

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <time.h>
5  void Merge(int *a1,int n1, int *a2,int n2, int *r)
6  {int i1,i2,i;
7   for(i1=0,i2=0,i=0;i1<n1&& i2<n2;)
8     if(a1[i1]<a2[i2])
9       r[i++]=a1[i1++];
10    else
11      r[i++]=a2[i2++];
12    while(i1<n1) r[i++]=a1[i1++];
13    while(i2<n2) r[i++]=a2[i2++];
14  }
15
16  void MSort(int *m,int n,int *t)
17  {int n1,n2;
18   if(n<=1) return;
19   n1=n/2; n2=n-n1;
20   MSort(m,n1,t);
21   MSort(m+n1,n2,t);
22   Merge(m,n1, m+n1,n2, t);
23   memcpy(m,t,n*sizeof(int));
24  }
25  #define SWAP(a,b) {tt=(a); (a)=(b); (b)=tt;}
26
27  void BSort(int *m,int n)
28  {int i,j,tt;
29   for(i=n-1;i>0;i--)
30     for(j=0;j<i;j++)
31       if(m[j]>m[j+1]) SWAP(m[j],m[j+1]);
32  }
33
34  void MSort2(int *m,int n,int *t)
35  {int i,k,k2,tt;
36   k=1;
37   { //m+i,k, m+i+k,k2
38     for(i=0,k2=k;i+1<n;i+=2)
39       if(m[i]>m[i+1]) SWAP(m[i],m[i+1]);
40   }
41   for(k=2;k<n;k*=2)
42     { //m+i,k, m+i+k,k2
43       for(i=0,k2=k;i+k<n;i+=2*k)
44         {
45           if(n-i-k<k2) k2=n-i-k;
46           Merge(m+i,k, m+i+k,k2, t);
47           memcpy(m+i,t,(k+k2)*sizeof(int));
48         }
49     }
50  }
51
52  void MSortX(int *m,int n,int *t)
53  {int n1,n2,n3,n4;
54   if(n<=1) return;
55   n1=n/2; n3=n-n1;
56   n2=n1/2;n1=n1-n2;
57   n4=n3/2;n3=n3-n4;
58  #pragma omp parallel sections
59  {
60  #pragma omp section
61    {MSort(m,n1,t);}
62  #pragma omp section
63    {MSort(m+n1,n2,t+n1);}
64  #pragma omp section
65    {MSort(m+n1+n2,n3,t+n1+n2);}
66  #pragma omp section
67    {MSort(m+n1+n2+n3,n4,t+n1+n2+n3);}
68  }
69  #pragma omp parallel sections

```

```

70     {
71     #pragma omp section
72     {Merge(m,n1, m+n1,n2, t);memcpy(m,t,(n1+n2)*sizeof(int));}
73     #pragma omp section
74     {Merge(m+n1+n2,n3, m+n1+n2+n3,n4,
75     t+n1+n2);memcpy(m+n1+n2,t+n1+n2,(n3+n4)*sizeof(int));}
76     }
77     Merge(m,n1+n2, m+n1+n2,n3+n4, t);
78     memcpy(m,t,n*sizeof(int));
79     }
80     void MSort2X(int *m,int n,int *t)
81     {int i,k,k2,tt;
82     k=1;
83     //m+i,k, m+i+k,k2
84     #pragma omp parallel for private(tt)
85     for(i=0;i<n-1;i+=2)
86     if(m[i]>m[i+1])SWAP(m[i],m[i+1]);
87     }
88     for(k=2;k<n;k*=2)
89     //m+i,k, m+i+k,k2
90     #pragma omp parallel for private(k2)
91     for(i=0;i<n-k;i+=2*k)
92     {
93     k2=k;
94     if(n-i-k<k2)k2=n-i-k;
95     Merge(m+i,k, m+i+k,k2, t+i);
96     memcpy(m+i,t+i,(k+k2)*sizeof(int));
97     }
98     }
99     }
100
101     #undef SWAP
102     #define SWAP(a,b,tt) {tt=(a); (a)=(b); (b)=tt;}
103
104     void QSort1(int *m,int p,int q)
105     {int i,j,tt;
106     if(p>=q)return;
107     i=p;j=q;//M=m[j], [p,i-1]<=M<=[j,q]
108     while(1)
109     {
110     while(m[i]<m[j])i++;//M=m[j], [p,i-1]<=M<=[j,q]
111     SWAP(m[i],m[j],tt);//M=m[i], [p,i]<=M<=[j,q]
112     j--;//M=m[i], [p,i]<=M<=[j+1,q]
113     if(i>=j)
114     {
115     //i==j: [p,j]<=M<=[j+1,q]
116     //i==j+1: [p,j]<=M<=[j+1,q]
117     QSort1(m,p,j);QSort1(m,j+1,q);return;
118     }
119     while(m[i]<m[j])j--;//M=m[i], [p,i]<=M<=[j+1,q]
120     SWAP(m[i],m[j],tt);//M=m[j], [p,i]<=M<=[j,q]
121     i++;//M=m[j], [p,i-1]<=M<=[j,q]
122     if(i>=j)
123     {
124     //i==j (M=m[i]): [p,j]<=M<=[j+1,q]
125     //i==j+1: [p,j]<=M<=[j+1,q]
126     QSort1(m,p,j);QSort1(m,j+1,q);return;
127     }
128     }
129     }
130
131     int QSort1_(int *m,int p,int q)
132     {int i,j,tt;
133     if(p>=q)return p;
134     i=p;j=q;//M=m[j], [p,i-1]<=M<=[j,q]
135     while(1)
136     {
137     while(m[i]<m[j])i++;//M=m[j], [p,i-1]<=M<=[j,q]

```

