na jednom stupu. Svećenici u hramu imaju zadatak (dan od stvoritelja) da prebace diskove poštujući gornja pravila. Kad uspiju riješiti zadatak, svijet će propasti. Kad bi uspijevali prebacivati diskove brzinom od jednog diska u sekundi i to po algoritmu koji zahtjeva najmanji broj prebacivanja, bilo bi im potrebno 2^{64}-1 sekundi što je 585 milijardi godina (današnja starost svemira je oko 14.5 milijardi godina).