```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <time.h>

void merge(int *m1,int n1,int *m2,int n2, int *r)
{int i,j,k;
 for(i=0,j=0,k=0;i<n1&&j<n2;)
  if(m1[i]<m2[j])
   r[k++]=m1[i++];
  else
   r[k++]=m2[j++];
 memcpy(r+k,m1+i, (n1-i)*sizeof(int));
 memcpy(r+k,m2+j, (n2-j)*sizeof(int));
}

void Sort1(int *m,int n,int *t)
{int n1,n2;
 if(n<=1)return ;
 n1=n/2;n2=n-n1;
 Sort1(m,n1,t);
 Sort1(m+n1,n2,t);
 merge(m,n1, m+n1,n2, t);
 memcpy(m,t,n*sizeof(int));
}

#define SWAP(a,b,t) {(t)=(a);(a)=(b);(b)=(t);}
#define MAX(a,b) ((a)>(b)?(a):(b))

void Sort2(int *m,int n,int *t)
{int l,i,n2,tmp;
 l=1;
 {
  for(i=0;i+1<n;i+=2)
  {
   if(n-(i+1)>0 && m[i]>m[i+1])
    SWAP(m[i],m[i+1],tmp);
  }
 }
 for(l=2;l<n;l*=2)
 {
  for(i=0;i+l<n;i+=2*l)
  {
   n2=l;if(n-(i+l)<l)n2=n-(i+l);
   merge(m+i,l,m+i+l,n2, t);
   memcpy(m+i,t,(l+n2)*sizeof(int));
  }
 }
}


void QSort1(int *m,int p,int q)//m[p],...,m[q]
{int i,j,t;
 if(p>=q)return;
 i=p;j=q; //M=m[j]; [p,i-1]<=M<=[j,q]
 while(1)
 {
  while(m[i]<m[j])i++;//M=m[j]; [p,i-1]<=M<=[j,q]
  SWAP(m[i],m[j],t);  //M=m[i]; [p,i]<=M<=[j,q]
  j--;                //M=m[i]; [p,i]<=M<=[j+1,q]
```

```
  if(i>=j)
  {
   //i==j:  [p,j]<=M<=[j+1,q]
   //i==j+1:[p,j]<=M<=[j+1,q]
   QSort1(m,p,j); QSort1(m,j+1,q); return;
  }
  while(m[i]<m[j])j--;//M=m[i]; [p,i]<=M<=[j+1,q]
  SWAP(m[i],m[j],t);  //M=m[j]; [p,i]<=M<=[j,q]
  i++;                //M=m[j]; [p,i-1]<=M<=[j,q]
  if(i>=j)
  {
   //i==j:    [p,j]<=[j,q]
   //i==j+1:  [p,j]<=[j,q]
   QSort1(m,p,j); QSort1(m,j+1,q); return;
  }
 }
}

void QSort2(int *m,int p,int q)
{int i,j,M,t;
l0:
 if(p>=q)return;
 if(q-p>8)
 {
  i=p+rand()%(q-p+1);
  SWAP(m[p],m[i],t);
 }
 i=p;j=q;
 M=m[p];//[p,i-1]<=M<=[j+1,q]
 while(1)
 {
  while(m[i]<M)i++;//[p,i-1]<=M<=[j+1,q]
  while(M<m[j])j--;//[p,i-1]<=M<=[j+1,q]
  if(i>=j)
  {
   //i==j,M==m[j]: [p,j]<=M<=[j+1,q]
   //i==j+1:       [p,j]<=M<=[j+1,q]
   if(j-p<q-j)
   {QSort2(m,p,j);    p=j+1; q=q; goto l0;}
   else
   {QSort2(m,j+1,q);    p=p;q=j; goto l0;}
  }
  SWAP(m[i],m[j],t);//[p,i]<=M<=[j,q]
  i++;j--;          //[p,i-1]<=M<=[j+1,q]
 }
}


void QSort2S(int *m,int p,int q)
{int i,j,M,t;
l0:
 if(p>=q)return;
 if(q-p<=300)
 {static int t[301];
  Sort1(m+p,q-p+1,t);
  return;
 }
 else //if(q-p>8)
 {
  i=p+rand()%(q-p+1);
  SWAP(m[p],m[i],t);
