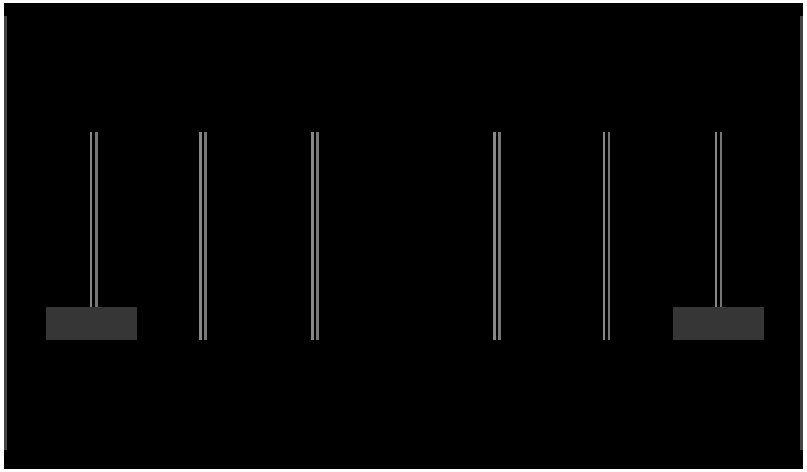
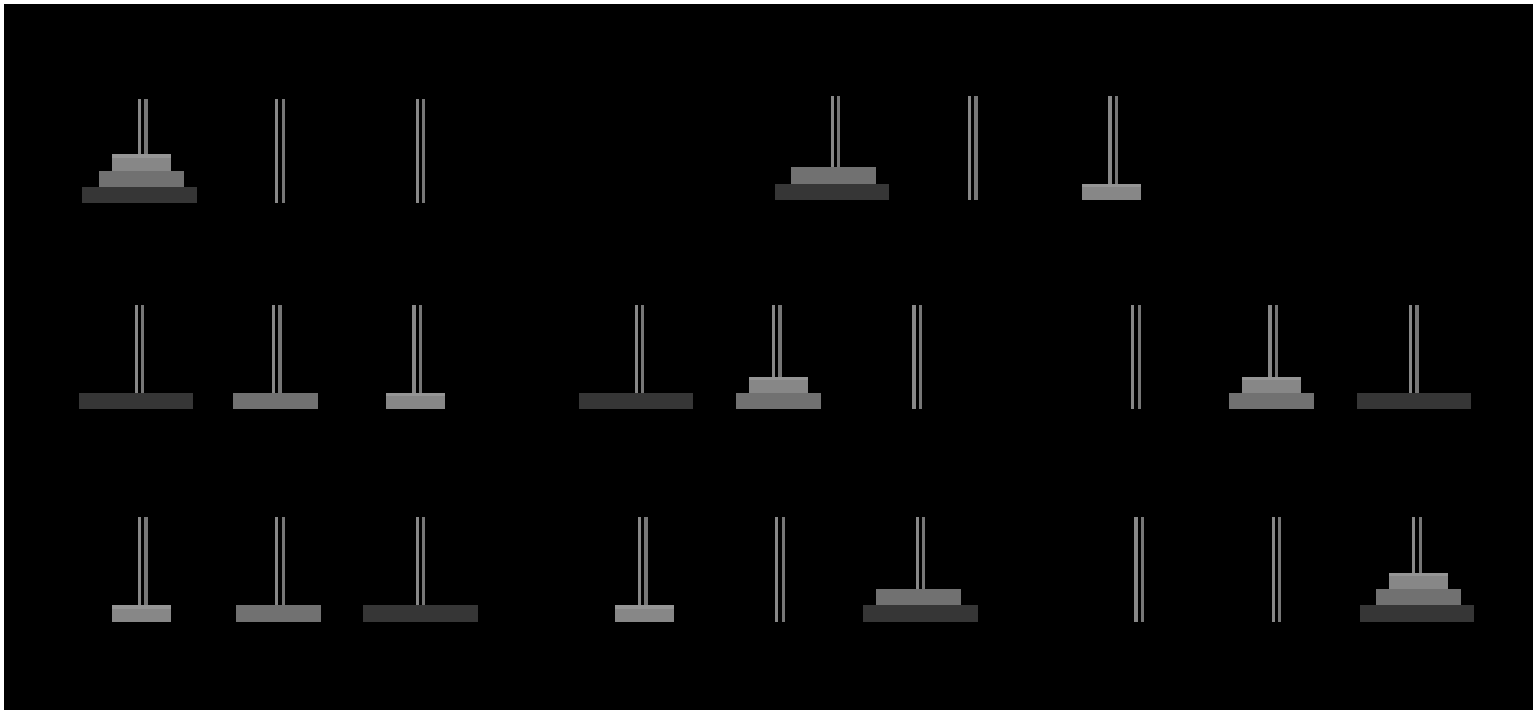
Problem hanojskih tornjeva

