1. Explique como seria possível melhorar o método bubblesort, armazenando não apenas a informação da troca, mas também a posição do vetor onde ocorreu a troca. Implemente essa modificação.
2. Aplique os procedimentos de ordenação (passo a passo)

a) Bubble Sort;
b) Insertion Sort
c) Quick Sort;
d) Heap Sort;

Ao seguinte vetor ={7, 13, 5, 8, 9, 2, 3}

3. Dada a função ordena( ) descrita a seguir:

```
#define MAX 40
struct pessoa
{
   char nome[30];
   int idade;
};
pessoa cad [MAX];
int n_cad;

1   void xsort( )
2   {
3      int i, j;
4      pessoa p;
5      for (i=1;i<=n_cad-1;i++)
6      {
7         p=cadastro[i];
8         j=i-1;
9         while((strcmp(p.nome,cad [j].nome)<0)&&(j>=0)) {
10          cad [j+1]=cad [j];
11          j=j-1;
12       }
13       cadastro[j+1]=p;
14     }
15  }
```

4. Diga qual é o método de ordenação implementado neste algoritmo e descreva o seu funcionamento. Um vetor *v[p..r]* está "arrumado" se existe $j \in 2[p, r]$ tal que $v[p..j-1] \leq v[j] < v[j+1..r]$. Escreva um algoritmo que decida se v[p..r] está arrumado. Em caso afirmativo, o seu algoritmo deve devolver o valor de j.

5. Escreva uma versão recursiva do algoritmo de ordenação por inserção.

6. Altere o algoritmo Insertion Sort reproduzido abaixo para que ele ordene o vetor de maneira decrescente, e imprima o primeiro e o último elementos do vetor ordenado, assim como o número de trocas realizadas.

```
1. void insertion_sort (int *vector, int n){
2.    int key = 0;
3.    int i,j;
4.    for (j = 1; j < n; j++) {
5.      key = vector[j];
6.      i = j - 1;
7.      while (i >= 0 && vector[i] > key) {
8.        vector[i+1] = vector[i];
9.        i--;
10.     }
11.     vector[i+1] = key;
12.   }
13. }
```

7. Observe o algoritmo original do exercício anterior. No laço *while* é realizada uma busca linear da posição na qual o valor da variável *key* deve ser inserido. Responda:
   a) É possível trocar a busca linear realizada pelo *while* (nas linhas 8,9,10,11) pela busca binária?
   Se for possível, reescreva o algoritmo com essa substituição.
   b) Comente se houve alteração na eficiência do algoritmo.

8. Dê um exemplo de um vetor com N elementos que maximiza o número de vezes que o mínimo é atualizado no método de ordenação seleção (selectionsort).

9. Mostre um exemplo de entrada que demonstra que o método de ordenação seleção não é estável.

10. Mostre um exemplo que demonstra que o Shellsort é instável para sequencia h= 1, 2, 4.

11. Qual dos métodos: bolha, inserção e seleção executa menos comparações para um vetor de entrada contendo valores idênticos.

12. Escreva uma versão iterativa do algoritmo Mergesort. Note que será necessário “simular” as chamadas recursivas (armazenadas na memória) por meio de alguma estrutura de dados auxiliar

13. Compare os algoritmos Insertionsort e Mergesort em termos: 1) de eficiência no pior caso, 2) da utilização de memória e 3) estabilidade. Com essas informações, responda: há alguma situação em que a utilização do Insertionsort é mais adequada do que a utilização do Mergesort? Qual seria?

14. Quando utilizamos o Mergesort porque devemos sempre dividi-lo em 2 sub-vetores? Tente implementar uma solução em que a divide-se o vetor em 3 subvetores para depois realizar a intercalação e responda à pergunta.

15. Uma **ordenação por contagem** de um vetor *x* de tamanho *n* é executada da seguinte forma: declare um vetor *count* e defina *count [i]* como o número de elementos menores que *x[i]*. Em seguida, coloque *x[i]* na posição *count [i]* de um vetor de saída (leve em consideração a possibilidade de elementos iguais). Escreva uma função para ordenar um vetor x de tamanho n usando esse método.

16. No caso do quicksort, considere que o pivô é o elemento central e apresente a ordenação do vetor A = {2, 5, 32, 21, 102, 1, 11, 24, 35, 44, 56}. Informe a quantidade de comparações e trocas efetuadas.