```

```
 }
 i=p;j=q;
 M=m[p];//[p,i-1]<=M<=[j+1,q]
 while(1)
 {
  while(m[i]<M)i++;//[p,i-1]<=M<=[j+1,q]
  while(M<m[j])j--;//[p,i-1]<=M<=[j+1,q]
  if(i>=j)
  {
   //i==j,M==m[j]: [p,j]<=M<=[j+1,q]
   //i==j+1:       [p,j]<=M<=[j+1,q]
   if(j-p<q-j)
   {QSort2(m,p,j);    p=j+1; q=q; goto l0;}
   else
   {QSort2(m,j+1,q);    p=p;q=j; goto l0;}
  }
  SWAP(m[i],m[j],t);//[p,i]<=M<=[j,q]
  i++;j--;          //[p,i-1]<=M<=[j+1,q]
 }
}

#define iL (2*i+1)
#define iR (2*i+2)
#undef C
#define C a[i]
#define L a[iL]
#define R a[iR]

void Heapify(int *a,int i,int n)
{int t;
 while(iR<n)
 {
  if(C>=L&&C>=R)return;
  if(L>R){SWAP(L,C,t);i=iL;}
  else{SWAP(R,C,t);i=iR;}
 }
 if(iL<n&&C<L)SWAP(L,C,t);
}

void HSort(int *a,int n)
{int i,t;
 for(i=n-2;i>=0;i--)Heapify(a,i,n);
 for(n--;n>=0;n--){SWAP(a[0],a[n],t);Heapify(a,0,n);}
}

void CSort(int *a,int n, int *c,  int *b,int M)
{int i;
 memset(b,0,(M+1)*sizeof(int));
 for(i=0;i<n;i++)b[a[i]]++;
 for(i=1;i<=M;i++)b[i]+=b[i-1];
 for(i=n-1;i>=0;i--){c[b[a[i]]-1]=a[i]; b[a[i]]--;}
}

void DSort(int *m_,int n, int *c,  int *b,int M)
{union IS{int i;unsigned char c[4];};
 union IS *a=(union IS*)m_; int I,i;
 for(I=0;I<4;I++)
 {
  memset(b,0,(M+1)*sizeof(int));
  for(i=0;i<n;i++)b[a[i].c[I]]++;
  for(i=1;i<=M;i++)b[i]+=b[i-1];
```

```
  for(i=n-1;i>=0;i--){c[b[a[i].c[I]]-1]=a[i].i; b[a[i].c[I]]--;}
  memcpy(a,c,n*sizeof(int));
 }
}

#define SET_INDEX(type,v,n,nmax) \
   if((n)+1>=(nmax)){v=(type*)realloc(v,(nmax=((n)*2+1))*sizeof(type));}

#define ARR(type,x) int x##__nmax=10; type *x=NULL; x=(type*)malloc(x##__nmax*sizeof(type));
#define SET_ARR(type,x,n) \
if((n)+1>=x##__nmax){x=(type*)realloc(x,(x##__nmax=((n)*2+1))*sizeof(type));}


struct S2
{
 char x;int y;
};

struct S3
{
 char x;char y;char z;
};

struct SStack_;

typedef
struct SStack_
{
 int *s;
 int n,nmax;
}SStack;

SStack *Create()
{SStack *x; x=(SStack*)malloc(sizeof(*x));
 memset(x,0,sizeof(*x));
return x;
}

int Empty(SStack *s)
{return s==NULL||s->n==0;}

int Push(SStack *s, int x)
{
 if(s==NULL)return -1;
 if(s->n>=s->nmax)
  s->s=(int*)realloc(s->s,(s->nmax=s->n*2+1)*sizeof(int));
 s->s[s->n++]=x;
 return 0;
}

int Get(SStack *s,int *top)
{if(s==NULL||s->n==0)return -1;
 *top=s->s[s->n-1];
return 0;
}

int Pop(SStack *s)
{if(s==NULL||s->n==0)return -1;
