```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <locale.h>

//protótipos das funções
int menu();
void ImprimeVetor(int v[], int tam);
void LimpaBuffer();
void bubble(int v[], int n);          // n - tamanho do vetor
void insertion(int v[], int n);
void selection(int v[], int n);
void shell(int v[], int n);
void mergeSort(int vetor[], int inicio, int fim);
void quick(int v[], int c, int f);    // c - começo; f - fim

int trocas = 0, comparacoes = 0;

int main()
{
    setlocale(LC_ALL, "Portuguese");
    int n = 1, m = 0, sort;
    int i = 0, j, tam;

    printf("Digite o tamanho do vetor:\n");
    scanf("%d", &tam);
    LimpaBuffer();

    int v[tam];

    //Geração do vetor em ordem decrescente
    for (i = tam, j = 0 ; i > 0, j < tam; i--, j++)
        v[j] = i;

    printf("\n");
    for(i = 0; i < tam; i++)
        printf("%d ", v[i]);

    printf("\n\nVETOR GERADO COM SUCESSO!\n\n");
    system("pause");

    //int size = sizeof(v)/sizeof(v[0]);

    sort = menu();    //opção de ordenação

    clock_t Ticks[2];    // início da contagem de tempo de execução do programa
    Ticks[0] = clock();

    switch(sort)
    {
    case 1:
        bubble(v, tam);
        system("cls");
        ImprimeVetor(v,tam);
        printf("\nBUBBLE SORT\n\nEstabilidade: Estável\n\nComplexidade(Pior Caso): O(n^2)\n\
nIn-Place: Sim\n");
        break;
    case 2:
        insertion(v, tam);
        system("cls");
        ImprimeVetor(v,tam);
        printf("\nINSERTION SORT\n\nEstabilidade: Estável\n\nComplexidade(Pior Caso): O(n^2)\n\
nIn-Place: Sim\n");
        break;
    case 3:
        selection(v, tam);
        system("cls");
        ImprimeVetor(v,tam);
        printf("\nSELECTION SORT\n\nEstabilidade: Não estável\n\nComplexidade(Pior Caso):
O(n^2)\n\nIn-Place: Sim\n");
        break;
    case 4:
        shell(v, tam);
        system("cls");
        ImprimeVetor(v,tam);
        printf("\nSHELL SORT\n\nEstabilidade: Não estável\n\nComplexidade(Pior Caso): O(n^2)\n\
nIn-Place: Sim\n");
        break;
    case 5:
        mergeSort(v, 0, tam-1);
```

```
 77          system("cls");
 78          ImprimeVetor(v,tam);
 79          printf("\nMERGE SORT\n\nEstabilidade: Estável\nComplexidade(Pior Caso): O(nlog(n))\nIn-
Place: Não\n");
 80          break;
 81      case 6:
 82          quick(v, 0, tam);
 83          system("cls");
 84          ImprimeVetor(v,tam);
 85          printf("\nQUICK SORT\n\nEstabilidade: Não estável\n\nComplexidade(Pior Caso): O(n^2)\n\
nIn-Place: Não\n");
 86          break;
 87      }
 88        Ticks[1] = clock();
 89      //Fim da contagem de tempo de execução
 90      double Tempo = (Ticks[1] - Ticks[0])/(CLOCKS_PER_SEC/1000);
 91      printf("\nTempo gasto: %g ms.\n", Tempo);
 92      getchar();
 93      printf("\nComparações: %d\n\nTrocas: %d\n", comparacoes, trocas);
 94      return 0;
 95  }
 96
 97  int menu()
 98  {
 99      system("cls");
100      int sort;
101      do
102      {
103          printf("\n=== OPÇÕES DE ORGANIZAÇÃO ===");
104          printf("\n1== Bubble Sort");
105          printf("\n2== Insertion Sort");
106          printf("\n3== Selection Sort");
107          printf("\n4== Shell Sort");
108          printf("\n5== Merge Sort");
109          printf("\n6== Quick Sort");
110          printf("\n\nMétodo: ");
111          scanf("%d", &sort);
112      }
113      while(sort > 7 && sort < 0);
114
115      return sort;
116  }
117
118  void bubble(int v[], int n)
119  {
120      int k, j, aux;
121      for (k = 1; k < n; k++)
122      {
123          for (j = 0; j < n; j++)
124          {
125              comparacoes++;
126              if (v[j] > v[j + 1])
127              {
128                  aux = v[j];
129                  trocas++;
130                  v[j] = v[j + 1];
131                  trocas++;
132                  v[j + 1] = aux;
133                  trocas++;
134              }
135          }
136      }
137  }
138
139  void insertion(int v[], int n)
140  {
141      int i, j, atual;
142
143      for(j = 1; j < n; j++)
