UNIVERSIDADE FEDERAL DO CEARÁ
Campus de Quixadá
Prof. Arthur Araruna
QXD0010- Estrutura de Dados

LE3
2019.2

Nome: _____ Matrícula: _____

1 Exercícios Auxiliares

1. Para cada um dos algoritmos de ordenação por comparação vistos, forneça um exemplo de vetor com 5 elementos que gere o pior caso desse algoritmo.

2. Existe a possibilidade de o algoritmo BubbleSort reconhecer que o vetor passado já se tornou ordenado ao longo do processo, de forma a encerrarmos o algoritmo antes do previsto? Se sim, o que devemos alterar no algoritmo para gerar esse comportamento? Se não, por que isso não é possível?

3. Tome o vetor inicial $V = [15, 20, 10, 30, 0, 25, 35, 40]$. Realize a ordenação de V segundo cada um dos algoritmos básicos vistos, a cada linha rescrevendo V segundo o que o algoritmo em questão faz após cada passada pelos elementos do vetor.

   Por exemplo, as três primeiras linhas para o SelectionSort seriam:

   $$[0, 20, 10, 30, 15, 25, 35, 40];$$
   $$[0, 10, 20, 30, 15, 25, 35, 40];$$
   $$[0, 10, 15, 30, 20, 25, 35, 40].$$

4. Observe o vetor V a seguir.

   $$V = [\underset{1}{20}, 25, 35, \underset{4}{55}, 0, 10, 30, \underset{8}{60}].$$

   Perceba que V está dividido em duas porções ordenadas, $V[1..4]$ e $V[5..8]$. Ilustre a execução de um entrelaçamento sobre V, destacando quais pares de elementos serão comparados a cada novo elemento inserido no vetor auxiliar.

5. Ilustre o processo de particionar o vetor a seguir segundo o pivô 15. A cada troca realizada pela versão escolhida, reescreva o vetor após a troca, destacando os elementos que mudaram de posição.

   $$[25, 30, 10, 0, 25, 35, 5, 20, 10, 15].$$

6. **POSCOMP 2015 (adaptada).**
   Analise o Algoritmo 1 na página seguinte, descrito na forma de pseudocódigo:

   Esse programa realiza a ordenação decrescente de um vetor de números inteiros, que implementa o algoritmo de:

   A. ordenação rápida.
   B. ordenação por troca.
   C. ordenação por seleção.
   D. ordenação por inserção.
   E. ordenação por intercalação.

7. Baseado no que vimos com relação a ordenação estável e multicritério, que informação podemos adicionar aos valores de um vetor a ser ordenado de forma a permitir que mesmo um algoritmo não-estável realize uma ordenação estável nesse vetor?

   Dica: Como saber, dados dois elementos de mesmo valor, quem deve vir antes no resultado?

Algoritmo 1:

```
para i ← 1,...,10 faça
    X[i] ←EntradaDoUsuário()
n ← 1
flag ← 1
enquanto n ≤ 10 E flag = 1 faça
    flag ← 0
    para i ← 1,...,9 faça
        se X[i] < X[i+1] então
            flag ← 1
            aux ← X[i]
            X[i] ← Xi+1
            X[i+1] ← aux
    n ← n+1
para i ← 1,...,10 faça
    Imprimir(X[i])
```

8. Observe as propostas abaixo e escolha a melhor forma de encontrar pessoas populares dentro de um conjunto de dados contendo entradas de diferentes pessoas de todo o mundo. Cada entrada do conjunto contém um número natural que especifica numericamente quão popular é essa pessoa (quanto maior, mais popular).

   Você possui duas opções:

   1. repetir p vezes uma busca por todo os dados e selecionar a cada vez o maior valor (descartando os já encontrados antes).
   2. ordenar os dados usando o QuickSort e então selecionar as p primeiras entradas com os maiores valores.

   Você descobriu que o tempo de execução da busca e da ordenação são respectivamente $f_b(n) = 0,1n$ ms e $f_o(n) = 0,1n \lg n$ ms.

   Qual das opções você recomendaria ser usada num caso em que não sabemos de antemão os valores de n e p? E qual você recomendaria se soubesse que seu conjunto de dados consiste de 10^6 entradas e que é necessário selecionar as 100 mais populares? Justifique ambas as respostas.

9. Forneça as informações que se pede sobre cada um dos algoritmos mencionados a seguir:

   - InsertionSort
   - SelectionSort
   - BubbleSort
   - MergeSort
   - QuickSort
   - CountingSort
   - RadixSort

   (a) Entrada de pior caso

   (b) Entrada de melhor caso

   (c) Estável?

   (d) Precisa de espaço extra?

