```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
void Merge(int *a1,int n1, int *a2,int n2, int *r)
{int i1,i2,i;
 for(i1=0,i2=0,i=0;i1<n1&&i2<n2;)
   if(a1[i1]<a2[i2])
     r[i++]=a1[i1++];
   else
     r[i++]=a2[i2++];
 while(i1<n1)r[i++]=a1[i1++];
 while(i2<n2)r[i++]=a2[i2++];
}

void MSort(int *m,int n,int *t)
{int n1,n2;
 if(n<=1)return;
 n1=n/2; n2=n-n1;
 MSort(m,n1,t);
 MSort(m+n1,n2,t);
 Merge(m,n1, m+n1,n2, t);
 memcpy(m,t,n*sizeof(int));
}
#define SWAP(a,b) {tt=(a); (a)=(b); (b)=tt;}

void BSort(int *m,int n)
{int i,j,tt;
 for(i=n-1;i>0;i--)
  for(j=0;j<i;j++)
   if(m[j]>m[j+1])SWAP(m[j],m[j+1]);
}

void MSort2(int *m,int n,int *t)
{int i,k,k2,tt;
k=1;
 {//m+i,k,  m+i+k,k2
  for(i=0,k2=k;i+1<n;i+=2)
   if(m[i]>m[i+1])SWAP(m[i],m[i+1]);
 }
 for(k=2;k<n;k*=2)
 {//m+i,k,  m+i+k,k2
  for(i=0,k2=k;i+k<n;i+=2*k)
  {
   if(n-i-k<k2)k2=n-i-k;
   Merge(m+i,k, m+i+k,k2, t);
   memcpy(m+i,t,(k+k2)*sizeof(int));
  }
 }
}

void MSortX(int *m,int n,int *t)
{int n1,n2,n3,n4;
 if(n<=1)return;
 n1=n/2; n3=n-n1;
 n2=n1/2;n1=n1-n2;
 n4=n3/2;n3=n3-n4;
#pragma omp parallel sections
 {
#pragma omp section
  {MSort(m,n1,t);}
#pragma omp section
  {MSort(m+n1,n2,t+n1);}
#pragma omp section
  {MSort(m+n1+n2,n3,t+n1+n2);}
#pragma omp section
  {MSort(m+n1+n2+n3,n4,t+n1+n2+n3);}
 }
#pragma omp parallel sections
```

```
 {
#pragma omp section
  {Merge(m,n1, m+n1,n2, t);memcpy(m,t,(n1+n2)*sizeof(int));}
#pragma omp section
  {Merge(m+n1+n2,n3, m+n1+n2+n3,n4,
  t+n1+n2);memcpy(m+n1+n2,t+n1+n2,(n3+n4)*sizeof(int));}
 }
 Merge(m,n1+n2, m+n1+n2,n3+n4, t);
 memcpy(m,t,n*sizeof(int));
}

void MSort2X(int *m,int n,int *t)
{int i,k,k2,tt;
k=1;
 {//m+i,k,  m+i+k,k2
#pragma omp parallel for private(tt)
  for(i=0;i<n-1;i+=2)
   if(m[i]>m[i+1])SWAP(m[i],m[i+1]);
 }
 for(k=2;k<n;k*=2)
 {//m+i,k,  m+i+k,k2
#pragma omp parallel for private(k2)
  for(i=0;i<n-k;i+=2*k)
  {
   k2=k;
   if(n-i-k<k2)k2=n-i-k;
   Merge(m+i,k, m+i+k,k2, t+i);
   memcpy(m+i,t+i,(k+k2)*sizeof(int));
  }
 }
}

#undef SWAP
#define SWAP(a,b,tt) {tt=(a); (a)=(b); (b)=tt;}

void QSort1(int *m,int p,int q)
{int i,j,tt;
 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],tt);//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],tt);//M=m[j], [p,i]<=M<=[j,q]
  i++;//M=m[j], [p,i-1]<=M<=[j,q]
  if(i>=j)
  {
   //i==j (M=m[i]): [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;
  }
 }
}

int QSort1_(int *m,int p,int q)
{int i,j,tt;
 if(p>=q)return p;
