Recursividade, Algoritmo de Ordenação Merge Sort

1. Recursividade

A recursividade é um conceito fundamental na programação pelo fato de permitir a resolução de problemas complexos, com estruturas repetitivas, por meio de blocos de códigos mais simples. Ela consiste na definição de uma função em termos dela mesma, ou seja, essa função possui a capacidade de chamar a si mesma como parte de seu próprio processo de execução.

A utilização de recursividade na solução de problemas computacionais possibilita a divisão do problema em casos menores. Essa redução do problema original em subproblemas menores tem como intenção facilitar a elaboração de uma resposta. Cada chamada recursiva irá trabalhar com uma parte menor do problema original até que o caso base seja encontrado e solucionado. A combinação dos resultados parciais permite a solução completa do problema.

Como exemplo, vamos analisar o algoritmo que calcula o fatorial de um número com e sem recursão:

Algoritmo sem recursão:

```
#include <stdio.h>

unsigned long fatorialIterativo(int n) {
  unsigned long resultado = 1;
  for (int i = 1; i <= n; i++) {
    resultado = resultado * i;
  }
  return resultado;
}

int main() {
  int nro;

  printf("Digite um número inteiro não negativo: ");
  scanf("%d", &nro);

  unsigned long resultado = fatorialIterativo(nro);

  printf("O fatorial de %d é: %lu \n", nro, resultado);

  return 0;
}
```

Algoritmo com recursão:

```
#include <stdio.h>
```

```c
unsigned long fatorialRecursivo(int n) {
  if (n == 0 || n == 1) {
    return 1; // Caso base: fatorial de 0 ou 1 é 1
  }
  return n * fatorialRecursivo(n - 1); // Caso recursivo
}

int main() {
  int nro;

  printf("Digite um número inteiro não negativo: ");
  scanf("%d", &nro);

  unsigned long resultado = fatorialRecursivo(nro);

  printf("O fatorial de %d é: %lu \n", nro, resultado);

  return 0;
}
```

Apesar de ser uma técnica bastante poderosa, a recursividade deve ser utilizada com cuidado, considerando-se questões de desempenho, consumo de memória, compreensão (legibilidade) do código e facilidade de depuração.

2. Algoritmo de ordenação Merge Sort

O Merge Sort, ou Ordenação por Intercalação, é um algoritmo que implementa a divisão da estrutura de dados em partes unitárias (indivisíveis), realiza a ordenação (chamada de intercalação) dessas partes e, em seguida, a combinação (agregação) delas para obter uma lista final ordenada. O algoritmo é baseado no princípio de “dividir para conquistar” e emprega recursividade para a realização da tarefa. O Merge Sort possui complexidade O (n log n).

Exemplo de implementação:

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

// Função para impressão do vetor
void impressao(char msg[], int array[], int t) {
  printf("%s", msg);
  for (int i = 0; i < t; i++) {
    printf("%d ", array[i]);
  }
  printf("\n");
}
```

```
// Função para mesclar duas sub-listas ordenadas
void merge(int array[], int esquerda, int meio, int direita) {
  int t1 = meio - esquerda + 1;
  int t2 = direita - meio;

  // Cria vetores temporários para armazenar as sub-listas
  int E[t1], D[t2];

  // Copia os elementos para os vetores temporários
  for (int i = 0; i < t1; i++)
    E[i] = array[esquerda + i];
  for (int j = 0; j < t2; j++)
    D[j] = array[meio + 1 + j];

  int i = 0, j = 0, k = esquerda;
  while (i < t1 && j < t2) {
    if (E[i] <= D[j]) {
      array[k] = E[i];  // Insere o elemento de E
      i++;
    } else {
      array[k] = D[j];  // Insere o elemento de D
      j++;
    }
    k++;
  }

  // Copia os elementos restantes de E e D, se houver
  while (i < t1) {
    array[k] = E[i];
    i++;
    k++;
  }

  while (j < t2) {
    array[k] = D[j];
    j++;
    k++;
  }
}

// Função Merge Sort
void mergeSort(int array[], int esquerda, int direita) {
```

```
  if (esquerda < direita) {
    int meio = esquerda + (direita - esquerda) / 2;

    // Chama o Merge Sort para as duas metades
    mergeSort(array, esquerda, meio);
    mergeSort(array, meio + 1, direita);

    // Mescla as duas metades ordenadas
    merge(array, esquerda, meio, direita);
  }
}

int main() {
  int array[] = {62, 35, 14, 32, 11};
  int tamanho = sizeof(array)/sizeof(array[0]);

  impressao("Vetor original: ", array, tamanho);

  mergeSort(array, 0, tamanho - 1);

  impressao("Vetor ordenado: ", array, tamanho);

  return 0;
}
```

3. Exercícios

3.1. A partir de dois vetores contendo 10 números cada um, gere um novo vetor (de tamanho 20) como resultado da ordenação dos dois primeiros.

3.2. Informe uma sequência de 6 (seis) números entre 1 (um) e 60 (sessenta) para simular uma aposta padrão (6 (seis) números) da mega sena. Depois disso, realize a impressão de 10 (dez) sequências com números aleatórios para simular a realização de 10 (dez) concursos da mega sena. Todas as sequencias devem ser exibidas em ordem crescente e a quantidade de acertos da aposta inicial em cada uma delas deve ser exibida.

3.3. Elabore um programa que cadastre a nota de 8 (oito) destinos turísticos (cidades) do Brasil. A nota deve ser especificada entre 1 (um) e 100 (cem). Após o cadastro, uma lista em ordem crescente com o nome do destino e sua nota deve ser exibida.

Observação: Não utilize struct.

3.4. Qual algoritmo é melhor para ordenação de valores: Merge Sort ou Quick Sort? Justifique sua resposta (informe a(s) referência(s) consultada(s)).