- 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).

Problem n kraljica (s predavanja)

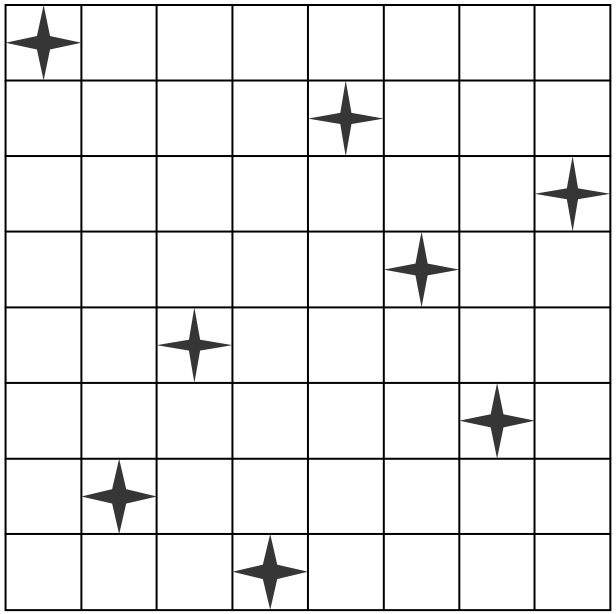
- Problem: na šahovsku ploču veličine n x n polja treba postaviti n kraljica tako da se one međusobno ne napadaju
- Postupak rješavanja: očito svaka kraljica mora biti u posebnom retku ploče, pa se može uzeti da je i-ta kraljica u i-tom retku i rješenje problema se može prikazati kao n-torka (x_1, x_2, …, x_n), gdje je x_i indeks stupca u kojem se nalazi *i*-ta kraljica
- Slijedi da je skup rješenja S_i={1,2,…,n} za svaki *i,* broj n-torki u prostoru rješenja je n^n
- Ograničenja koja rješenje (x_1, x_2, …, x_n) mora zadovoljavati izvode se iz zahtjeva da se niti jedan par kraljica ne smije nalaziti u istom stupcu i istoj dijagonali
- pomoćna logička funkciju place() provjerava da li se k-ta kraljica može staviti u k-ti redak i x[k]-ti stupac tako da je već postavljenih k-1 kraljica ne napada (dakle elementi polja 1,…,k-1 su već određeni)
- Funkcija queens() ispisuje sva rješenja problema

```
#include <stdio.h>
#include <stdlib.h>
#define MAXLEN 8
int x[MAXLEN];

void queens(int n) {
 int k,ind;
 FILE *fout;
 if (!(fout=fopen("kraljice.txt","w"))) exit(1);
 x[1]=0; k=1; /* k je trenutni redak, x[k] je trenutni stupac */
 while (k > 0) { /* ponavlja za sve retke (kraljice) */
   x[k]++;
   while ( (x[k]<=n) && (place(k)==0) )  x[k]++; /* trazi stupac */
    if (x[k]<=n)  /* naden stupac */
      if (k == n) {     /* nadeno potpuno rjesenje */
          for (ind=1;ind<=MAXLEN; ind++){
                    printf("x[%d]=%d ",ind,x[ind]);
                    fprintf(fout,"x[%d]=%d ",ind,x[ind]);  }
                    printf("\n");
                    fprintf(fout,"\n");  }
      else {  /* trazi sljedeci redak (kraljicu) */
         k++; x[k]=0;       }
   else k--; } /* vraca u prethodni redak */
 fclose (fout); }
```

```
int place (int k) {
int i;
for (i=1; i<k; i++)
    if ( (x[i]==x[k]) || (abs(x[i]-x[k])==abs(i-k)) ) return 0;
return 1;
}

void main(void){
 queens(MAXLEN);
 exit;
}
```



