**EQUIPE: TIAGO MURILO, RODRIGO CONINK, MAURICIO NUNES E LUCAS PINHO**

**Comparação em desempenho dos métodos de ordenação Bubble Sort, Insertion Sort, Selection Sort, Merge Sort e Quick Sort**

Após nossos estudos sobre os cinco métodos de ordenações ficam evidente que o método Quick Sort e o Merge Sort são os melhores, conforme é ilustrado nas figuras (1, 2, 3, 4 e 5) abaixo conseguimos identificar que sua quantidade de comparações e quantidade de trocas diminui significativamente.

****

Figura 1 - Gráfico de desempenho do método Bubble Sort

****

Figura 2 - Gráfico de desempenho do método Insertion Sort

  

Figura 3 - Gráfico de desempenho do método Selection Sort

****

Figura 4 - Gráfico de desempenho do método Merge Sort

****

Figura 5 - Gráfico de desempenho do método Quick Sort

**Atividade:**

Baseado no código abaixo, que gera números ordenados, invertidos e aleatórios, construa um programa para provar os resultados das cinco figuras acima.

Deve ser simulando com dez mil dados do tipo inteiro, conforme é realizado na função bubbleSort() do código abaixo.

/\*

PROCESSO AVALIATIVO N2 – ESTRUTURA DE DADOS

EQUIPE

NOMES:

CONSEGUIU CHEGAR NO RESULTADO?

R:

SE NÃO, POR QUE NÃO CONSEGUIU CHEGAR NO RESULTADO?

R.:

\*/

#include <stdio.h>

#include <time.h>

#define TAMANHO 10000

// Prototipo de Função

void geraNumero(int \*vet, int op);

void bubbleSort(int \*vet);

void imprimirVetor(int \*vet);

int main (void){

// Variáveis

int vet1[TAMANHO];

// Entrada de dados;

geraNumero(vet1,3);

// printf("Desordenado: \n\n");

// imprimirVetor(vet1);

// Processamentos dos dados

bubbleSort(vet1);

// Saída de dados

// printf("\n\nOrdenado: \n\n");

//imprimirVetor(vet1);

return 0;

}

// Função geração de números

void geraNumero(int \*vet, int op) {

int i, j;

switch (op) {

case 1: // Ordenados

for (i = 0; i < TAMANHO; i++) {

vet[i] = i + 1;

}

break;

case 2: //Invertidos

for (i = 0; i < TAMANHO; i++) {

vet[i] = TAMANHO-i;

}

break;

case 3: // Aleatórios

for (i = 0; i < TAMANHO; i++) {

vet[i] = (int) (rand() % TAMANHO);

}

break;

}

}

// Função ordenação bubble sort

void bubbleSort(int \*vet){

int n, troca, i, aux, qtd\_trocas, qtd\_comparacoes;

n = 1;

troca = 1;

qtd\_trocas = 0;

qtd\_comparacoes = 0;

// Ponto do algoritmo para iniciar o tempo de execução

float tempo\_inicial = clock();

while (n <= TAMANHO && troca == 1) {

troca = 0;

for (i = 0; i <= TAMANHO-2; i++) {

// Ponto do algoritmo para contar as comparações

qtd\_comparacoes++;

if (vet[i] > vet[i + 1]) {

// Ponto do algoritmo para contar as trocas

qtd\_trocas++;

troca = 1;

aux = vet[i];

vet[i] = vet[i + 1];

vet[i + 1] = aux;

}

}

n = n + 1;

}

// Ponto do algoritmo para calcular o tempo de execução

float tempo\_final = clock() - tempo\_inicial;

// Saída de dados

printf("\nQuantidade de comparacoes: %i\n",qtd\_comparacoes);

printf("Quantidade de trocas: %i\n",qtd\_trocas);

printf("Tempo de execucao do algoritmo: %.3f",tempo\_final/1000);

}

// Função impressão do vetor

void imprimirVetor(int \*vet){

int i;

for(i=0;i<TAMANHO;i++){

printf("%i, ",vet[i]);

if ((i+1)%14 == 0){

printf("\n");

}

}

}

Esta atividade pode ser realizada em equipes no máximo de três integrantes e deve ser entrega no dia 9/11/2020 até às 23:59 no AVA.

No AVA um membro da equipe deve postas o algoritmo, com o cabeçalho do código exemplo preenchido, nomes dos integrantes e se conseguiu chegar no resultado.

Bom trabalho.