 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],tt);//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]
     return j;
    }
    while(m[i]<m[j])j--;//M=m[i], [p,i]<=M<=[j+1,q]
    SWAP(m[i],m[j],tt);//M=m[j], [p,i]<=M<=[j,q]
    i++;//M=m[j], [p,i-1]<=M<=[j,q]
    if(i>=j)
    {
     //i==j (M=m[i]): [p,j]<=M<=[j+1,q]
     //i==j+1:        [p,j]<=M<=[j+1,q]
     return j;
    }
   }
  }

  void QSort1X(int *m,int n)
  {int j[9]={-1,0,0,0, 0, 0,0,0,n-1},i;//{j[k]+1,j[k]}
   j[4]=QSort1_(m,0,n-1);
  #pragma omp parallel sections
   {
  #pragma omp section
    {j[2]=QSort1_(m,0,j[4]);}
  #pragma omp section
    {j[6]=QSort1_(m,j[4]+1,j[8]);}
   }
  #pragma omp parallel sections
   {
  #pragma omp section
    {j[1]=QSort1_(m,0,j[2]);}
  #pragma omp section
    {j[3]=QSort1_(m,j[2]+1,j[4]);}
  #pragma omp section
    {j[5]=QSort1_(m,j[4]+1,j[6]);}
  #pragma omp section
    {j[7]=QSort1_(m,j[6]+1,j[8]);}
   }
  #pragma omp parallel for
   for(i=0;i<8;i++)
    QSort1(m,j[i]+1,j[i+1]);
  }

  void QSort2(int *m,int p,int q)
  {int i,j,M,tt;
  l1:
   if(p>=q)return;
   if(q-p>20)
   {
    i=p+rand()%(q-p+1);
    SWAP(m[p],m[i],tt);
   }
   i=p;j=q;M=m[i];//[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[i]): [p,j]<=[j+1,q]
     //i==j+1:        [p,j]<=[j+1,q]
     if(q-j>j-p)
     {
      QSort2(m,p,j);
      p=j+1;
      goto l1;
```

```
    }
    else
    {
     QSort2(m,j+1,q);
     q=j;
     goto l1;
    }
    //return;
   }
   SWAP(m[i],m[j],tt);//[p,i]<=M<=[j,q]
   i++;j--;//[p,i-1]<=M<=[j+1,q]
  }
 }

 void QSort2X(int *m,int n)
 {int j[9]={-1,0,0,0, 0, 0,0,0,n-1},i;//{j[k]+1,j[k]}
  j[4]=QSort1_(m,0,n-1);
 #pragma omp parallel sections
  {
 #pragma omp section
   {j[2]=QSort1_(m,0,j[4]);}
 #pragma omp section
   {j[6]=QSort1_(m,j[4]+1,j[8]);}
  }
 #pragma omp parallel sections
  {
 #pragma omp section
   {j[1]=QSort1_(m,0,j[2]);}
 #pragma omp section
   {j[3]=QSort1_(m,j[2]+1,j[4]);}
 #pragma omp section
   {j[5]=QSort1_(m,j[4]+1,j[6]);}
 #pragma omp section
   {j[7]=QSort1_(m,j[6]+1,j[8]);}
  }
 #pragma omp parallel for
  for(i=0;i<8;i++)
   QSort2(m,j[i]+1,j[i+1]);
 }

 #define c m[i]
 #define l m[2*i+1]
 #define r m[2*i+2]
 #define HasR (2*i+2<n)
 #define HasL (2*i+1<n)

 void Heapify(int *m,int i,int n)
 {int tt;
  while(HasR)
  {
   if(c>=l&&c>=r)return;
   if(l>r){SWAP(c,l,tt);i=2*i+1;}
   else {SWAP(c,r,tt);i=2*i+2;}
  }
  if(HasL && l>c)SWAP(l,c,tt);
 }

 #undef c
 #undef l
 #undef r
 #undef HasR
 #undef HasL

 void HSort(int *m,int n)
