#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

#include <time.h>

#include <unistd.h>

#define tamanho 100000

void QuickSort(int\* v, int tam){

int j = tam, k;

if(tam <= 1)

return;

else {

int x = v[0];

int a = 1;

int b = tam - 1;

do{

while ((a < tam) && (v[a] <= x))

a++;

while (v[b] > x)

b--;

if (a < b) { // faz troca

int temp = v[a];

v[a] = v[b];

v[b] = temp;

a++;

b--;

}

} while(a <= b);

//troca pivo

v[0] = v[b];

v[b] = x;

//ordena sub-vetores restantes

QuickSort(v, b);

QuickSort(&v[a], tam - a);

}

}

void InsertionSort(int\* v, int tam){

int i, j, k, chave;

for (j = 1; j < tam; j++) {

chave = v[j];

i = j - 1;

while ((i >= 0) && (v[i] > chave)) {

v[i + 1] = v[i];

i--;

}

v[i + 1] = chave;

}

}

void MergeSort(int\* v, int inicio, int fim){

int i, j, k, meio, \* t, z;

if (inicio == fim)

return;

//ordenacao recurviva das duas metades

meio = (inicio + fim) / 2;

MergeSort(v, inicio, meio);

MergeSort(v, meio + 1, fim);

//intercalacao no vetor temporario t

i = inicio;

j = meio + 1;

k = 0;

t = (int\*)malloc(sizeof(int) \* (fim - inicio + 1));

while (i < meio + 1 || j < fim + 1){

if (i == meio + 1){

t[k] = v[j];

j++; k++;

}

else if (j == fim + 1){

t[k] = v[i];

i++; k++;

}

else if (v[i] < v[j]){

t[k] = v[i];

i++; k++;

}

else {

t[k] = v[j];

j++; k++;

}

}

//copia vetor intercalado para o vetor original

for (i = inicio; i <= fim; i++)

v[i] = t[i - inicio];

free(t);

}

int main(){

int vqs[tamanho]; // vetor para o QuickSort

int vis[tamanho]; // vetor para o HeapSort

int vms[tamanho]; // vetor para o MergeSort

FILE\* f1;

FILE\* f2;

FILE\* f3;

f1 = fopen("100k.txt", "r");

f2 = fopen("100k.txt", "r");

f3 = fopen("100k.txt", "r");

clock\_t start\_qs, end\_qs, start\_hs, end\_hs, start\_ms, end\_ms;

double cpu\_time\_used\_qs, cpu\_time\_used\_hs, cpu\_time\_used\_ms;

int c1 = 0;

while(c1 <= tamanho){

fscanf(f1, "%d\n", &vqs[c1]); // vetoriza os 100k de dados sem duplicidades

c1++;

}

start\_qs = clock();

QuickSort(vqs, tamanho);

end\_qs = clock();

cpu\_time\_used\_qs = ((double) (end\_qs - start\_qs)) / CLOCKS\_PER\_SEC;

int contador2 = 0;

while(contador2 < tamanho){

printf("%d, ", vqs[contador2]);

contador2++;

}

printf("Acima voce pode ver o vetor ordenado.\n");

printf("QuickSort tempo: %f\n", cpu\_time\_used\_qs);

sleep(3);

//--------------------------------------------------//

int c2 = 0;

while(c2 <= tamanho){

fscanf(f2, "%d\n", &vis[c2]); // vetoriza os 100k de dados sem duplicidades

c2++;

}

start\_hs = clock();

InsertionSort(vis, tamanho);

end\_hs = clock();

;

contador2 = 0;

while(contador2 < tamanho){

printf("%d, ", vis[contador2]);

contador2++;

}

cpu\_time\_used\_hs = ((double) (end\_hs - start\_hs)) / CLOCKS\_PER\_SEC;

printf("Acima voce pode ver o vetor ordenado.\n");

printf("InsertionSort tempo: %f\n", cpu\_time\_used\_hs);

sleep(3);

//----------------------------------------------------//

// adiociona os dados no vetor do mergesort

int c3 = 0;

while(c3 <= tamanho){

fscanf(f3, "%d\n", &vms[c3]); // vetoriza os 100k de dados sem duplicidades

c3++;

}

start\_ms = clock();

MergeSort(vms, 0, tamanho-1);

end\_ms = clock();

contador2 = 0;

while(contador2 < tamanho){

printf("%d, ", vms[contador2]);

contador2++;

}

cpu\_time\_used\_ms = ((double) (end\_ms - start\_ms)) / CLOCKS\_PER\_SEC;

printf("Acima voce pode ver o vetor ordenado.\n");

printf("MergeSort tempo: %f\n", cpu\_time\_used\_ms);

sleep(3);

fclose(f1);

fclose(f2);

fclose(f3);

}