1 Exercícios de busca sequencial e busca binária

1. **Busca sequencial**: A forma mais fácil de procurar por um elemento é percorrer toda a sequência, comparando o elemento com cada um de seus itens. Este algoritmo é conhecido como busca sequencial (ou busca linear, pois sua complexidade é $O(n)$). Crie uma função que receba uma lista l contendo números e um número x, e retorne verdadeiro caso x ocorra em l e falso caso contrário.

2. **Busca sequencial v.2**: Considere agora uma lista na qual cada elemento é um tupla, sendo que o terceiro elemento da tupla é sempre um inteiro. Crie uma função busca sequencial que verifique se um número x ocorre na terceira posição de algum elemento da lista l.

3. **Busca binária**: Implemente a busca binária, que recebe uma lista l com números inteiros em ordem *decrescente* e verifique se x pertende ou não a l.

4. **Busca binária v.2**: Implemente a busca binária, que recebe uma lista l com números inteiros em ordem *crescente* e retorne a posição em que x se encontra em l (ou -1, caso a lista não contenha x).

5. **Busca sequencial e binária**: Certifique-se de que você consegue implementar sozinho:

   - A busca sequencia de um elemento x numa lista l.
   - A busca binária de um elemento x numa lista l.
   - A busca sequencia de um elemento x numa lista de tuplas, sendo que o elemento estaria na segunda posição de uma das tuplas (caso exista).
   - A busca binária de um elemento x numa lista de tuplas, sendo que o elemento estaria na segunda posição de uma das tuplas (caso exista).

2 Exercícios de ordenação

1. **Bubble sort**: Implemente a versão do método bubble sort, para ordenar os elementos de uma lista em ordem *crescente*, na qual o algoritmo encerra quando não há mais trocas a serem realizadas.

2. **Selection sort**: Implemente o método selection sort para ordenar os elementos de uma lista em ordem *crescente*.

3. **Insertion sort**: Implemente o método insertion sort para ordenar os elementos de uma lista em ordem *decrescente*.

4. **Merge sort**: Crie uma lista de tuplas, sendo que cada tupla armazena um ponto do plano cartesiano (abscissa e ordenada). Popule a lista com 2000 pontos gerados aleatoriamente. Em seguida, implemente o merge sort para colocar os elementos da lista em ordem crescente pela ordenada. Por fim, imprima a abscissa do último elemento.

5. **Quick sort**: Utilizando a mesma lista de pontos da questão anterior, implemente o quick sort para colocar os elementos da lista em ordem crescente pela distância de cada ponto até a origem do plano cartesiano (ou seja, quanto mais perto da origem $(0, 0)$, mais no início da lista o ponto deve ficar). Imprima a soma das 5 primeiros abscissas.

6. **Ordenação de listas de tuplas**: Adapte todos os cinco métodos de ordenação para que eles ordem uma lista de tuplas de acordo com o elemento que se encontra na segunda posição da tupla.

7. **Busca binária de pontos no plano cartesiano**: Utilizando a mesma lista de pontos, já ordenada em função da distância até a origem, peça que o usuário digite um ponto e utilize busca binária para verificar se algum ponto da lista possui a mesma distância até a origem.