```

138     SWAP(m[i],m[j],tt); //M=m[i], [p,i]<=M<=[j,q]
139     j--; //M=m[i], [p,i]<=M<=[j+1,q]
140     if(i>=j)
141     {
142         //i==j: [p,j]<=M<=[j+1,q]
143         //i==j+1: [p,j]<=M<=[j+1,q]
144         return j;
145     }
146     while(m[i]<m[j]) j--; //M=m[i], [p,i]<=M<=[j+1,q]
147     SWAP(m[i],m[j],tt); //M=m[j], [p,i]<=M<=[j,q]
148     i++; //M=m[j], [p,i-1]<=M<=[j,q]
149     if(i>=j)
150     {
151         //i==j (M=m[i]): [p,j]<=M<=[j+1,q]
152         //i==j+1: [p,j]<=M<=[j+1,q]
153         return j;
154     }
155 }
156 }
157
158 void QSort1X(int *m,int n)
159 {int j[9]={-1,0,0,0,0,0,0,0,n-1},i; // {j[k]+1,j[k]}
160   j[4]=QSort1_(m,0,n-1);
161   #pragma omp parallel sections
162   {
163     #pragma omp section
164     {j[2]=QSort1_(m,0,j[4]);}
165     #pragma omp section
166     {j[6]=QSort1_(m,j[4]+1,j[8]);}
167   }
168   #pragma omp parallel sections
169   {
170     #pragma omp section
171     {j[1]=QSort1_(m,0,j[2]);}
172     #pragma omp section
173     {j[3]=QSort1_(m,j[2]+1,j[4]);}
174     #pragma omp section
175     {j[5]=QSort1_(m,j[4]+1,j[6]);}
176     #pragma omp section
177     {j[7]=QSort1_(m,j[6]+1,j[8]);}
178   }
179   #pragma omp parallel for
180   for(i=0;i<8;i++)
181     QSort1(m,j[i]+1,j[i+1]);
182 }
183
184 void QSort2(int *m,int p,int q)
185 {int i,j,M,tt;
186 l1:
187   if(p>=q) return;
188   if(q-p>20)
189   {
190     i=p+rand()% (q-p+1);
191     SWAP(m[p],m[i],tt);
192   }
193   i=p;j=q;M=m[i]; // [p,i-1]<=M<=[j+1,q]
194   while(1)
195   {
196     while(m[i]<M) i++; // [p,i-1]<=M<=[j+1,q]
197     while(M<m[j]) j--; // [p,i-1]<=M<=[j+1,q]
198     if(i>=j)
199     {
200       //i==j (M=m[i]): [p,j]<=[j+1,q]
201       //i==j+1: [p,j]<=[j+1,q]
202       if(q-j>j-p)
203       {
204         QSort2(m,p,j);
205         p=j+1;
206         goto l1;

```

```

207     }
208     else
209     {
210         QSort2(m,j+1,q);
211         q=j;
212         goto l1;
213     }
214     //return;
215 }
216 SWAP(m[i],m[j],tt);//[p,i]<=M<=[j,q]
217 i++;j--;//[p,i-1]<=M<=[j+1,q]
218 }
219 }
220
221 void QSort2X(int *m,int n)
222 {int j[9]={-1,0,0,0, 0, 0,0,0,n-1},i;//{j[k]+1,j[k]}
223 j[4]=QSort1_(m,0,n-1);
224 #pragma omp parallel sections
225 {
226 #pragma omp section
227 {j[2]=QSort1_(m,0,j[4]);}
228 #pragma omp section
229 {j[6]=QSort1_(m,j[4]+1,j[8]);}
230 }
231 #pragma omp parallel sections
232 {
233 #pragma omp section
234 {j[1]=QSort1_(m,0,j[2]);}
235 #pragma omp section
236 {j[3]=QSort1_(m,j[2]+1,j[4]);}
237 #pragma omp section
238 {j[5]=QSort1_(m,j[4]+1,j[6]);}
239 #pragma omp section
240 {j[7]=QSort1_(m,j[6]+1,j[8]);}
241 }
242 #pragma omp parallel for
243 for(i=0;i<8;i++)
244     QSort2(m,j[i]+1,j[i+1]);
245 }
246
247 #define c m[i]
248 #define l m[2*i+1]
249 #define r m[2*i+2]
250 #define HasR (2*i+2<n)
251 #define HasL (2*i+1<n)
252
253 void Heapify(int *m,int i,int n)
254 {int tt;
255 while(HasR)
256 {
257     if(c>=l&&c>=r) return;
258     if(l>r) {SWAP(c,l,tt);i=2*i+1;}
259     else {SWAP(c,r,tt);i=2*i+2;}
260 }
261 if(HasL && l>c) SWAP(l,c,tt);
262 }
263
264 #undef c
265 #undef l
266 #undef r
267 #undef HasR
268 #undef HasL
269
270 void HSort(int *m,int n)
271 {int i,tt;
272 for(i=n-1;i>=0;i--) Heapify(m,i,n); //1
273 for(n--;n>0;n--) {SWAP(m[0],m[n],tt); Heapify(m,0,n);} //2
274 }
275

```

```

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 }

```