 s->n--;
return 0;
}
```

```
void Delete(SStack **s)
{
 if(s!=NULL&&*s!=NULL)
 {
  if((*s)->s!=NULL)free((*s)->s);
  free(*s);
  *s=NULL;
 }
}


int **Create2()
{int **s,nmax=10;
 s=(int**)malloc(3*sizeof(int*)+nmax*sizeof(int));
 s[0]=s[1]=(int*)(s+3);
 s[2]=s[1]+nmax;
 return s;
}

void Delete2(int ***s)
{
 if(s==NULL||*s==NULL)return;
 free(*s);
 *s=NULL;
}

int Empty2(int **s)
{return s[0]==s[1];}

int Push2(int **s, int x)
{
 if(s==NULL||s[1]==s[2])return -1;
 *(s[1]++)=x;
 return 0;
}

int Get2(int **s, int *x)
{
 if(s==NULL ||Empty2(s))return -1;
 *x=s[1][-1];
 return 0;
}

int Pop2(int **s)
{
 if(s==NULL||Empty2(s))return -1;
 s[1]--;
 return 0;
}

typedef
struct SQueue_
{
 int *q;
 int nmax;
 int Head,Tail;
}SQueue;

SQueue *CreateQ(void)
{int nmax=10; SQueue *q;
```

```
 q=(SQueue*)malloc(sizeof(SQueue)+nmax*sizeof(int));
 q->q=(int*)(q+1);
 q->nmax=nmax;
 q->Head=q->Tail=nmax-1;
 return q;
}

int DeleteQ(SQueue**q)
{
 if(q==NULL||*q==NULL)return -1;
 free(*q);
 *q=NULL;
 return 0;
}

int EmptyQ(SQueue *q)
{return q==NULL||q->Head==q->Tail;}

int PushQ(SQueue *q,int x)
{
 if(q==NULL || q->Tail-q->Head==1 || q->Head-q->Tail==q->nmax-1)return -1;
 q->q[q->Tail--]=x;
 if(q->Tail<0)q->Tail=q->nmax-1;
 return 0;
}

int TopQ(SQueue *q, int *x)
{
 if(EmptyQ(q))return -1;
 *x=q->q[q->Head];
 return 0;
}

int CleanTopQ(SQueue *q)
{
 if(EmptyQ(q))return -1;
 q->Head--;
 if(q->Head<0)q->Head=q->nmax-1;
 return 0;
}

typedef struct PriorityQueue_
{
 int *q;
 int n,nmax;
}PriorityQueue;

PriorityQueue *CreatePQ()
{PriorityQueue *q; int nmax=10;
 q=(PriorityQueue*)malloc(sizeof(PriorityQueue)+nmax*sizeof(int));
 q->n=0;q->nmax=nmax;
 q->q=(int*)(q+1);
 return q;
}

void DeletePQ(PriorityQueue **q)
{
 if(q==NULL||*q ==NULL)return;
 free(*q);
 *q=NULL;
}
```

```
int EmptyPQ(PriorityQueue *q)
{return q==NULL||q->n==0;}

int PushPQ(PriorityQueue *q,int x)
{int i,t;
 if(q==NULL||q->n>=q->nmax)return -1;
 q->q[q->n]=x;
 for(i=q->n;i>0;i=(i-1)/2)
  if(q->q[(i-1)/2]<q->q[i])
   SWAP(q->q[(i-1)/2],q->q[i],t)
  else
   break;
 q->n++;
 return 0;
}

int GetPQ(PriorityQueue *q,int *x)
{
 if(EmptyPQ(q))return -1;
 *x=q->q[0];
 return 0;
}

int CleanTopPQ(PriorityQueue *q)
{int t;
 if(EmptyPQ(q))return -1;
 SWAP(q->q[0],q->q[q->n-1],t);
 q->n--;
 Heapify(q->q,0,q->n);
