Problema 01) Counting Sort

Escreva um programa em C que ordena números inteiros, usando o algoritmo *counting sort*. Conforme trabalhado em sala de aula, considere os vetores A, B e C explorados no algoritmo.

Entrada: A primeira linha contém a quantidade **n** (n>0) de números a ordenar, seguido de um espaço em braco e do número inteiro **k** (k>0). A segunda linha possui os números a ordenar, separados por um espaço em branco entre si, tal que cada número $\mathbf{x_i}$ é $0 <= x_i <= k$ ($1 <= i <= n$).

Saída: A primeira linha possui a *string* "B : ", seguida dos números ordenados (vetor B), separados por um espaço em branco entre si. A segunda linha inicia com a *string* "C : ", seguida do conteúdo do vetor C ao final da execução do algoritmo, com valores separados por espaço em branco entre si.

Entrada	Saída
`10 5` `2 5 3 0 2 3 0 5 3 0`	`B : 0 0 0 2 2 3 3 3 5 5` `C : 0 3 3 5 8 8`

Problema 02) Ordenação

Escreva um programa em C que ordena números inteiros usando os algoritmos ordenação por seleção, por inserção e por bolha. A ordenação deve ser estável.

Entrada: A primeira linha contém **n** (n>0) a quantidade de números a ordenar. A segunda linha possui os números a ordenar, separados por um espaço em branco entre si.

Saída: A três primeiras linhas apresentam: uma *string* com a identificação do algoritmo ("selection sort : ", "insertion sort : " ou "bubble sort : "), seguido do número de comparações e do número de trocas, realizadas pelo algoritmo, separados por um espaço. A quarta linha apresenta os números ordenados.

Entrada	Saída

`6`
`4 2 4 1 7 2`
	`selection sort : 15 3`
	`insertion sort : 10 7`
	`bubble sort : 14 7`
	`1 2 2 4 4 7`

Problema 03) Contagem de dígitos

Considere dois números inteiros positivos entre **x** e **y** na base decimal e sem zeros à esquerda. Deseja-se saber quantas vezes cada um dos dígitos (0 a 9) é usado no intervalo fechado definido entre **x** e **y**.

Entrada: Uma única linha com os valores **x** e **y** ($0 < x <= y$) separados por um espaço em branco.

Saída: Uma linha para cada dígito, segundo o formato "d : n", em que **d** refere-se ao dígito e **n** refere-se ao número de ocorrências. As linhas devem ser ordenadas pelo número de ocorrências; em cado de empate, considerar o dígito em si.

Entrada	Saída

`15 22`	`3 : 0`
	`4 : 0`
	`0 : 1`
	`5 : 1`
	`6 : 1`
	`7 : 1`
	`8 : 1`
	`9 : 1`
	`2 : 4`
	`1 : 6`

Problema 04) Ordenação de strings

Escreva um programa que ordena *strings* composta por números e letras. Na ordem lexicográfica, considere que letras maiúsculas antecedem as minúsculas.

Entrada: A primeira linha contém um inteiro **n** (0 < n < 100) que corresponde ao número de *strings*. As demais linhas possuem as *strings* em si, cada qual limitada a 100 caracteres.

Saída: As *strings* ordenadas de forma ascendente, cada uma em linha distinta.

Entrada	Saída
6 **jose de souza** **jose da silva** **maria madalena** **maria aparecida** **joana silva** **Joana silva**	**Joana silva** **joana silva** **jose da silva** **jose de souza** **maria aparecida** **maria madalena**

Problema 05) Ordenação de strings 2

O programa anterior foi modificado, pela inclusão de duas strings adicionais, o que torna a entrada com três *strings*: *string1*, *string2* e *string3*.

Entrada: A primeira linha contém um inteiro **n** (0 < n < 100) que corresponde ao número de entradas. As demais linhas possuem as *strings* em si, tal que cada entrada possui três strings em linhas separadas. Cada strings é limitada a 100 caracteres.

Saída: As **n** entradas ordenadas de forma ascendente, em que *string1*, *string2* e *string3* estão concatenadas em uma mesma linha, com uma vírgula e um espaço em branco entre si. Se houver empate na ordenação em *string1*, usa-se *string2*. Se ainda houver empate, usa-se *string3*.

Entrada	Saída
4 **jose de souza** **rua dos tamoios 11** **1999/12/11** **jose da silva** **rua da praia 22** **2005/09/09** **jose de souza** **rua dos amoras 22** **2000/10/10** **jose de souza** **rua dos tamoios 11** **1999/12/10**	**jose da silva, rua da praia 22, 2005/09/09** **jose de souza, rua dos amoras 22, 2000/10/10** **jose de souza, rua dos tamoios 11, 1999/12/10** **jose de souza, rua dos tamoios 11, 1999/12/11**

Problema 06) Quicksort

Seja a versão recursiva do algoritmo quicksort ao lado.

```
QUICKSORT(A, p, r)
1  if p < r
2      q = PARTITION(A, p, r)
3      QUICKSORT(A, p, q − 1)
4      QUICKSORT(A, q + 1, r)
```

Considere o último elemento do vetor, ou seja a posição **r**, é escolhido como pivô a cada execução do particionamento.

Entrada: A primeira linha contém **n** (n>0) a quantidade de números a ordenar. A segunda linha possui os números a ordenar, separados por um espaço em branco entre si.

Saída: A cada particionamento ocorrido, é impressa uma linha no formato: "v(**pivô**)=**valor** * **vetor**", em que **pivô** refere-se a posição do pivô após o particionamento, **valor** é o valor do pivô, e **vetor** é

o conteúdo do vetor (elementos separados por um espaço em branco). Ao apresentar o vetor, usar os caracteres '[' e ']' para designar a partição em análise. Ao final imprimir uma linha com o vetor ordenado.

Entrada	Saída
10 3 5 8 9 4 5 7 1 3 6	v(7)=6 * [3 5 4 5 1 3 6 8 9 7] v(3)=3 * [3 1 3 5 5 4] 6 8 9 7 v(1)=1 * [1 3] 3 5 5 4 6 8 9 7 v(4)=4 * 1 3 3 [4 5 5] 6 8 9 7 v(6)=5 * 1 3 3 4 [5 5] 6 8 9 7 v(8)=7 * 1 3 3 4 5 5 6 [7 9 8] v(9)=8 * 1 3 3 4 5 5 6 7 [8 9] 1 3 3 4 5 5 6 7 8 9