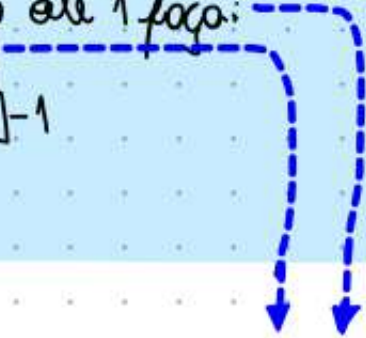


COUNTING_SORT(A, n):

EX 8

```
PARA i ← 1 até k faça:  
    C[i] ← 0  
PARA j ← 1 até n faça:  
    C[A[j]] ← C[A[j]] + 1  
PARA i ← 2 até k faça:  
    C[i] ← C[i] + C[i-1]  
PARA j ← n decrescendo até 1 faça:  
    B[C[A[j]]] ← A[j]  
    C[A[j]] ← C[A[j]] - 1  
Devolva B
```



Um algoritmo de ordenação é considerado estável se ele preserva a ordem relativa de elementos com chaves iguais. O fato do último algoritmo percorrer o array sempre em ordem decrescente faz com que ele coloque os elementos com a mesma chave no array de saída seguindo a mesma ordem que entraram (sem inversão) e esse loop nunca é incrementado. Isso garante a estabilidade.

Simulação →

SIMULAÇÃO

lista original

$[(3,7), (1,3), (2,5), (4,3), (5,2)]$

aplicando o COUNTING-SORT nas chaves secundárias (b_i):

	0	1	2	3	4	5	6	7
				(3,7)	(1,3)	(2,5)	(4,3)	(5,2)
0	0	0	1	2	0	1	0	1
0	0	0	1	3	3	4	4	5
		(5,2)	(1,3)	(4,3)	(2,5)	(3,7)		

max = 7

~ calcula frequência

~ soma cumulativa

Podemos observar que, ao ordenar por b_i , as chaves a_i dos elementos $(1,3)$ e $(4,3)$ (b_i são iguais) continuam mantendo a ordem de entrada.

EX 14

MULT_COMPLEXA(a, b, c, d):

$$ac = a * c$$

$$bd = b * d$$

$$real = ac - bd$$

$$img = (a + b) * (c + d) - ac - bd \quad \# \quad ad - bc$$

DEVOLVA real, img

```

fun  o calcula_silhueta(edif_list)
    silhueta = []
    pr ximo = 1
    enquanto pr ximo <= comprimento(edif_list) fa a
        silhueta = unirSilhuetas(silhueta,
            silhuetaDoEdif cio(edif_list[pr ximo -
            1]))
        pr ximo = pr ximo + 1
    retorne silhueta

fun  o unirSilhuetas(s1, s2)
    resultado = []
    h1, h2 = 0, 0
    i, j = 0, 0

    para _ de 1 at  comprimento(s1) + comprimento(s2)
        fa a
            se i < comprimento(s1) e (j >= comprimento(s2)
                ou s1[i].x < s2[j].x) ent o
                x = s1[i].x
                h1 = s1[i].h
                h = m ximo(h1, h2)
                i = i + 1
            sen o se j < comprimento(s2) ent o
                x = s2[j].x
                h2 = s2[j].h
                h = m ximo(h1, h2)
                j = j + 1

            se n o resultado ou h      resultado[-1].h
                ent o
                    adicioneElementoNaSilhueta(resultado,
                        ElemSilhueta(x, h))

    retorne resultado

```

análise assintótica:

$O(1) + O(1) + O(1) + O(n) + O(m) + O(1) + O(1) = O(n + m)$ onde m é o tamanho da lista *silhueta* e n da lista *edif_{list}*
 portanto $O(n)$ é a complexidade do algoritmo