Nome: Mateus Silva Araújo

1) A função fornecida refere-se a uma função de busca sequencial, na qual o vetor precisa estar ordenado para que funcione corretamente.

A função retorna o índice do número procurado no vetor, caso o encontre, caso não o encontre, ela retorna -1.

No caso dessa função, vemos que ela também retorna -1 se o número no vetor for maior que o número procurado, por isso o vetor precisa estar ordenado, isso é bom pois acelera a execução do programa em caso de o número procurado ser pequeno.

Abaixo segue um exemplo de código para correto funcionamento da função:

```
function buscaSequencial(vetor, numeroProcurado){
    for(let i=0; i<vetor.length; i++){
        if(vetor[i]===numeroProcurado){
            return i
        }
        else if(vetor[i]>numeroProcurado){
            return -1
        }
    }
    return -1
}

let vetorOrdenado = [1,2,3,4,5]
let numeroProcurado = 3
let resultado;
resultado = buscaSequencial(vetorOrdenado, numeroProcurado)
console.log(resultado + " - Indice do Numero procurado no vetor")
```

a saída esperada para o programa é:

“2 – Índice do Número procurado no vetor”

2) A função fornecida representa o algoritmo do bubble sort.

Na função temos os parâmetros: a, b, c, d e x;

a → Representa a função a ser chamada;

b → Iteração para percorrer todo vetor

c → Iteração para comparar 2 elementos e colocar o maior elemento no final do vetor;

d → Variável auxiliar para realizar a troca dos 2 elementos

x → Vetor a ser ordenado;

A função irá ordenar o vetor, comparando dois elementos e trocando-os caso o elemento à esquerda seja maior, de modo a colocar o maior elemento do vetor no final.

3) a)A escolha do pivô é arbitrária, de modo que qualquer elemento do vetor pode ser escolhido, porém, ela influencia em seu desempenho, uma vez que caso o maior ou o menor elemento do vetor sejam escolhidos como pivô, teremos apenas 1 troca na primeira iteração e já vai haver a chamada da recursão, diminuindo a eficiência do código. Assim, o pivô ideal escolhido seria o número que representa a mediana do vetor, assim a recursão dividiria o vetor exatamente no meio.

b) vetor = [22, 74, 55, 31, 10, 63, 87]

escolha do pivô → (posição início+ posição final)/2 → arredondamento para baixo;

Nesse caso o elemento escolhido estaria na posição (0+6)/2 = 3 → posição 3

22 74 55 31 10 63 87 → pivô

22 74 55 31 10 63 87

I j

22 74 55 31 10 63 87 → i para;

I j

22 74 55 31 10 63 87 → j para, realiza a troca;

I j

22 10 55 31 74 63 87 → i e j param e realiza a troca;

I j

22 10 31 55 74 63 87 → chamada da recursão dividindo em 2 vetores;

Vetor da esquerda executa:

Pivô → (0 + 1)/2 → 0.5 → arredondando para baixo → pivô na posição 0;

22 10 → pivô

22 10 → i e j param, realiza a troca;

I j

10 22 → chamada da recursão à esquerda e a direita do pivô;

Vetor de 1 elemento, na chamada não acontecerá nenhuma troca;

Vetor da direita executa:

Pivô → (3 + 6)/2 → 9/2 → 4.5 → 4

55 74 63 87 → pivô

55 74 63 87 → i para no pivo e j para no elemento menor que o pivo à esquerda, troca;

I j

55 63 74 87 → chamada da recursão;

Executa vetor da esquerda:

Pivo → (3+4)/2 → 7/2 → 3.5 –> 3

55 63 → i e J param no pivo e não a trocas, chamada da recursão da direita, como a 1 so elemento não acontece nada;

Chamada da recursão da direita do vetor de 4 elementos, como só há 1 elemento, o código executa e não faz nada.

Vetor final ordenado:

Vetor = [10, 22, 55, 63, 74, 87]

4) A função basicamente calcula a exponencial de 2 para um número n passado como parâmetro, sendo uma função recursiva porque faz a chamada de si mesma, no formato:

```
Function x(n){]
        If(n == 0){
                Return -1
        }
        Return x(n-1)
}
```

Nesse caso, há uma condição de parada a ser alcançada quando n chegar a 0.

No caso da função fornecida, a condição de parada esta quando n == 1 e quando o n == 0;

b) fazendo a chamada da função temos:

func(4) = func(3) + 2 * func(2) → executando func(3) temos:

func(3) = func(2) + 2 * func(1)

func(2) = func(1) + 2 * func(0) → func(2) ao chamar func(3);

func(1) = 2

func(0) = 1

func(1) = 2 → que aparece ao chamar func(3)

func(2) = func(1) + 2 * func(0) → func(2) ao chamar func(4):

func(1) = 2

func(0) = 1

totalizando 9 chamadas e o resultado final será 16 = 2^4;

5) para ordenar o vetor: [39 21 54 16 32 18] poderíamos escolher basicamente qualquer método, uma vez que se trata de um vetor pequeno então seu tempo de execução não será demorado.

Dito isso, eu escolheria o selection sort, por ser um método fácil de entender e implementar.

Esse método consiste em achar o menor elemento do vetor e trocar com a primeira posição do vetor percorrido, do seguinte modo:

39 21 54 16 32 18 → encontra o menor e troca com a primeira posição

16 21 54 39 32 18 → a partir daqui, a primeira posição passa a ser o índice 1, porque há a realização de um loop para passar por todos os elementos do vetor;

16 21 54 39 32 18 → encontra o menor e troca com a posição do primeiro loop;

16 18 54 39 32 21 → Segue assim sucessivamente até percorrer todo o vetor;

16 18 21 39 32 54

16 18 21 32 39 54 → Aqui o 39 trocaria de posição consigo mesmo, uma vez que a função executaria do mesmo jeito.

16 18 21 32 39 54 → Fim do primeiro loop e fim da ordenação do vetor!