 return 0;
}

typedef
struct SItem_
{
 int v;
 struct SItem_ *prev,*next;
}SItem;

typedef struct SList2_
{
 SItem *first,*last,*cur;
}SList2;

SList2 *CreateL()
{SList2 *l;
 l=(SList2*)malloc(sizeof(*l));
 l->first=l->last=l->cur=NULL;
 return l;
}

int DelCur(SList2 *l)
{
 if(l==NULL||l->cur==NULL)return -1;
 if(l->cur->prev==NULL)
 {
  if(l->cur->next==NULL)
  {free(l->cur);l->cur=l->first=l->last=NULL;}
  else
  {l->first=l->cur->next;free(l->cur); l->cur=l->first; l->cur->prev=NULL;}
```

```
 }
 else
 {
  if(l->cur->next==NULL)
  {l->last=l->cur->prev;free(l->cur);l->cur=l->last;l->cur->next=NULL;}
  else
  {SItem *c=l->cur;
   l->cur->prev->next=l->cur->next; l->cur->next->prev=l->cur->prev;l->cur=l->cur-
>prev;free(c);
  }
 }
 return 0;
}

int AddAfter(SList2 *l, int x)
{
 if(l==NULL)return -1;
 if(l->cur==NULL)
 {l->cur=l->first=l->last=(SItem*)malloc(sizeof(SItem)); l->cur->v=x; l->cur->prev=l->cur-
>next=NULL;}
 else
 {SItem *i;
  i=(SItem*)malloc(sizeof(SItem)); i->v=x; i->prev=i->next=NULL;
  if(l->first==NULL){l->first=l->last=i;return 0;}
  i->prev=l->cur; i->next=l->cur->next;
  l->cur->next=i;
  if(i->next)
   i->next->prev=i;
  else
   l->last=i;
 }
 return 0;
}

int GetL(SList2 *l, int *x)
{
 if(l==NULL||l->cur==NULL)return -1;
 *x=l->cur->v;
 return 0;
}

int GoToFirst(SList2 *l)
{
 if(l==NULL||l->cur==NULL)return -1;
 l->cur=l->first;
 return 0;
}

int GoToNext(SList2 *l)
{
 if(l==NULL||l->cur==NULL||l->cur->next==NULL)return -1;
 l->cur=l->cur->next;
 return 0;
}

void DeleteL(SList2 **l)
{
 for(GoToFirst(*l);DelCur(*l)==0;GoToFirst(*l));
 free(*l);
 *l=NULL;
}
```

```
typedef struct SItemM_
{int v; int prev,next;}SItemM;

typedef struct SListM_
{
 int cur,first,last;
 SItemM *m; int n;
}SListM;

SListM *CreateM()
{SListM *l; int i;
 l=(SListM *)malloc(sizeof(SListM));
 l->m=(SItemM*)malloc((l->n=5)*sizeof(SItemM));
 l->cur=l->first=l->last=0;
 for(i=0;i<l->n-1;i++)l->m[i].next=i+1; l->m[i].next=0;
 return l;
}

int MallocM(SListM *l)
{int i;
 if(l->m->next==0)
 {
  l->m=(SItemM*)realloc(l->m, 2*l->n*sizeof(SItemM));
  for(l->m->next=l->n, i=l->n;i<2*l->n-1;i++)l->m[i].next=i+1;
  l->m[i].next=0;
  l->n*=2;
 }
 i=l->m->next; l->m->next=l->m[i].next;
 return i;
}

void FreeM(SListM *l, int i)
{l->m[i].next=l->m->next; l->m->next=i;}

