2.1 Considere o algoritmo de busca abaixo:

```
int  buscarNaLista(int ** l, int n, int x)
{
  int i, busca;
  busca = -1; //flag para erro
  for (i=0; i < n; i = i+1)
  {
    if (l[i] == NULL) i = n;
    else if (*(l[i]) == x)
         {
           busca = i;
           i = n;
         }
  }
  return busca;
}
```

Para cada elemento da lista são feitas três comparações:
a) `i<n`
b) `l[i] == NULL`
c) `*(l[i]) == x`

Muitas vezes um pequeno artifício pode contribuir para a melhoria do processo. Modifique o algoritmo acima de tal modo que ele inclua o valor a ser buscado na posição imediatamente posterior à do último elemento previamente inserido na lista. Isso pode significar a alocação de um novo índice na lista. Dessa forma, o algoritmo sempre encontra um índice na tabela com as condições desejadas, evitando o fim de tabela.

2.2 Quando uma lista está ordenada, pode-se tirar proveito desse fato. Se o número procurado não pertence à lista, não há necessidade de percorrê-la até o final. Crie um algoritmo de busca, considerando uma lista ordenada.

2.3 Ainda no caso de listas ordenadas, um algoritmo diverso e bem mais eficiente é a busca binária. A ideia básica do algoritmo é percorrer a tabela como se folheia uma lista telefônica ou um dicionário, abandonando-se as partes onde o nome procurado, com certeza, não será encontrado. Em tabelas, o primeiro nó pesquisado é o que se encontra no meio; se a comparação não é positiva, metade da tabela pode ser abandonada na busca, uma vez que o valor procurado se encontra ou na metade inferior (se for menor), ou na metade superior (se for maior). Esse procedimento, aplicado iterativamente, esgota a tabela. Crie um algoritmo de busca binária, considerando uma lista ordenada.

2.4 Pesquisar pelo funcionamento do algoritmo de ordenação conhecido como método da bolha – ou *bubblesort* – e implementá-lo em uma lista sequencial.