Centro Universitário Senac
Bacharelado em Ciência da Computação
Análise e projeto de algoritmos

Professor: Leonardo Takuno
{leonardo.takuno@gmail.com}

7 de abril de 2020

1. Prove a corretude e determine a complexidade de tempo do algoritmo de ordenação descrito a seguir, conhecido como BubbleSort

```
bubbleSort(A,n)
para i = 1 até n-1 faça
  para j = 1 até n-i faça
    se A[j] > A[j+1] então
      A[j] <-> A[j+1]   // troca A[j] com A[j+1]
devolva x
```

2. Escreva uma função para inverter a ordem dos elementos de um vetor V[]. Você não pode usar outro vetor como área auxiliar. A função deve ter deve ter complexidade $O(n)$, ou seja, o tamanho do vetor V[].

3. Escreva uma função que recebe um vetor A[] e troca de posição seu maior e seu menor elementos. A função deve ter deve ter complexidade $O(n)$, ou seja, o tamanho do vetor V[].

4. Dado um vetor de n números inteiros, faça uma função para determinar o comprimento de um segmento crescente de comprimento máximo. Exemplos: Na sequência $\{5, 10, 3, 2, 4, 7, 9, 8, 5\}$ o comprimento do segmento crescente máximo é 4 $\{2, 4, 7, 9\}$. Na sequência $\{10, 8, 7, 5, 2\}$ o comprimento de um segmento crescente máximo é 1. A função deve ter deve ter complexidade $O(n)$, ou seja, o tamanho do vetor.

5. Escreva o algoritmo que recebe um vetor A de tamanho n contendo inteiros e encontra o par de elementos distintos a e b do vetor que fazem com que a diferença a-b seja a maior possível. A função deve ter deve ter complexidade $O(n)$, ou seja, o tamanho do vetor V[].

6. Escreva uma função que receba dois vetores (A[] e B[]) já ordenados em ordem crescente e ambos possuem o mesmo tamanho. A sua função imprime a INTERSECÇÃO entre os dois vetores, ou seja, os elementos em comum entre os vetores A[] e B[]. Considere que os vetores não contêm valores duplicados. A função deve ter deve ter complexidade $O(n)$, ou seja, o tamanho do vetor A[] e do vetor B[].

7. Repita o exercício anterior, agora deve ser impresso os elementos que estão em A[] mas não estão em B[]. A função deve ter deve ter complexidade $O(n)$, ou seja, o tamanho dos vetores.

8. Escreva uma função que receba dois vetores (A[] e B[]), com n e m elementos, respectivamente. Os vetores estão ordenados em ordem crescente, a função aloca um vetor C[], exatamente com soma dos tamanhos de A e B, e intercala os elementos de A[] e B[] em C[], de forma que o vetor C[] fique em ordem crescente. A função deve ter deve ter complexidade $O(n + m)$, ou seja, a soma dos tamanho dos vetores.

9. Escreva uma função que recebe um vetor como parâmetro, a sua função seleciona o primeiro elemento de um vetor e rearranja o vetor de forma que todos elementos menores ou iguais ao primeiro elemento fiquem a sua esquerda e os maiores a sua direita. No vetor $\{5, 6, 2, 7, 9, 1, 8, 3, 7\}$ após ser rearranjado teríamos $\{1, 3, 2, 5, 9, 7, 8, 6, 7\}$. A função deve rearranjar o vetor com a complexidade $O(n)$.

10. Escreva um algoritmo que calcula a soma dos prefixos de um vetor em tempo $O(n)$. A soma de prefixos de um vetor V em S pode ser definida por:

$$S[0] = V[0]$$
$$S[i] = V[i] + V[i-1] + V[i-2] + \cdots + V[0]$$

11. Dado um vetor com números pares e ímpares, escreva uma função para colocar todos os números pares à frente no vetor e os ímpares ao final. Você não pode usar outro vetor como área auxiliar. A função deve ter deve ter complexidade $O(n)$, ou seja, o tamanho do vetor V[].