```c
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#define n 1000000
//             Mikeias Silva Gomes de Azevedo RA: 15058923

int vetor[n];
 criaVetor(){
    srand(time(NULL));
    int i;
    for(i = 0; i<n; i++){
        vetor[i]= rand()%n-1;
    }
}


bubblesort(){
    int cont =1;
    int i, j;
    int aux;
    for(i=0; i<n; i++){
        for(j=0; j<n; j++){
            if(vetor[j] > vetor[j+1]){
                aux=vetor[j];
                vetor[j]=vetor[j+1];
                vetor[j+1]=aux;
            }
        }
       cont++;
    }
}

selectionSort(){
    int menor, aux;
    int i, j;

    for(i = 0; i<n-1; i++){
            menor =i;
        for(j=i+1; j<n; j++){
            if(vetor[menor] > vetor[j])
                menor = j;
        }
            if(i != menor){
                aux=vetor[i];
                vetor[i]=vetor[menor];
                vetor[menor]=aux;
            }
        }
    }

int particionar(int esquerda, int direita){
    int i, aux;
    int cont = esquerda;
    for(i = esquerda+1; i<=direita; i++){
        if(vetor[i] < vetor[esquerda]){
            cont++;
            aux=vetor[i];
            vetor[i]=vetor[cont];
            vetor[cont]=aux;
        }
    }
    aux=vetor[esquerda];
    vetor[esquerda]=vetor[cont];
    vetor[cont]=aux;
    return cont;
}
```

```
 67
 68  quicksort(int esquerda, int direita){
 69      int temp;
 70
 71      if(esquerda<direita){
 72          temp = particionar(esquerda, direita);
 73          quicksort(esquerda, temp-1);
 74          quicksort(temp+1, direita);
 75      }
 76
 77  }
 78
 79  insertionSort(){
 80      int i;
 81      int atual;
 82      for(i = 0; i<n; i++){
 83          atual = vetor[i];
 84          int j = i-1;
 85          while(j>=0 && vetor[j] >= atual){
 86              vetor[j+1] = vetor[j];
 87              j--;
 88          }
 89          vetor[j+1] = atual;
 90      }
 91  }
 92
 93  merges(int inicio, int meio, int fim){
 94      int *temp, p1, p2, tamanho, i, j, k;
 95      int fim1 = 0, fim2;
 96      tamanho = fim-inicio+1;
 97
 98      p1 = inicio;
 99      p2 = meio + 1;
100      temp = (int *) malloc(tamanho*sizeof(int));
101      if(temp != NULL){
102          for(int i=0; i<tamanho; i++){
103              if(!fim1 && !fim2){
104                  if(vetor[p1] < vetor[p2])
105                      temp[i]=vetor[p1++];
106                  else
107                      temp[i]=vetor[p2++];
108
109                  if(p1>meio)
110                      fim1=1;
111                  if(p2>fim)
112                      fim2=1;
113            }else{
114                if(!fim1)
115                  temp[i]=vetor[p1++];
116                else
117                  temp[i]=vetor[p2++];
118                  }
119          }
120          for(int j = 0, k=inicio; j<tamanho; j++, k++)
121              vetor[k]=temp[j];
122      }
123      free(temp);
124  }
125
126  mergeSort(int inicio, int fim){
127      int meio;
128      if(inicio < fim){
129          meio = ((inicio+fim)/2);
130          mergeSort(inicio, meio);
131          mergeSort(meio+1, fim);
132          merges(inicio, meio, fim);
```

```
133     }
134 }
135 troca(int j, int aposJ){
136 int aux = vetor[j];
137 vetor[j]= vetor[aposJ];
138 vetor[aposJ]=aux;
139 }
140 
141 
142 maxheapify(int pos, int tamanhoDoVetor){
143     int maxh = 2 * pos + 1, direita = maxh + 1;
144     if(maxh < tamanhoDoVetor){
145         if(direita< tamanhoDoVetor && vetor[maxh] < vetor [direita]){
146             maxh = direita;
147         }
148         if(vetor[maxh] > vetor[pos]){
149             troca(maxh, pos);
150             maxheapify(maxh, tamanhoDoVetor);
151         }
152     }
153 }
154 
155 criaheap(){
156     int i;
157     int j = n;
158     for(i = n / 2 - 1; i >= 0; i--){
159 
160         maxheapify(i, j);
161     }
162 }
163 
164 heapSort(){
165     criaheap();
166     int j = n;
167 
168     for(int i = n -1; i> 0; i--){
169         troca(i, 0);
170         maxheapify(0, --j);
171     }
172 }
173 
174 
175 /*imprime(){
176 int i;
177     for(i = 0; i<n; i++){
178         if(i%10 == 0){
179             printf("\n");
180         }
181         printf(" %3d", vetor[i]);
182     }
183 }*/
184 
185 
186 int main(){
187 clock_t tempo1, tempo2;
188 //BUBBLESORT
189 criaVetor();
190 tempo1 = clock();
191 bubblesort();
192 tempo2 = clock()- tempo1;
193 printf("Ordenado pelo bubble sort | tempo gasto: %f .", (float) tempo2/
CLOCKS_PER_SEC);
194 printf("\n\n\n");
195 
196 //insertion sort
197 criaVetor();
```

```
198  tempo1 = clock();
199  insertionSort();
200  tempo2 = clock()-tempo1;
201  printf("\nOrdenado pelo Insertion Sort | tempo gasto: %f .", (float) tempo2/
CLOCKS_PER_SEC);
202  printf("\n\n\n");
203
204  //Selection Sort
205  criaVetor();
206  tempo1 = clock();
207  selectionSort();
208  tempo2 = clock()-tempo1;
209  printf("\nOrdenado pelo Selection Sort | tempo gasto: %f .", (float) tempo2/
CLOCKS_PER_SEC);
210  printf("\n\n\n");
211
212  //HEAPSORT
213  criaVetor();
214  tempo1 = clock();
215  heapSort();
216  tempo2 = clock()-tempo1;
217  printf("\nOrdenado pelo Heap Sort | tempo gasto: %f .", (float) tempo2/
CLOCKS_PER_SEC);
218  printf("\n\n\n");
219
220  //merge sort
221  criaVetor();
222  tempo1 = clock();
223  mergeSort(0, n);
224  tempo2 = clock()-tempo1;
225  printf("\nOrdenado pelo Merge Sort | tempo gasto: %f .", (float) tempo2/
CLOCKS_PER_SEC);
226  printf("\n\n\n");
227
228  //Quick Sort
229  criaVetor();
230  tempo1 = clock();
231  quicksort(0, n);
232  tempo2 = clock()-tempo1;
233  printf("\nOrdenado pelo Quick Sort | tempo gasto: %f .", (float) tempo2/
CLOCKS_PER_SEC);
234  printf("\n\n\n");
235
236
237  return 0;
238  }
239
```