Técnicas de Pesquisa

1) Existem várias aplicações onde nem todas as entradas (posição do registro ou dado) possuem a mesma probabilidade de serem solicitadas. Em certas situações, um registro que foi ultimamente acessado pode ter maior probabilidade de um novo acesso do que os demais. Sendo assim, seria útil ter um algoritmo que reorganizasse continuamente a tabela de modo que os registros mais acessados fossem deslocados para o início, enquanto os acessados com menos frequência fossem deslocados para o final. No método de **busca por transposição**, cada chave recuperada com sucesso implica na sua movimentação para a posição imediatamente anterior. Assim, os elementos mais acessados serão deslocados naturalmente para o início do *array*, facilitando, dessa forma, sua localização em poucos passos. De acordo com o comportamento citado, implemente o método de **busca por transposição**, obedecendo ao seguinte protótipo de função:

```
def buscaTransposição(chave:int, array:List[int])->int:
```

A função deve retornar o índice do array em que a chave foi encontrada. Se a chave não for encontrada, lançar uma exceção **KeyError**.

2) Codifique uma função que, dada uma chave de busca e um array de entrada, retorne uma tupla contendo todas as posições no array em que foram encontradas a ocorrência da chave. No caso de não haver duplicatas, a função dever retornar None ou a tupla vazia.

```
def getIndiceDuplicatas(chave:int, array:List[int])->tuple:
```

Atenção: Esta função pode receber tanto arrays ordenados quanto desordenados. Considere que não pode ser aplicado, internamente, nenhum método de ordenação. Em sua solução, procure idealizar aquela que maximize o desempenho da função

3) Em uma agência bancária, as filas de atendimento são ordenadas da esquerda para a direita, e o gerente dessa agência percebeu a presença equivocada de um idoso, com a senha 52, na fila de atendimento não preferencial. Visando a sanar o equívoco, o gerente resolveu que, na primeira oportunidade, faria uma busca no sistema para saber se a senha 52 ainda estava ativa, indicando a presença do idoso na fila de atendimento não preferencial. Em caso de resposta positiva, procuraria o cliente para trocar sua senha por outra de atendimento preferencial; se não, apenas registraria o fato para posterior discussão no grupo de qualidade de atendimento.

Considerando o uso de um algoritmo de busca sequencial otimizado, partindo da esquerda para a direita, e as sequências hipotéticas das senhas da fila de atendimento não preferencial e suas regras de ordenação, segundo as quais quem está à esquerda é atendido antes de quem está à direita, o menor número de comparações para o gerente conhecer o resultado de sua busca ocorre em:

	Regra de ordenação	Sequência das senhas na fila	Atendimento
a)	Sequência ordenada crescente	23; 45; 81; 97; 112; 138; 154	Não preferencial
b)	Sequência ordenada crescente	13; 25; 37; 44; 52; 78; 83; 91	Não preferencial
c)	Sequência ordenada crescente	17; 28; 32; 49; 67; 85; 94; 103	Não preferencial
d)	Sequência desordenada	27; 95; 148; 117; 33; 59; 52	Não preferencial
e)	Sequência desordenada	32; 48; 12; 55; 93; 27; 66	Não preferencial

Justifique seu raciocínio.