#define CUR (l->m+l->cur)
#define I (l->m+i)

int DelCurM(SListM *l)
{
 if(l==NULL||l->cur==0)return -1;
 if(CUR->prev==0)
 {
  if(CUR->next==0)
  {FreeM(l,l->cur);l->cur=l->first=l->last=0;}
  else
  {l->first=CUR->next;FreeM(l,l->cur); l->cur=l->first; CUR->prev=0;}
 }
 else
 {
  if(CUR->next==NULL)
  {l->last=CUR->prev;FreeM(l,l->cur);l->cur=l->last;CUR->next=0;}
  else
  {int c=l->cur;
   l->m[CUR->prev].next=CUR->next; l->m[CUR->next].prev=CUR->prev;l->cur=CUR->prev;
   FreeM(l,c);
  }
 }
 return 0;
}
```

```
int AddAfterM(SListM *l, int x)
{
 if(l==NULL)return -1;
 if(l->cur==0)
 {l->cur=l->first=l->last=MallocM(l); CUR->v=x; CUR->prev=CUR->next=0;}
 else
 {int i;
  i=MallocM(l); I->v=x; I->prev=I->next=NULL;
  if(l->first==0){l->first=l->last=i;return 0;}
  I->prev=l->cur; I->next=CUR->next;
  CUR->next=i;
  if(I->next)
   l->m[I->next].prev=i;
  else
   l->last=i;
 }
 return 0;
}

int GetM(SListM *l, int *x)
{
 if(l==NULL||l->cur==0)return -1;
 *x=CUR->v;
 return 0;
}

int GoToFirstM(SListM *l)
{
 if(l==NULL||l->cur==0)return -1;
 l->cur=l->first;
 return 0;
}

int GoToNextM(SListM *l)
{
 if(l==NULL||l->cur==0||CUR->next==0)return -1;
 l->cur=CUR->next;
 return 0;
}

void DeleteM(SListM **l)
{
 for(GoToFirstM(*l);DelCurM(*l)==0;GoToFirstM(*l));
 free(*l);
 *l=NULL;
}

typedef struct SItem1_
{
 int v;
 struct SItem1_ *next;
}SItem1;

typedef struct SList1_
{
 SItem1 m0;
 SItem1 *cur;
}SList1;

SList1 *Create1()
{SList1 *l;
```

```
 l=(SList1 *)malloc(sizeof(SList1));
 l->cur=&(l->m0);
 l->m0.next=0;
 return l;
}
//----------------------------
typedef struct SItemC_
{
 int v;
 struct SItemC_ *next,*prev;
}SItemC;

typedef struct SListC_
{
 SItemC *cur;
}SListC;
//----------------------------
//1)++: 100 200
//2)-+
//3)+-
//--
//----------------------------
int main(void)
{int i,j;
int a1[100][200],(*a2)[200],*a3[100],**a4;
 for(i=0;i<100;i++)for(j=0;j<200;j++)a1[i][j]=i+j;
 a2=(int(*)[200])malloc(100*200*sizeof(int));
 for(i=0;i<100;i++)for(j=0;j<200;j++)a2[i][j]=i+j;
 a3[0]=(int*)malloc(100*200*sizeof(int));
 for(i=1;i<100;i++)a3[i]=a3[i-1]+200;
 for(i=0;i<100;i++)for(j=0;j<200;j++)a3[i][j]=i+j;
 a4=(int**)malloc(100*sizeof(int*)+100*200*sizeof(int));
 a4[0]=(int*)(a4+100);
 for(i=1;i<100;i++)a4[i]=a4[i-1]+200;
