Lista de Exercícios 06
Introdução à Ciência de Computação II (SCC0201/501)

Prof. Moacir P. Ponti Jr.

12 de novembro de 2010

1. Suponha uma lista ordenada contendo n itens e um item x que **não** está presente na lista. O problema consiste em determinar entre qual par de itens na lista estaria o item x, isso é, encontrar $a[i]$ e $a[i+1]$ de tal forma que $a[i] < x < a[i]$ para $1 \leq i \leq n$ ou que $x < a[1]$ ou que $x > a[n]$. Isso é interessante para melhorar a performance do algoritmo *Insertionsort*, com relação ao número de comparações, pois ele busca a posição na qual inserir um elemento a ser ordenado.
   a) Encontre o limite assintótico inferior para essa classe de problemas quanto ao número de comparações.
   b) Apresente a intuição (ou uma prova) na qual se baseia o limite encontrado.
   c) Você conhece algum algoritmo que seja ótimo para resolver esse problema? Qual seria?

2. Modifique o algoritmo *insertion sort* para que este utilize a busca binária para encontrar a posição na qual inserir um novo elemento na lista ordenada, supondo que os dados estejam armazenados em um arranjo.
   - encontre a complexidade de tempo para esse algoritmo modificado quanto ao número de comparações em cada iteração e total.
   - apresente a intuição (ou uma prova) na qual se baseia a complexidade encontrada.
   - a busca interpolada seria melhor nesse caso? justifique.
   - seria possível utilizar esse algoritmo em uma lista encadeada? justifique.

3. Implemente em `C` uma função que, supondo que haja recuperação recorrente de registros, realiza busca sequencial em um arranjo não ordenado de 10.000 elementos e utiliza o método da transposição. Nesse método um registro recuperado com sucesso é trocado com o registro anterior. Faça um experimento com esse método
   - insira 10.000 chaves aleatórias entre 1 e 100.000 no arranjo
   - marque o tempo para realizar a busca sequencial convencional de 50 chaves diferentes (ex. os números de 1 a 51), buscando cada chave 1000 vezes.

- marque o tempo para realizar a busca sequencial com o método da transposição, das mesmas 50 chaves diferentes, buscando cada chave 1000 vezes.

4. Implemente em `C` uma função que realiza busca sequencial indexada. Para isso, utilize um arranjo original com o número de itens $n = 10.000$ e o número de índices $k = 50$. Faça um experimento com esse método:
   - insira 10.000 chaves aleatórias entre 1 e 100.000 no arranjo, e ordene o arranjo utilizando algum método de ordenação (obs: não vale utilizar *bubblesort* e suas variantes).
   - marque o tempo para realizar a busca sequencial convencional no arranjo original, de 50 chaves diferentes, buscando cada chave 1.000 vezes.
   - marque o tempo para realizar a busca indexada, das mesmas 50 chaves, buscando cada chave 1.000 vezes.

5. Implemente em `C` uma função que realiza busca binária e uma função que realiza busca interpolada. Faça um experimento com esse método:
   - insira 100 chaves aleatórias entre 1 e 1.000 no arranjo, e ordene o arranjo utilizando algum método de ordenação (obs: não vale utilizar *bubblesort* e suas variantes).
   - faça a busca binária e por interpolação e imprima quantas iterações foram necessárias para encontrar a chave em cada um dos métodos (o número de iterações é o número de vezes que a posição `meio` é (re)calculada.