Problem n kraljica – drugi primjer rješenja

- Algoritam rješenja:
- promatramo stupce na šahovskoj ploči od prvog prema zadnjem, u svaki postavljamo jednu kraljicu
- promatramo ploču u situaciji kada je već postavljeno *i* kraljica (u *i* različitih stupaca) koje se međusobno ne napadaju
- želimo postaviti *i* + 1 kraljicu tako da ona ne napada niti jednu od već postavljenih kraljica i da se ostale kraljice mogu postaviti uz uvjet nenapadanja
- Funkcija nenapadaju(x1,y1,x2,y2): funkcija koja govori da li se dvije kraljice, postavljene na polja (x1, y1) i (x2, y2) međusobno napadaju
- Rekurzivno rješenje

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

int KN(int *k, int i, int n) {
   int a, b;
   int dobar;
   if (i == n) return 1;
   for (a = 0; a < n; a++) {
      dobar = 1;
      for (b = 0; b < i; b++) {
         if (!nenapadaju(b + 1, k[b] + 1,
                         i + 1, a + 1)) {
            dobar = 0;
            break;         }
      }
      if (dobar) {
         k[i] = a;
         if (KN(k,i+1,n) == 1) return 1;
    }
   }
   return 0;
}
```

```
int nenapadaju(int x1, int y1, int x2, int y2) {
   int retval = 1;
   if ((x1 == x2) || (y1 == y2)) retval = 0;
   if (abs(x1-x2)==abs(y1-y2)) retval = 0;
   return retval; }

int main() {
   int k[8] = {0}, i;
   KN(k, 0, 8);
   for (i = 0; i < 8; i++)
      printf("(%d, %d)\n", i + 1, k[i] + 1);
     return 0;}
```

Usporedba algoritama sortiranja

- Primjer programa u kojem se uspoređuju brzine izvršavanja sljedećih algoritama sortiranja:

  1) sortiranje izborom (Selection Sort) – pronađe se najmanji element u nizu i stavi se na prvo mjesto (zamijene se mjesta najmanjeg i prvog), pa se postupak ponavlja za preostale elemente na kraćim listama

  2) mjehuričasto sortiranje (Bubble Sort) – svaki element se usporedi sa sljedbenikom, veći uvijek ide otraga, na kraju prvog koraka najveći element za zadnjem mjestu, postupak se ponavlja za kraću listu

  3) poboljšano mjehuričasto sortiranje: ako nema zamjene elemenata u nekom i-tom koraku, znači da je ostatak liste sortiran

  4) sortiranje umetanjem (Insertion Sort) – niz se dijeli na već sortiran i još nesortiran podniz, u prvom koraku je u prvom podnizu samo prvi element, u svakom koraku se uzima prvi element iz drugog podniza i uspoređuje s elementima u prvom podnizu dok se ne nađe pozicija na kojoj mora biti

5) Shellovo sortiranje - sortiranje kraćih podnizova u koracima upotrebom sortiranja umetanjem

6) sortiranje pomoću hrpe (Heap Sort) – zasniva se na specijalnom svojstvu hrpe da je roditelj uvijek veći (manji) od potomaka

7) sortiranje spajanjem (Merge Sort) – algoritam podijeli-pa-vladaj, rekurzivno dijeli niz na podnizove i sortirane podnizove spaja u konačni sortirani niz

8) brzo sortiranje (Quicksort) – poboljšano sortiranje spajanjem u kojem je izbjegnut sam proces spajanja podnizova; rekurzivni postupak u kojem se izabire pivot (stožerni) element koji se uspoređuje sa preostalim elementima, manji od njega se stavljaju prije pivota, a veći iza pivota

- Generiranje niza za sortiranje: elementi niza su indeksi polja, razbacaju se po polju upotrebom generatora slučajnih brojeva
- Svaki algoritam sortiranja se testira na razbacanom i sortiranom nizu i mjeri se vrijeme izvršavanja algoritma

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <sys\timeb.h>
typedef int tip;

// vrijeme u ms
int Trajanje (struct timeb *vrijeme1) {
  struct timeb vrijeme2;
  ftime (&vrijeme2);
  return 1000 * (vrijeme2.time - vrijeme1->time) +
              vrijeme2.millitm - vrijeme1->millitm;
}

// ispis poruke i prekid programa
void Fatalno (char *niz) {
  printf ("\n %s \n", niz);
     exit (1);
}
```