144      {
145          trocas++;
146          atual = v[j];
147          i = j-1;
148          comparacoes++;
149          while(i >= 0 && v[i] > atual)
150          {
151              trocas++;
152              v[i+1] = v[i];
153              i--;
154          }
```

```
155         trocas++;
156         v[i+1] = atual;
157     }
158 }
159 
160 void troca(int *x, int *y)
161 {
162     int temp = *x;
163     trocas++;
164     *x = *y;
165     trocas++;
166     *y = temp;
167     trocas++;
168 }
169 
170 void selection(int v[], int n)
171 {
172     int i, j, min;
173     for (i = 0; i < n-1; i++)
174     {
175         min = i;
176         for (j = i+1; j < n; j++)
177         {
178             comparacoes++;
179             if (v[j] < v[min])
180             {
181                 min = j;
182             }
183         }
184         troca(&v[min], &v[i]);
185     }
186 }
187 
188 void shell(int v[], int n)
189 {
190     for (int h = n/2; h > 0; h /= 2)
191     {
192         for (int i = h; i < n; i += 1)
193         {
194             int temp = v[i];
195             trocas++;
196             int j;
197             comparacoes = comparacoes + 3;
198             for (j = i; j >= h && v[j - h] > temp; j -= h)
199                 v[j] = v[j - h];
200                 trocas++;
201             v[j] = temp;
202             trocas++;
203         }
204     }
205 }
206 
207 void merge(int vetor[], int inicio, int meio, int fim)
208 {
209     int inicio_A, inicio_B, inicio_Aux, tamanho;
210     int *vetAux;
211 
212     inicio_A = inicio;
213     inicio_B = meio+1;
214     inicio_Aux = 0;
215     tamanho = fim-inicio+1;
216 
217     vetAux = (int*)malloc(tamanho * sizeof(int));
218     while(inicio_A <= meio && inicio_B <= fim)
219     {
220         if(vetor[inicio_A] < vetor[inicio_B])
221         {
222             vetAux[inicio_Aux] = vetor[inicio_A];
223             inicio_A++;
224         }
225         else
226         {
227             vetAux[inicio_Aux] = vetor[inicio_B];
228             inicio_B++;
229         }
230         inicio_Aux++;
231     }
232     while(inicio_A <= meio)
233     {
234         vetAux[inicio_Aux] = vetor[inicio_A];
```

```
235         inicio_Aux++;
236         inicio_A++;
237     }
238     while(inicio_B <= fim)
239     {
240         vetAux[inicio_Aux] = vetor[inicio_B];
241         inicio_Aux++;
242         inicio_B++;
243     }
244     for(inicio_Aux = inicio; inicio_Aux <= fim; inicio_Aux++)
245     {
246         vetor[inicio_Aux] = vetAux[inicio_Aux-inicio];
247     }
248     free(vetAux);
249 }
250 
251 void mergeSort(int vetor[], int inicio, int fim)
252 {
253     if (inicio < fim)
254     {
255         int meio = ( fim + inicio ) / 2;
256         mergeSort(vetor, inicio, meio);
257         mergeSort(vetor, meio+1, fim);
258         merge(vetor, inicio, meio, fim);
259     }
260 }
261 
262 void quick(int v[], int c, int f)
263 {
264     int i, j, pivo, aux;
265     i = c;
266     j = f-1;
267     pivo = v[(c + f) / 2];
268     comparacoes++;
269     while(i <= j)
270     {
271         comparacoes++;
272         comparacoes++;
273         while(v[i] < pivo && i < f)
274         {
275             i++;
276         }
277         comparacoes++;
278         comparacoes++;
279         while(v[j] > pivo && j > c)
280         {
281             j--;
282         }
283 
284         if(i <= j)
285         {
286             aux = v[i];
287             trocas++;
288             v[i] = v[j];
289             trocas++;
290             v[j] = aux;
291             trocas++;
292             i++;
293             j--;
294         }
295     }
296     comparacoes++;
297     if(j > c)
298         quick(v, c, j+1);
299     if(i < f)
300         quick(v, i, f);
301 }
302 
303 void ImprimeVetor(int v[], int tam)
304 {   int i;
305     printf("\n\nVetor final : { ");
306     for(i = 0; i < tam; i++)
307     {
308         printf("%d ", v[i]);
309     }
310     printf("}\n");
311 }
312 
313 void LimpaBuffer()
314 {
```

```
315       setbuf(stdin,NULL);
316   }
```