Nome: Mateus dos Santos Ribeiro, Nº USP: 11796997

Questão 1

Para obter a forma fechada eu transformei a **função** em **novaFuncao**

```
49
50 unsigned char* funcao(unsigned short* v1, unsigned short* v2, int n) {
51     unsigned char* found = calloc(n, sizeof(unsigned char));
52     for (int i = 0; i < n; i++) {
53         unsigned short value = v1[i];
54         for (int j = 0; j < n; j++) {
55             if (value == v2[j]) found[i] = 1;
56         }
57     }
58     return found;
59 }
60
61 unsigned char* novaFuncao(unsigned short* v1, unsigned short* v2, int n, int i, unsigned char *found){
62     if (i==n) return found; // c
63     unsigned short value = v1[i]; // a
64     for (int j = 0; j < n; j++)
65     {
66         if (value == v2[j]) found[i] = 1; // c + 2a
67     }
68     return novaFuncao(v1, v2, n, i+1, found);
69 }
70
```

Há, ao todo, 3 acessos à vetores e 2 comparações, e como é uma função recursiva teremos:
f(n) = 3a + 2c + f(n-1)

Essa recursão irá se repetir até que encontremos o caso base, que é i == n teremos então algo como:
f(n) = 3a + 2c + 3a + 2c f(n-2)
f(n) = 3a + 2c + 3a + 2c + 3a + 2c + f(n-3)
.
.
.
f(n) = (3a + 2c).i + f(n-i)

Como o caso base é atingido em n == i, (ou n-i == 0), substituiremos o i por n na equação acima e teremos:
f(n) = (3a + 2c).n + f(n-n)
Como f(0) é 1 (Pois haverá apenas uma comparação a ser feita e em seguida a função retornará o vetor found), podemos substituir isso na fórmula, obtendo
f(n) = (3a + 2c).n + f(0)
f(n) = (3a + 2c).n + 1

Questão 3

A função otimizada foi a seguinte:

```
unsigned char* funcaoOtimizada(unsigned short* v1, unsigned short* v2, int n, int i, unsigned char *found){
    if (i==n) return found; // c
    unsigned short chave = v1[i]; // a
    int inicio = 0, fim = n-1, meio;

    while(inicio <= fim){
        meio = (int)((fim + inicio)/2.0);
        if (v2[meio] == chave)//c
        {
            found[i] = 1;//a
            break;
        }else if (v2[meio] > chave)//a + c
        {
            fim = meio-1;
        }else
        {
            inicio = meio+1;
        }
    }
    return funcaoOtimizada(v1, v2, n, i+1, found);
}
```

Dentro do quadrado vermelho existe uma busca binária. Como eu tive dificuldades em analisar a busca binária nesse formato, eu criei uma busca binária recursiva que teria o mesmo comportamento dessa iterativa, para usar a equação de recorrência

```
92
93  int buscaBinariaRecursiva(int *vetor, int chave, int inicio, int fim){
94      if(inicio >= fim) return -1; //c
95
96      int centro = (int)((fim + inicio)/2.0);
97      if (vetor[centro] == chave)// a + c
98          return centro;
99      else if(vetor[centro] < chave) // a + c
100         return buscaBinariaRecursiva(vetor, chave, centro + 1, fim);
101     else
102         return buscaBinariaRecursiva(vetor, chave, inicio, centro -1);
103 }
```

Como podemos ver, existem 3 comparações e 2 acessos à vetores nessa função.

f(n) = 3c + 2a + (f(n/2)+-1)

retirando o +-1 para facilitar a análise e expandindo a equação, chegaremos em:

f(n) = 3c + 2a + 3c + 2a + (f(n/4))

f(n) = 3c + 2a + 3c + 2a + 3c + 2a + (f(n/8))

.

.

.

f(n) = (3c + 2a).k + (f(n/2^k))

O caso base é atingido quando o inicio é maior ou igual ao fim, ou seja, quando n/2^k == 1.
Nesse caso, passando o 2^k para o outro lado multiplicando, temos
n == 2^k ou k == log_2^n
Substituindo na equação encontrada, temos:
f(n) = (3c + 2a). log_2^n **+ 1**

CÓDIGO:

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

unsigned char* funcao(unsigned short* v1, unsigned short* v2, int n);
unsigned char* novaFuncao(unsigned short* v1, unsigned short* v2, int n, int i, unsigned char *found);
unsigned char* funcaoOtimizada(unsigned short* v1, unsigned short* v2, int n, int i, unsigned char *found);

int main(int argc, char const *argv[])
{

	int n;
	scanf("%d", &n);

	unsigned short* v1 = (unsigned short*) malloc(n * sizeof(unsigned short));
	unsigned short* v2 = (unsigned short*) malloc(n * sizeof(unsigned short));

	srand(1);

	v1[0] = rand()%2;
	for (int i = 1; i < n; i++) {
		v1[i] = v1[i-1]+(rand()%3)+1;
	}

	v2[0] = rand()%2;
```

```
	for (int i = 1; i < n; i++) {
		v2[i] = v2[i-1]+(rand()%3)+1;
	}
	unsigned char *found = calloc(n, sizeof(unsigned char));
	found = funcaoOtimizada(v1, v2, n, 0, found);

	for (int i = 0; i < n; i++)
	{
		if (found[i] == 1)
		{
			printf("%04hu ", v1[i]);
		}
	}
	printf("\n");


	free(v1);
	free(v2);

	return 0;
}

unsigned char* funcao(unsigned short* v1, unsigned short* v2, int n) {
	unsigned char* found = calloc(n, sizeof(unsigned char));
	for (int i = 0; i < n; i++) {
		unsigned short value = v1[i];
		for (int j = 0; j < n; j++) {
			if (value == v2[j]) found[i] = 1;
		}
	}
	return found;
}

unsigned char* novaFuncao(unsigned short* v1, unsigned short* v2, int
n, int i, unsigned char *found){
	if (i==n) return found; // c
	unsigned short value = v1[i]; // a
	for (int j = 0; j < n; j++)
	{
		if (value == v2[j]) found[i] = 1; // c + 2a
```

```
        }
        return novaFuncao(v1, v2, n, i+1, found);
}

unsigned char* funcaoOtimizada(unsigned short* v1, unsigned short*
v2, int n, int i, unsigned char *found){
        if (i==n) return found; // c
        unsigned short chave = v1[i]; // a
        int inicio = 0, fim = n-1, meio;

        while(inicio <= fim){
                meio = (int)((fim + inicio)/2.0);
                if (v2[meio] == chave)//c
                {
                        found[i] = 1;//a
                        break;
                }else if (v2[meio] > chave)//a + c
                {
                        fim = meio-1;
                }else
                {
                        inicio = meio+1;
                }
        }
        return funcaoOtimizada(v1, v2, n, i+1, found);
}

int buscaBinariaRecursiva(int *vetor, int chave, int inicio, int fim){
        if(inicio >= fim) return -1; //c

        int centro = (int)((fim + inicio)/2.0);
        if (vetor[centro] == chave)// a + c
                return centro;
        else if(vetor[centro] < chave) // a + c
                return buscaBinariaRecursiva(vetor, chave, centro + 1, fim);
        else
                return buscaBinariaRecursiva(vetor, chave, inicio, centro -1);
}
```