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

// sortiranje izborom
void SelectionSort (tip A [], int N) {
     int i, j, min;
     for (i = 0; i < N; i++) {
             min = i;
    for (j = i+1; j < N; j++) {
      if (A[j] < A[min]) min = j;
             }
             Zamijeni(&A[i], &A[min]);
     }
}
```

```
// mjehuricasto sortiranje
void BubbleSort (tip A [], int N) {
     int i, j;
     for (i = 0; i < N-1; i++) {
    for (j = 0; j < N-1-i; j++) {
      if (A[j+1] < A[j]) Zamijeni (&A[j], &A[j+1]);
              }
     }
}

//poboljšano mjehuricasto sortiranje
void BubbleSortPoboljsani (tip A [], int N) {
     int i, j, BilaZamjena;
     for (i = 0, BilaZamjena = 1; BilaZamjena; i++) {
              BilaZamjena = 0;
    for (j = 0; j < N-1-i; j++) {
      if (A[j+1] < A[j]) {
                                    Zamijeni (&A[j], &A[j+1]);
                                    BilaZamjena = 1;
                          }
              }
     }
}
```

```
// sortiranje umetanjem
void InsertionSort (tip A [], int N) {
  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;  }
}

// Shellovo sortiranje
void ShellSort (tip A [], int N) {
  int i, j, korak;
  tip pom;
  for (korak = N / 2; korak > 0; korak /= 2) {   // Insertion sort s vecim 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;    }  }
}
```

```
// podešavanje hrpe
void Podesi (tip A[], int i, int n) {
  int j;
  tip stavka;
  j = 2*i;
  stavka = A[i];
  while (j <= n ) {
    if ((j < n) && (A[j] < A[j+1])) j++;
    if (stavka >= A[j]) break;
    A[j/2] = A[j];
    j *=2;  }
  A[j/2] = stavka; }
// inicijalno stvaranje hrpe
void StvoriGomilu (tip A[], int n) {
  int i;
  for (i = n/2; i >= 1; i--)    Podesi (A, i, n); }
// sortiranje pomoću hrpe
void HeapSort (tip A[], int n) {  // A[1:n] sadrzi podatke koje treba sortirati
  int i;
  StvoriGomilu (A, n);
  for (i = n; i >= 2; i--) {// Zamijeni korijen i zadnji list, skrati polje za 1 i podesi hrpu
            Zamijeni (&A[1], &A[i]);
    Podesi (A, 1, i-1);
 }}
```

```
// 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 pelja
    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);
 } }
```

```
//sortiranje spajanjem
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
            Fatalno ("Nema mjesta za PomPolje!");
}

// QuickSort - 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]
  Zamijeni (&A [sredina], &A [desno - 1]);  // Vrati stozer
  return A [desno - 1];
}
```

```
// QuickSort - rekurzivno sortiranje podpolja
#define Cutoff (3)
void Qsort (tip A [], int lijevo, int desno) {
  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;    }    // Obnovi stozer
    Zamijeni (&A [i], &A [desno - 1]);
    Qsort (A, lijevo, i - 1);
    Qsort (A, i + 1, desno);
  } else {    // Sortiraj podpolje
    InsertionSort (A + lijevo, desno - lijevo + 1);  }}
// brzo sortiranje
void QuickSort (tip A [], int N) {
  Qsort (A, 0, N - 1);
}
```

```
// generira podatke za sort
void Generiraj (tip A [], int N) {
  int i;
     srand ((unsigned) time (NULL));
     // vrijednosti elemenata kao vrijednosti njihovih indeksa
  for( i = 0; i < N; i++ ) A [i] = i;
     // promijesaj vrijednosti
  for( i = 1; i < N; i++ )
    Zamijeni (&A [i], &A [rand () % (i + 1)]);
}