   (e) Ideia por traz da ordenação

10. **POSCOMP 2009 (adaptada).**
    Deseja-se efetuar uma busca para localizar uma certa chave fixa x, em um vetor contendo n elementos. A busca considerada pode ser a linear ou binária. No primeiro caso pode-se considerar que o vetor esteja ordenado ou não. No segundo caso o vetor está, de forma óbvia, ordenado. Assinale a alternativa correta:

    A. A busca binária sempre localiza x, efetuando menos comparações que a busca linear.

    B. A busca linear ordenada sempre localiza x, efetuando menos comparações que a não ordenada.

    C. A busca linear não ordenada sempre localiza x, com menos comparações que a ordenada.

D. A busca binária requer $O(\lg n)$ comparações, no máximo, para localizar x.

E. A busca linear ordenada nunca requer mais do que $\frac{n}{2}$ comparações para localizar x.

11. Dê um exemplo de entrada de pior caso para a BuscaInterpolação, com pelo menos 8 elementos, e forneça a lista em ordem dos índices que o algoritmo inspeciona para a sua entrada.

2 Exercícios de Aplicação

12. Devemos encontrar os 1000 itens mais caros de uma lista de preços não-ordenada contendo 10^7 itens distintos. Três esquemas de solução são propostos:

Esquema A: Repetir 1000 vezes o algoritmo Maior, desconsiderando os valores encontrados nas vezes anteriores.

Esquema B: Realizar 1000 iterações do algoritmo SelectionSort, ordenando do fim para o início.

Esquema C: Converter a lista de preços em um vetor (via cópia), depois ordenar eficientemente e retornar os últimos 1000 elementos.

Qual das soluções acima é preferível sobre as outras? Por quê?

13. Vimos que a busca binária é capaz de encontrar o antecessor de um valor em um vetor ordenado em tempo logarítmico. A ordenação por inserção necessita dessa informação para manter uma porção do vetor em que trabalha sempre ordenada. Discuta se seria útil adicionarmos uma busca binária no algoritmo InsertionSort visto, com relação à complexidade de pior caso deste.

14. Escreva um algoritmo que divide um dado vetor de inteiros entre ímpares e pares, de forma que todos os ímpares venham antes de todos os pares, e retorna o índice onde se localiza o primeiro número par. Se seu algoritmo não possuir pior caso $O(n)$, veja se não é possível adaptar algum dos algoritmos lineares que já vimos para auxiliar no seu objetivo.

15. Implemente cada um dos algoritmos de ordenação que vimos na linguagem de sua preferência.

16. Suponha que em uma dada aplicação temos um vetor V com todos os seus valores sabidamente no conjunto $\{1, 2, \ldots, k\}$, sendo $|V| = n$. No nosso domínio, estamos interessados em saber quantos elementos de V estão no intervalo $[a, b]$, para a e b dados. Essa informação será solicitada diversas vezes, mas o vetor V não será alterado; dessa forma, é útil realizar um preprocessamento para viabilizar uma resposta mais rápida. Proponha um algoritmo de preprocessamento de tempo $O(n + k)$ de forma a permitir que a informação buscada seja fornecida em tempo $O(1)$.

17. O algoritmo BinaryInsertionSort utiliza uma busca binária para determinar a posição apropriada do elemento atual $V[i]$ dentro da porção $V[1..i-1]$ previamente ordenada. Determine a complexidade de pior caso desse algoritmo.

18. Escreva um algoritmo para resolver o problema da *Busca de Intervalo*, onde dados um vetor ordenado V com n elementos distintos e dois valores a e b com $a \leqslant b$, desejamos obter o par de índices de V que define um intervalo de elementos tais que seus valores recaem na faixa de valores $[a, b]$. Seu algoritmo deve ter complexidade $O(\lg n)$.

19. Dado um vetor V, suponha que sabemos que existe um índice k de forma que todos os elementos em $V[1..k]$ sejam menores que um valor x e todos em $V[k+1..n]$ sejam maiores que x, com $n = |V|$, porém não sabemos qual é o valor desse índice. Adapte a idéia da busca binária para, dados V e x, encontrar o valor do índice k em $O(\lg n)$.

20. Imagine que, dado um número b em base binária, podemos acessar cada um de seus bits usando a notação de acesso de vetores (i-ésimo bit seria acessado como $b[i]$). Suponha que sabemos que o número b é formado por uma sequência de 0s seguido por uma sequência de 1s. Adapte a idéia da busca binária para encontrar a posição i em que ocorre o primeiro dos 1 em $O(\lg n)$, supondo que existem n bits em b.

3 Desafios

21. Escreva uma versão iterativa do MergeSort. Essa versão iterativa é mais fácil de ser concebida se você imaginar que vamos realizar vários entrelaçamentos no vetor original, em várias passadas, começando com subvetores de tamanho 1, depois com subvetores de tamanho 2, depois com tamanho 4 e assim sucessivamente. (Perceba que o comportamento é semelhante à volta das recursões da versão recursiva a partir dos casos-base.)

    OBS: Perceba que o último subvetor pode conter menos elementos que os demais em algumas passadas, quando o número de elementos do vetor não for potência de 2.

22. Quais alterações podemos fazer no RadixSort para que ele funcione corretamente quando passarmos a incluir números negativos na entrada?