Pontifícia Universidade Católica de Campinas
PUC Campinas

Trabalho apresentado por:
Gustavo Moraes Mercedes RA:21010281
Matheus Althman Hespagnola
RA:21007271
Pedro Zampieri RA:21001597
Nicolas Augusto de Almeida Bortoloto RA:21008596

Professora:
Lucia Filomena de Almeida Guimarães

- Nesse primeiro bloco, está contido os includes e defines do programa, além de estar sendo criado uma struct, que irá armazenar altura, peso e idade.

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

#define swap( _a, _b ) do{ Data _tmp = _a; _a = _b; _b = _tmp; } while(0)
#define TAM 1000
#define SEED 34

typedef struct Data
{
    float altura,peso;
    int idade;
} Data;
```

- Nesse bloco está sendo mostrado a função printData, que em resumo, tem o objetivo de mostrar todos os dados que foram guardados, de todas as structs

```c
void printData(Data data[])
{
    int i=0;
    for(i; i<TAM; i++)
    {
        printf("\tAltura: %.2f m",data[i].altura);
        printf("\tPeso: %.2f Kg ",data[i].peso);
        printf("\tIdade: %d Anos",data[i].idade);
        printf("\n");
    }
}
```

- O fillData é uma função com o objetivo de preencher dados através de números aleatórios, pegando a seed (valor aleatório).

```
void fillData(Data data[])
{
    srand(SEED);

    for(int i=0; i<TAM; i++)
    {
        data[i].altura = 0.1 + rand() * (2.20 - 0.00) / RAND_MAX;   ;
        data[i].peso =0.00 + rand() * (200.00 - 15.00) / RAND_MAX;
        data[i].idade = (rand() % 101);

        printf("Altura: %.2f \n", data[i].altura);
        printf("Peso: %.2f \n", data[i].peso);
        printf("Idade: %d\n\n", data[i].idade);
    }
    return;
}
```

- A função bubbleSort serve para realizar a ordenação, seguindo o algoritmo do bubble sort. Vale detalhar que ordena de forma decrescente.

```
void bubbleSort (Data data[]) // Ordena pelo tamanho de forma decrescente
{
    printf("\n\n=======================BubbleSort============================\n\n");

    for (int cont = 1; cont < TAM; cont++)
    {

        for (int j = 0; j < TAM - 1; j++) {

            if (data[j].altura < data[j + 1].altura) {
                swap(data[j], data[j+1]);
            }
        }
    }
}
```

- A função particao é a logica que determina a quebra dos elementos de uma array, colocando-os em ordem.

```
int particao(Data data[],int LI,int LS,int J)
{
    Data A = data[LS];
    int inferior = LI, superior = LS;

    while(superior>inferior)
    {
        while(data[inferior].altura > A.altura)
        {
            inferior = inferior + 1;
        }

        while(data[superior].altura <=A.altura)
        {
            superior = superior - 1;
        }
        if(inferior < superior)
        {
            swap(data[superior], data[inferior]);
        }
    }
    swap(data[inferior], data[LS]);
    J = inferior;
    return J;
}
```

- O quicksort é uma função que realiza uma ordenação usando o algoritmo do quick sort, e a função “particao”, feita anteriormente.

```
void quickSort(Data data[],int LI,int LS)
{
    int J;
    if(LI < LS)
    {
        J = particao(data, LI, LS, J);
        quickSort(data,LI, J-1);
        quickSort(data, J+1, LS);
    }
}
```

- Por fim, o programa principal, que está responsável por rodar as funções feitas anteriormente, e mostrar o tempo em milissegundos da execução.

```
int main() {

    clock_t time;
    double time_execucao;

    Data data[TAM];
    time = clock();
    fillData(data);
    printData(data);

    //bubbleSort(data);
    quickSort(data, 0, TAM);
     time = clock() - time;
    printf("Tempo de execucao: %lf milisegundos", ((double)time)/((CLOCKS_PER_SEC/1000)));

    return 0;
}
```

Execução com vetor de tamanho 100			
Casos	Bubble sort (tempo em segundos)	Merge Sort (tempo em segundos)	Semente usada
1	0.001964	0.001778	5
2	0.002169	0.003504	10
3	0.004071	0.002131	11
4	0.00225	0.002662	12
5	0.002802	0.002465	13
Tempo médio	0,013256	0,002508	

Execução com vetor de tamanho 1000			
Casos	Bubble sort (tempo em segundos)	Merge Sort (tempo em segundos)	Semente usada
1	0.017744	0.020442	5
2	0.022989	0.022902	10
3	0.023659	0.02194	11
4	0.019806	0.016943	12
5	0.02179	0.017579	13
Tempo médio	0,021197	0,0199612	

Execução com vetor de tamanho 5000			
Casos	Bubble sort (tempo em segundos)	Merge Sort (tempo em segundos)	Semente usada
1	0.276845	0.074965	5
2	0.185316	0.0816	10
3	0.168444	0.069682	11
4	0.213118	0.12244	12
5	0.186337	0.074897	13
Tempo médio	0,206012	0,0847168	

Execução com vetor de tamanho 10000			
Casos	Bubble sort (tempo em segundos)	Merge Sort (tempo em segundos)	Semente usada
1	0.601289	0.146138	5
2	0.574068	0.140505	10
3	0.593025	0.14801	11
4	0.58035	0.196805	12
5	0.581915	0.147677	13
Tempo médio	0,586129	0,156736	

Execução com vetor de tamanho 50000			
Casos	Bubble sort (tempo em segundos)	Merge Sort (tempo em segundos)	Semente usada
1	7.8634	X	5
2	7.775	X	10
3	7.7526	X	11
4	7.6889	X	12
5	7.727	X	13
Tempo médio	7.76138	X	

Tamanho do vetor	Bubble sort (tempo médio em segundos)	Merge Sort (tempo médio em segundos)
100	0,013256	0,002508
1000	0,021197	0,0199612
5000	0,206012	0,0847168
10000	0,586129	0,156736
50000	7.76138	X

OBS: Quick Sort com tamanho de 50000 não rodou, acabou quebrando antes de ditar o tempo usado para a operação.