 for(i=0;i<100;i++)for(j=0;j<200;j++)a4[i][j]=i+j;
 //--
 free(a2); a2=NULL;
 free(a3[0]);
return 0;
}

int main_x6(void)
{SListM *l; int i,x; time_t t0,t1; long  S=0;
time(&t0);
 l=CreateM();
 for(i=0;i<100*200000;i++)AddAfterM(l,i);
 if(GoToFirstM(l)==0)
  do
 {GetM(l,&x);S+=x;if(0)printf("%d ",x);}
 while(GoToNextM(l)==0);
 DeleteM(&l);
 printf("\n---\n");
 if(GoToFirstM(l)==0)
  do
 {GetM(l,&x);if(0)printf("%d ",x);}
 while(GoToNextM(l)==0);
 time(&t1);
 printf("dt=%d S= %ld\n",t1-t0,S);
 getchar();
 return 0;
}
```

```
#undef I

int main_x4(void)
{SList2 *l; int i,x; time_t t0,t1; long  S=0;
time(&t0);
 l=CreateL();
 for(i=0;i<200000;i++)AddAfter(l,i);
 if(GoToFirst(l)==0)
  do
 {GetL(l,&x);S+=x;if(0)printf("%d ",x);}
 while(GoToNext(l)==0);
 DeleteL(&l);
 printf("\n---\n");
 if(GoToFirst(l)==0)
  do
 {GetL(l,&x);if(0)printf("%d ",x);}
 while(GoToNext(l)==0);
 time(&t1);
 printf("dt=%d S=-1474936480= %ld\n",t1-t0,S);
 getchar();
 return 0;
}


int main_x3(void)
{PriorityQueue *q; int x;
 q=CreatePQ();
 PushPQ(q, 4);PushPQ(q, 1);PushPQ(q, 6);PushPQ(q, 2);
 while(!EmptyPQ(q)){GetPQ(q,&x);printf("%d ",x);CleanTopPQ(q);}
 DeletePQ(&q);
 getchar();
 return 0;
}

int main_x2(void)
{int x=1,y=32;
 printf("%d\n",x<<3);
 printf("%d\n",x<<31);
 printf("%d\n",1<<y);
 getchar();
 {SQueue *q; int i,x; q=CreateQ();
 for(i=0;i<7;i++)PushQ(q,i);
 while(!EmptyQ(q)){TopQ(q,&x);printf("%d ",x);CleanTopQ(q);}
 for(i=0;i<7;i++)PushQ(q,i+7);
 while(!EmptyQ(q)){TopQ(q,&x);printf("%d ",x);CleanTopQ(q);}

 DeleteQ(&q);
 getchar();
 }
 return 0;
}

int main_x1(void)
{int **s,i,x;
 s=Create2();
 for(i=0;i<18;i++)if(Push2(s,i))printf("Can't push %d\n",i);
 for(i=0;i<18;i++)
 {if(Get2(s,&x))printf("Can't get %d\n",i); else printf("top=%d\n",x); Pop2(s);}

 Delete2(&s);
 getchar();
```

```
 return 0;
}

int main_____(void)
{SStack *s=NULL; int i,x;
s=Create();
 for(i=0;i<10;i++)Push(s,2*i);
 //while(!Empty(s)){Get(s,&x); printf("%d ",x); Pop(s);}
 while(Get(s,&x)==0){printf("%d ",x);Pop(s);}
 Delete(&s);
 getchar();
return 0;
}

/*int main____(void)
{struct S3 x[10];
 struct SStack y;
 memset(&(y.n),0,((char*)&y.n)-((char*)&y.s)+sizeof(y.n));
 printf("%d %d %d %d\n",sizeof(struct SStack),sizeof(struct S2),sizeof(struct
S3),sizeof(x));
 getchar();
 return 0;
}*/

int main___(void)