// provjeri da li svi elementi imaju vrijednost jednaku indeksu
void ProvjeriSort (tip A [], int N) {
  int i, flag = 0;
  for (i = 0; i < N; i++) {
    if (A[i] != i) {
      printf( "Sort ne radi: %d %d\n", i, A [i]);
                          flag = 1;
              }
     }
  if (!flag) printf( "Provjera zavrsena: sort OK\n" );
}
```

```
// kopira polje desno[] u polje lijevo[]
void Kopiraj (tip lijevo [], tip desno [], int N) {
  int i;
  for (i = 0; i < N; i++) lijevo [i] = desno [i]; }

// pokretanje potprograma za sort
void TestSorta (tip A[], tip B[], int N, char *ImeSorta, void (*Sort) (tip A[], int N)) {
      // A - polje koje se sortira
      // B - polje s podacima za sort
      // N - broj clanova polja
      // ImeSorta - naziv algoritma
      // Sort - pokazivac na funkciju koja obavlja sort
  struct timeb Vrijeme1;
              Kopiraj (A, B, N); // kopiraj podatke iz B u A
      // sortiraj i mjeri vrijeme
  printf ("%s...\n", ImeSorta);
  ftime (&Vrijeme1);
      if (strcmp(ImeSorta, "Heap Sort") == 0) {
    Sort (A-1, N); // da HeapSort "vidi" A[0] kao A[1]
      } else {
              Sort (A, N);  }
  printf ("Trajanje: %d ms\n", Trajanje(&Vrijeme1));
  ProvjeriSort (A, N);
```

```
// sortiraj prethodno sortirano polje A
  printf ("%s sortiranog polja...\n", ImeSorta);
     ftime (&Vrijeme1);
     if (strcmp(ImeSorta, "Heap Sort") == 0) {
    Sort (A-1, N);
     } else {
            Sort (A, N);
     }
  printf ("Trajanje: %d ms\n", Trajanje(&Vrijeme1));
  ProvjeriSort (A, N);
     printf ("Pritisni bilo koju tipku...\n\n");
     getchar();
}
```

```
void main (void) {
  int *Polje1, *Polje2, Duljina;
     // inicijalizacija
     printf ("Unesi broj clanova polja >");
     scanf ("%d", &Duljina);
     Polje1 = (int *) malloc (Duljina * sizeof (int));
     Polje2 = (int *) malloc (Duljina * sizeof (int));
     if (!Polje1 || !Polje2) Fatalno ("Nema dovoljno memorije!");
     // generiranje podataka
  Generiraj (Polje2, Duljina);
  // sortiranje
  TestSorta (Polje1, Polje2, Duljina, "Selection Sort", SelectionSort);
  TestSorta (Polje1, Polje2, Duljina, "Bubble Sort", BubbleSort);
  TestSorta (Polje1, Polje2, Duljina, "Bubble Sort poboljsani", BubbleSortPoboljsani);
  TestSorta (Polje1, Polje2, Duljina, "Insertion Sort", InsertionSort);
  TestSorta (Polje1, Polje2, Duljina, "Shell Sort", ShellSort);
  TestSorta (Polje1, Polje2, Duljina, "Heap Sort", HeapSort);
  TestSorta (Polje1, Polje2, Duljina, "Merge Sort", MergeSort);
  TestSorta (Polje1, Polje2, Duljina, "Quick Sort", QuickSort);
  system("PAUSE");
  exit(0);
}
```

Slučajno generiranje zadataka za ispit

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define studenti 29
#define zadaci 50
int main(){
 int flag[zadaci+1]={0}, out[studenti], zgen, x, ind;
 FILE *fout;
 srand(time(NULL));
 zgen=0;
 if (!(fout=fopen("zadaci.txt","w"))) exit(1);
 srand(time(NULL)/(rand()%1000));
 while (zgen<studenti){
      x= 1 + zadaci*((float)rand() / (RAND_MAX + 1));
       if (flag[x]==0) {
                flag[x]++;
                out[zgen]=x;
                zgen++;}
              }
  for (ind=0; ind <zgen; ind++){
       printf("%d. broj je; %d\n",ind+1,out[ind]);
       fprintf(fout,"%d. broj je: %d\n",ind+1,out[ind]);}
  fclose(fout);
  system("PAUSE");}
```