

```

1  #include <stdio.h>
2  #include <time.h>
3  #include <stdlib.h>
4  #include <limits.h>
5  #include <string.h>
6
7  int bubble_sort(int *A, int n)
8  {
9      int i,newn;
10     int temp;
11
12     while(1){
13         newn = -1;
14         for(i=0;i<n-1;i++){
15             if(A[i]>A[i+1]){
16                 temp = A[i]; A[i] = A[i+1]; A[i+1] = temp;
17                 newn = i;
18             }
19         }
20         if(newn==-1) break;
21     }
22     return 0;
23 }
24
25 int insertion_sort(int *A, int n)
26 {
27     int i,j;
28     int key;
29
30     for(i=1;i<n;i++){
31         key = A[i];
32         j = i - 1;
33         while ((j>=0) && (A[j]>key)){
34             A[j+1] = A[j];
35             j = j-1;
36         }
37         A[j+1] = key;
38     }
39     return 0;
40 }
41
42 int selection_sort(int *A, int n)
43 {
44     int i,j,mini;
45     int temp;
46
47     for(i=0;i<n-1;i++){
48         mini = i;
49         for(j=i+1;j<n;j++){
50             if(A[j]<A[mini]){
51                 mini=j;
52             }
53         }
54         if(mini!=i){
55             temp = A[i];
56             A[i] = A[mini];
57             A[mini] = temp;
58         }
59     }
60     return 0;
61 }
62
63 static int * L;
64 static int * R;
65
66 int merge(int *A, int p, int q, int r)

```

```

67 {
68     int i,j,k;
69     //     for(k=p;k<=q;k++){
70     //         L[k-p] = A[k];
71     //     }
72     memcpy(L,&A[p],(q-p+1)*sizeof(int));
73     L[q-p+1] = INT_MAX;
74     //     for(k=q+1;k<=r;k++){
75     //         R[k-q-1] = A[k];
76     //     }
77     memcpy(R,&A[q+1],(r-q)*sizeof(int));
78     R[r-q] = INT_MAX;
79     i = 0;
80     j = 0;
81     for(k=p;k<=r;k++){
82         if(L[i]<=R[j]){
83             A[k] = L[i];
84             i++;
85         }
86         else{
87             A[k] = R[j];
88             j++;
89         }
90     }
91
92     return 0;
93 }
94
95 int m_sort(int *A, int p, int r)
96 {
97     int q;
98
99     if(p<r){
100         q = (p+r)/2;
101         m_sort(A,p,q);
102         m_sort(A,q+1,r);
103         merge(A,p,q,r);
104     }
105     return 0;
106 }
107
108 int merge_sort(int *A, int n)
109 {
110     L = malloc((n/2+n%2+1)*sizeof(int));
111     R = malloc((n/2+1)*sizeof(int));
112
113     m_sort(A,0,n-1);
114
115     free(L);
116     free(R);
117     return 0;
118 }
119
120 void exchange(int *A, int i, int j)
121 {
122     int temp;
123
124     if(i!=j){
125         temp = A[i];
126         A[i] = A[j];
127         A[j] = temp;
128     }
129 }
130
131 int partition(int *A, int p, int r)
132 {

```

```

133     int x;
134     int i, j;
135
136     x = A[r];
137     i = p - 1;
138     for(j=p; j<r; j++){
139         if(A[j]<=x){
140             i++;
141             exchange(A, i, j);
142         }
143     }
144     exchange(A, i+1, r);
145     return(i+1);
146 }
147
148 int sort(int *A, int p, int r)
149 {
150     int q;
151     if(p<r){
152         q = partition(A, p, r);
153         sort(A, p, q-1);
154         sort(A, q+1, r);
155     }
156     return 0;
157 }
158
159 int quick_sort(int *A, int n)
160 {
161
162     sort(A, 0, n-1);
163
164     return 0;
165 }
166
167 int main()
168 {
169     int i, j, n;
170
171     // int maxn = 6;
172     // int A[maxn];
173     // A[0] = 5; A[1] = 2; A[2] = 4;
174     // A[3] = 6; A[4] = 1; A[5] = 3;
175     // clock_t start = clock(); merge_sort(A, maxn); clock_t end = clock();
176     // float M_seconds = (float)(end - start) / CLOCKS_PER_SEC;
177     // printf("\n T(%7d) = %8.3f\n", maxn, M_seconds);
178     // for(i=0; i<maxn; i++) printf("%d ", A[i]);
179     // return 0;
180
181
182     int maxn=160000;
183     int A[maxn];
184     int B[maxn];
185
186     srand(time(NULL));
187
188     for (i=0; i<maxn; i++){
189         A[i] = rand();
190         B[i] = A[i];
191     }
192
193     n = 10000;
194     printf("\n          |   B   |   I   |   S   |   M   |   Q\n");
195     printf("-----+-----+-----+-----+-----\n");
196     for(j=1; j<5; j++){
197         n = n*2;
198         // Bubble sort

```

```

199     clock_t start = clock(); bubble_sort(A,n); clock_t end = clock();
200     float B_seconds = (float)(end - start) / CLOCKS_PER_SEC;
201     memcpy(A,B,n*sizeof(int));
202     // Insertion sort
203     start = clock(); insertion_sort(A,n); end = clock();
204     float I_seconds = (float)(end - start) / CLOCKS_PER_SEC;
205     memcpy(A,B,n*sizeof(int));
206     // Selection sort
207     start = clock(); selection_sort(A,n); end = clock();
208     float S_seconds = (float)(end - start) / CLOCKS_PER_SEC;
209     memcpy(A,B,n*sizeof(int));
210     // Merge sort
211     start = clock(); merge_sort(A,n); end = clock();
212     float M_seconds = (float)(end - start) / CLOCKS_PER_SEC;
213     memcpy(A,B,n*sizeof(int));
214     // Quick sort
215     start = clock(); quick_sort(A,n); end = clock();
216     float Q_seconds = (float)(end - start) / CLOCKS_PER_SEC;
217     // Scores
218     printf(" %7d | %8.3f | %8.3f | %8.3f | %8.3f | %8.3f \n",n,B_seconds,I_seconds,S_seconds,M_seconds
,Q_seconds);
219     }
220
221 //     for(i=0;i<maxn;i++) printf("%d ",A[i]);
222 //     printf("\nZoran Ovcin\nmaxn = %d, sortiranje je trajalo %8.5f sekundi.\n",maxn,seconds);
223
224     return 0;
225 }

```