QUICKSORT:

```
#include <stdio.h>

void imprime(int v[7]) {
    int i;
    for (i=0; i<7; i++) {
        printf("%d ", v[i]);
    }
        printf("\n");
}

void quicksort(int *v, int a, int b) {
      int i, j, x, pivo, y;
      pivo = (a + b) / 2;
      i = a;
      j = b;
      x = v[pivo];
      do
      {
            while(v[i] < x && i < b) {
                  i++;
            }
            while(x < v[j] && j > a) {
                  j--;
            }
            if(i <= j)
            {
                  y = v[i];
                  v[i] = v[j];
                  v[j] = y;
                  i++;
                  j--;
            }
      } while(i <= j);
      if(a < j) {
            quicksort(v, a, j);
      }
      if(i < b) {
            quicksort(v, i, b);
      }
}


int main() {
        int i, Vetor[7]= {7,2,12,9,6,1,0};
        printf("elementos fora de ordem: ");
        imprime(Vetor);

        quicksort(Vetor, 0, 6); /* na primeira chamada, os parametros iniciais
sao os extremos do vetor */

        printf("elementos ordenados: ");
        imprime(Vetor);
        return 0;
}
```

MERGESORT:

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

void intercala(int v[11], int inicio, int meio, int fim)
{
    int i, j, k, aux[11];
    i = inicio;
    j = meio+1;
    k = 0;
    while(i <= meio && j <= fim)
    {
        if(v[i] <= v[j])
        {
            aux[k] = v[i];
            i++;
        }
        else
        {
            aux[k] = v[j];
            j++;
        }
        k++;
    }

    while(i <= meio)
    {
        aux[k] = v[i];
        i++;
        k++;
    }

    while(j<=fim)
    {
        aux[k] = v[j];
        j++;
        k++;
    }

    for(i = 0; i< (fim - inicio)+1; i++) //copia valores do vetor auxiliar para
o original
    {
        v[inicio + i] = aux[i];
    }
}

void mergesort(int v[11],int inicio,int fim) // para dividir os elementos em
subvetores
{
    int meio;
    if (inicio < fim)
    {
        meio = (inicio + fim)/2 ;
        mergesort(v,inicio,meio);
        mergesort(v,meio+1,fim);
        intercala(v, inicio, meio, fim);
    }
}

void imprime (int v[11])
{
    int i;
```

```
    for(i = 0; i < 11; i++)
    {
        printf(" %d ", v[i]);
    }
    printf("\n");
}

int main()
{
    int i;
    int aMerge[11] = {20, 31, 14, 8 , 92, 17, 11, 1, 0, 5, 4};
    printf("Nao ordenado: \t");
    imprime(aMerge);
    mergesort(aMerge, 0, 10);
    printf("Ordenado: \t");
    imprime(aMerge);
    printf("\n");
    return 0;
}
```

1. Verificar se o código possui algum erro? Considere como erros problemas de lógica que levem a instabilidade, ou seja, não ordenar corretamente sempre que for executado;
não aparenta ter erros que possam ocasionar problemas de ordenação, mas nos laços das linhas 19 e 22 do quicksort tanto i como j são comparados com o fim e início do vetor respectivamente fazendo com que se verifique todos os elementos, poderia comparar i e j com o pivô e não com os extremos

2. Realizar um teste de mesa detalhado em cada um dos algoritmos a fim de entender seu funcionamento e lógica. Depois, responder às seguintes questões: 2.1. quickSort.c:

2.1.1. Qual o objetivo dos laços das linhas 19 a 24?
o laço da linha 19 procura elementos a esquerda do pivô que sejam maiores ou iguais ao pivô e o laço da linha 22 procura elementos a direita ou após o pivô que sejam maiores ou iguais a ele

2.1.2. Na linha 33, por que foi utilizada essa condição de parada?
Porque quando i for maior que j quer dizer que já foi verificado todos os elementos a esquerda e direita do pivô e já foram feitas todas as trocas necessárias

2.1.3. Explicar os testes condicionais das linhas 34 e 37;
na linha 34 verifica se o subconjunto entre as posições de inicio e posição [j] já foi ordenado e na linha 37 se o subconjunto dos elementos entre a posição i e o final do vetor já foi ordenado

2.1.4. Nas linhas 35 e 38, por que as chamadas recursivas possuem esses argumentos como parâmetros?
Depois de verificar o subvetor na linha 34, caso ainda não tenha sido ordenado chama a função recursiva passando os extremos desse subvetor e na linha 37 se o subvetor ainda não foi ordenado chama a função passando também os valores de inicio e fim do subvetor

2.2. mergeSort.c:

2.2.1. Explique a necessidade das chamadas de funções das linhas 51, 52 e 53.
dividir o vetor em partes menores e ordenar essas partes, depois adicionar no vetor original essa parte já ordenada

2.2.2. Qual seria o impacto da modificação da ordem dessas três linhas? Exemplo: invocar a função intercala() entre as duas chamadas recursivas a mergesort();
invocar intercala() entre as duas a ordenação não funcionou corretamente, não ficou ordenado, invertendo a ordem das chamadas de
mergesort() a ordenação funcionou normalmente

2.2.3. Qual o objetivo do laço das linhas 10 a 23? Por que foram usadas as condições de execução em questão?
Adicionar de modo ordenado no vetor auxiliar os elementos do vetor original e também impedir que elementos duplcados sejam adicionados

2.2.4. As condições de execução dos laços das linhas 25 e 32 são semelhantes ao da linha 10. Justifique a utilização dos mesmos
para adicionar no vetor auxiliar algum elemento que ainda não tenha sido adicionado no loop da linha 10 porque ele pode ter encerrado e ainda ter indices pra adicionar no vetor auxiliar