 {int i,tt;
  for(i=n-1;i>=0;i--)Heapify(m,i,n);//1
  for(n--;n>0;n--){SWAP(m[0],m[n],tt); Heapify(m,0,n);}//2
 }
```

```
276  void CSort(int *m,int n, int *b,int B, int *r)
277  {int i;
278   memset(b,0,(B+1)*sizeof(int));
279   for(i=0;i<n;i++)b[m[i]]++;
280   for(i=1;i<=B;i++)b[i]+=b[i-1];
281   for(i=n-1;i>=0;i--)r[--b[m[i]]]=m[i];
282  }
283
284  void DSort(int *m_,int n, int *b,int B, int *r)
285  {int i; typedef union US2I_{unsigned int i; unsigned short int s[2];}US2I;
286   US2I *m=(US2I*)m_;
287   memset(b,0,(B+1)*sizeof(int));
288   for(i=0;i<n;i++)b[m[i].s[0]]++;
289   for(i=1;i<=B;i++)b[i]+=b[i-1];
290   for(i=n-1;i>=0;i--)r[--b[m[i].s[0]]]=m[i].i;
291   //
292   memcpy(m,r,n*sizeof(int));
293   //
294   memset(b,0,(B+1)*sizeof(int));
295   for(i=0;i<n;i++)b[m[i].s[1]]++;
296   for(i=1;i<=B;i++)b[i]+=b[i-1];
297   for(i=n-1;i>=0;i--)r[--b[m[i].s[1]]]=m[i].i;
298  }
299
300  int main(void)
301  {int *a0=NULL,*a=NULL,*b=NULL,*t=NULL,*b_=NULL, n=1000*1000*100,N=1,i; time_t t0,t1;
302   a0=(int*)malloc(n*sizeof(int));
303   a=(int*)malloc(n*sizeof(int));
304   b=(int*)malloc(n*sizeof(int));
305   b_=(int*)malloc(n*sizeof(int));
306   t=(int*)malloc(n*sizeof(int));
307   for(i=0;i<n;i++)a0[i]=((rand()+(rand()<<15))%n)&(~(1<<31));//0...RAND_MAX
308   //--
309   if(0)
310   {
311    time(&t0);
312    for(i=0;i<N;i++)
313    {
314     memcpy(a,a0,n*sizeof(int));
315     BSort(a,n);
316    }
317    time(&t1);
318    printf("BSort:dt=%d\n",(int)(t1-t0));
319    for(i=1;i<n;i++)if(a[i-1]>a[i])printf("Error1: i=%d\n",i);
320   }
321   //--
322   time(&t0);
323   for(i=0;i<N;i++)
324   {
325    memcpy(a,a0,n*sizeof(int));
326    MSort(a,n,t);
327   }
328   time(&t1);
329   printf("MSort:dt=%d\n",(int)(t1-t0));
330   for(i=1;i<n;i++)if(a[i-1]>a[i])printf("Error1: i=%d\n",i);
331   //--
332   if(1){
333   time(&t0);
334   for(i=0;i<N;i++)
335   {
336    memcpy(b,a0,n*sizeof(int));
337    MSort2(b,n,t);
338   }
339   time(&t1);
340   printf("MSort2:dt=%d\n",(int)(t1-t0));
341   for(i=0;i<n;i++)if(b[i]!=a[i])printf("Error2: i=%d\n",i);
342   }
343   //--
344   if(1){
```

```
345     time(&t0);
346     for(i=0;i<N;i++)
347     {
348      memcpy(b,a0,n*sizeof(int));
349      MSortX(b,n,t);
350     }
351     time(&t1);
352     printf("MSortX:dt=%d\n",(int)(t1-t0));
353     for(i=0;i<n;i++)if(b[i]!=a[i])printf("ErrorX: i=%d\n",i);
354     }
355     //--
356     if(1){
357     time(&t0);
358     for(i=0;i<N;i++)
359     {