{ARR(int,x); ARR(float,y); int i;
 for(i=0;i<100;i++)
 {
  SET_ARR(int,x,i);
  x[i]=i;
 }
 for(i=0;i<100;i++)printf("%d ",x[i]);
 getchar();
return 0;
}

int main__(void)
{int nmax=10,*m=NULL,i;
 m=(int*)malloc(nmax*sizeof(int));
 for(i=0;i<100;i++)
 {
  SET_INDEX(int,m,i,nmax);
  m[i]=i;
 }
 for(i=0;i<100;i++)printf("%d ",m[i]);
 getchar();
 return 0;
}

int main_(void)
{int *m=NULL,*m1=NULL,*m2=NULL,*m3=NULL,*t=NULL,n=400,i,N=1000000,I; time_t t0,t1;
printf("RAND_MAX=%d\n",RAND_MAX);
 m=(int*)malloc(n*sizeof(int));
 m1=(int*)malloc(n*sizeof(int));
 m2=(int*)malloc(n*sizeof(int));
 m3=(int*)malloc(n*sizeof(int));
 t=(int*)malloc(MAX(RAND_MAX+1,n)*sizeof(int));
 for(i=0;i<n;i++)m[i]=m1[i]=(rand()+rand()*32000)%n;//0,..,RAND_MAX
 //---
 if(0)
 {
```

```
 time(&t0);
 for(I=0;I<N;I++)
 {
  memcpy(m1,m,n*sizeof(int));
  Sort1(m1,n,t);
 }
 time(&t1);printf("Sort1: %d\n",(int)(t1-t0));
 for(i=1;i<n;i++)if(m1[i-1]>m1[i])printf("error:%d\n",i);
}
//---
if(0)
{
 time(&t0); memcpy(m2,m,n*sizeof(int));
 Sort2(m2,n,t);
 time(&t1);printf("Sort2: %d\n",(int)(t1-t0));
 for(i=0;i<n;i++)if(m1[i]!=m2[i])printf("error2: %d\n",i);
}
//---
if(0)
{
 time(&t0); memcpy(m2,m,n*sizeof(int));
 QSort1(m2,0,n-1);
 time(&t1);printf("QSort1: %d\n",(int)(t1-t0));
 for(i=0;i<n;i++)if(m1[i]!=m2[i])printf("error2: %d\n",i);
}
//---
if(1)
{
 time(&t0);
 for(I=0;I<N;I++)
 {
  memcpy(m2,m,n*sizeof(int));
  QSort2(m2,0,n-1);
 }
 time(&t1);printf("QSort2: %d\n",(int)(t1-t0));
 //for(i=0;i<n;i++)if(m1[i]!=m2[i])printf("error2: %d\n",i);
}
memcpy(m1,m2,n*sizeof(int));
//---
if(0)
{
 time(&t0); memcpy(m2,m,n*sizeof(int));
 HSort(m2,n);
 time(&t1);printf("HSort: %d\n",(int)(t1-t0));
 for(i=0;i<n;i++)if(m1[i]!=m2[i])printf("error2: %d\n",i);
}
//---
if(0)
{
 time(&t0); memcpy(m3,m,n*sizeof(int));
 CSort(m3,n, m2, t,RAND_MAX);
 time(&t1);printf("CSort: %d\n",(int)(t1-t0));
 for(i=0;i<n;i++)if(m1[i]!=m2[i])printf("error2: %d\n",i);
}
//---
if(1)
{
 time(&t0);
 for(I=0;I<N;I++)
 {
  memcpy(m3,m,n*sizeof(int));
```

```
   DSort(m3,n, m2, t,255);
  }
  time(&t1);printf("DSort: %d\n",(int)(t1-t0));
  for(i=0;i<n;i++)if(m1[i]!=m2[i])printf("error2: %d\n",i);
 }
 //---
 free(m);m=NULL;
 free(m1);m1=NULL;
 free(m2);m2=NULL;
 free(m3);m3=NULL;
 free(t);t=NULL;
 printf("end\n");
 getchar();
 return 0;
}
```