Aula 06 – Exercícios - Gabarito

1. Crie um programa para armazenar em um struct os dados do cliente (nome – caracteres, cpf – inteiro) e um vetor de no máximo 50 elementos. Realize as seguintes operações :

    - Em uma função cadastrar(), solicite a digitação e armazene alguns desses clientes ;
    - Usando o BubbleSort recursivo, faça uma função de ordenação alfabética crescente pelo nome do cliente ;
    - Exibir todos os dados dos clientes nesta nova ordem, novamente em uma função.

R:

```
// Aula 06 - Exercício01
#include <iostream>
using namespace std ;

//definir o registro
struct cliente{
      string nome ;
      int cpf ;
};

// função para cadastrar cliente
cliente cadastrar()
{
      cliente c ;
      cout << "Digite o nome: " ;
      cin >> c.nome ;
      cout << "Digite o CPF: " ;
      cin >> c.cpf ;

      return c ;
}

// função para exibir cliente
void exibir(cliente c)
{
      cout << "Nome: " << c.nome ;
      cout << " / CPF: " << c.cpf << endl ;
      return ;
}

// bubblesort() recursivo
void bubblesort (cliente arr[], int n)
{  if (n == 1) // Caso base
       return;
   // Um passo do bubble sort onde o menor
   // elemento flutua ("bubble") para início.
   cliente aux ;
   for (int i = 0; i < n-1; i++)
       if (arr[i + 1].nome < arr[i].nome) // troca o elemento
        {   aux = arr[i];
                arr[i] = arr[i + 1] ;
                arr[i + 1] = aux ; }
     // Menor elemento está fixado e
     // a recursão continua no array
    bubblesort (arr, n-1);
}
```

```
// programa principal
int main()
{
      // variáveis
      cliente vetor[50] ;
      int quant = 3 ;

      cout << "Aula06 - Exercicio01" << endl ;
      cout << "Cadastrar clientes"   << endl ;

      //entrada de dados
      for (int i=0; i<quant ; i++)
                vetor[i] = cadastrar() ;


      //processamento
      bubblesort(vetor,quant) ;

      //saída de dados
      for (int i=0; i<quant ; i++)
                exibir(vetor[i]) ;

    return 0 ;
} // fim main()
```

2. Crie outro programa semelhante agora usando o MergeSort recursivo.

R:

```
// Aula 06 - Exercício02
#include <iostream>
using namespace std ;

#define MAX 50

//definir o registro
struct cliente{
      string nome ;
      int cpf ;
};

// função para cadastrar cliente
cliente cadastrar()
{
      cliente c ;
      cout << "Digite o nome: " ;
      cin >> c.nome ;
      cout << "Digite o CPF: " ;
      cin >> c.cpf ;

      return c ;
}

// função para exibir cliente
void exibir(cliente c)
{
      cout << "Nome: " << c.nome ;
      cout << " / CPF: " << c.cpf << endl ;
}

// ordenação por intercalação
void intercala(int p, int q, int r, cliente v[])
```

```
{
	int i, j, k ;
	cliente w[MAX];
	i = p;
	j = q;
	k = 0;
	while (i < q && j < r) {
		if (v[i].nome < v[j].nome) {
			w[k] = v[i];
			i++;
		}
		else {
			w[k] = v[j];
			j++;
			}
		k++;
	}
	while (i < q) {
		w[k] = v[i];
		i++;
		k++;
	}
	while (j < r) {
		w[k] = v[j];
		j++;
		k++;
	}
	for (i = p; i < r; i++)
		v[i] = w[i-p];
}


void mergesort(int p, int r, cliente v[])
{
	int q;
	if (p < r - 1) {
		q = (p + r) / 2; // divisão inteira
		mergesort(p, q, v);
		mergesort(q, r, v);
		intercala(p, q, r, v);
	}
}

// programa principal
int main()
{
	// variáveis
	cliente vetor[MAX] ;
	int quant = 3 ;

	cout << "Aula06 - Exercicio02" << endl ;
	cout << "Cadastrar clientes"   << endl ;

	//entrada de dados
	for (int i=0; i<quant ; i++)
		vetor[i] = cadastrar() ;

	//processamento
	mergesort(0, quant, vetor) ;

	//saída de dados
	for (int i=0; i<quant ; i++)
```

```
            exibir(vetor[i]) ;

    return 0 ;
} // fim main()
```

3. Assinale a opção correta. O que faz a função F definida a seguir ?

```
int F (int n) {
    if (n == 0)
      return(0);
    return(n + F(n-1));
}
```

a) Não compila, pois não tem caso base.
X) Soma os inteiros de n até 0, inclusive.
c) Soma os inteiros de 0 até n-1, inclusive.
d) Calcula o fatorial de um inteiro n passado como parâmetro.
e) Soma os inteiros de 1 até n-1, inclusive.

4. Assinale a opção correta. Sobre o Mergesort é correto afirmar que :

X) É uma ordenação por intercalação que no pior caso tem complexidade $O(n \log_2 n)$.
b) Trabalha dividindo a lista apenas uma vez.
c) É uma ordenação por trocas que no pior caso tem complexidade $O(\log_2 n)$.
d) Só pode ser implementado de forma recursiva, pois usa a técnica de divisão e conquista.
e) Trabalha com um pivô para separar os dados que depois serão intercalados.

5. A ordenação por intercalação é um método eficiente baseado na técnica recursiva chamada Dividir para conquistar.

6. Analise as sentenças abaixo sobre recursividade e indique V para verdadeiro e F para falso :

(V) Um programa recursivo é um programa que chama a si mesmo.
(V) Uma função recursiva é definida em termos dela mesma.
(V) A recursividade pode ser classificada como direta ou indireta.
(V) Para cada chamada de uma função recursiva os parâmetros e as variáveis locais são empilhados na pilha de execução.
(F) A recursividade é a única alternativa para a implementação do algoritmo Merge Sort.

==========