360      memcpy(b,a0,n*sizeof(int));
361      MSort2X(b,n,t);
362     }
363     time(&t1);
364     printf("MSort2X:dt=%d\n",(int)(t1-t0));
365     for(i=0;i<n;i++)if(b[i]!=a[i])printf("Error2X: i=%d\n",i);
366     }
367     //--
368     if(1){
369     time(&t0);
370     for(i=0;i<N;i++)
371     {
372      memcpy(b,a0,n*sizeof(int));
373      QSort1(b,0,n-1);
374     }
375     time(&t1);
376     printf("QSort1:dt=%d\n",(int)(t1-t0));
377     for(i=0;i<n;i++)if(b[i]!=a[i])printf("ErrorQ1: i=%d\n",i);
378     }
379     //--
380     if(1){
381     time(&t0);
382     for(i=0;i<N;i++)
383     {
384      memcpy(b,a0,n*sizeof(int));
385      QSort1X(b,n);
386     }
387     time(&t1);
388     printf("QSort1X:dt=%d\n",(int)(t1-t0));
389     for(i=0;i<n;i++)if(b[i]!=a[i])printf("ErrorQ1X: i=%d\n",i);
390     }
391     //--
392     if(1){
393     time(&t0);
394     for(i=0;i<N;i++)
395     {
396      memcpy(b,a0,n*sizeof(int));
397      QSort2(b,0,n-1);
398     }
399     time(&t1);
400     printf("QSort2:dt=%d\n",(int)(t1-t0));
401     for(i=0;i<n;i++)if(b[i]!=a[i])printf("ErrorQ2: i=%d\n",i);
402     }
403     //--
404     if(1){
405     time(&t0);
406     for(i=0;i<N;i++)
407     {
408      memcpy(b,a0,n*sizeof(int));
409      QSort2X(b,n);
410     }
411     time(&t1);
412     printf("QSort2X:dt=%d\n",(int)(t1-t0));
413     for(i=0;i<n;i++)if(b[i]!=a[i])printf("ErrorQ2X: i=%d\n",i);
```

```
414     }
415     //--
416     if(1){
417     time(&t0);
418     for(i=0;i<N;i++)
419     {
420      memcpy(b,a0,n*sizeof(int));
421      HSort(b,n);
422     }
423     time(&t1);
424     printf("HSort:dt=%d\n",(int)(t1-t0));
425     for(i=0;i<n;i++)if(b[i]!=a[i])printf("HSort: i=%d\n",i);
426     }
427     //--
428     if(0){
429     time(&t0);
430     for(i=0;i<N;i++)
431     {
432      memcpy(b_,a0,n*sizeof(int));
433      CSort(b_,n, t,1<<16, b);
434     }
435     time(&t1);
436     printf("CSort:dt=%d\n",(int)(t1-t0));
437     for(i=0;i<n;i++)if(b[i]!=a[i]){printf("CSort: i=%d\n",i);break;}
438     }
439     //--
440     if(1){
441     time(&t0);
442     for(i=0;i<N;i++)
443     {
444      memcpy(b_,a0,n*sizeof(int));
445      DSort(b_,n, t,1<<16, b);
446     }
447     time(&t1);
448     printf("DSort:dt=%d\n",(int)(t1-t0));
449     for(i=0;i<n;i++)if(b[i]!=a[i]){printf("DSort: i=%d\n",i);break;}
450     }
451     //--
452     free(a);a=NULL;free(a0);a0=NULL;free(b);b=NULL;free(t);t=NULL;free(b_);b_=NULL;
453     printf("done\n");
454     return 0;
455    }
```