IMD0029 - Estrutura de Dados Básicas 1 – Turma 04 – 2016.2 – Prova 01
Prof. Eiji Adachi M. Barbosa

Nome:_____

Matrícula: _____

ANTES DE COMEÇAR A PROVA, leia atentamente as seguintes instruções:

- Esta é uma prova escrita de caráter individual e sem consultas a pessoas ou material (impresso ou eletrônico).
- A prova vale 10,0 pontos e o valor de cada questão é informado no seu enunciado.
- Ao fim da prova há uma questão valendo 1,0 ponto extra. A questão extra só será levada em consideração se nenhuma outra questão tiver nota igual a 0,0.
- Preze por respostas legíveis, bem organizadas e simples.
- As respostas devem ser fornecidas preferencialmente em caneta. Respostas fornecidas a lápis serão aceitas, mas eventuais questionamentos sobre a correção não serão aceitos.
- Celulares e outros dispositivos eletrônicos devem permanecer desligados durante toda a prova.
- Desvios éticos ou de honestidade levarão à anulação da prova do candidato (nota igual a zero).

Questão 1: (3,0 pontos) O auto-complemento (do inglês *autocomplete*) é uma funcionalidade em que uma aplicação tenta "prever" o restante de uma palavra que o usuário está digitando. Tal funcionalidade aumenta a agilidade de escrita quando consegue prever corretamente uma palavra que o usuário pretende digitar a partir de um número reduzido de caracteres. Por este motivo, esta funcionalidade tornou-se bastante popular em aplicações web e móveis. Nesta questão, você deverá implementar uma função que será reusada no contexto de uma aplicação com a funcionalidade de auto-complemento. A função a ser implementada baseia-se em dois parâmetros: um dicionário de palavras (*Strings*) e um prefixo, que é uma *String* representando os caracteres digitados pelo usuário até o momento. Dado um dicionário de palavras e um prefixo, a sua função deverá retornar um par de inteiros (x, y): o primeiro inteiro – x – indica o índice da primeira palavra no dicionário que começa com o prefixo passado como parâmetro e o segundo inteiro – y – indica o índice da última palavra no dicionário que começa com o prefixo. Caso nenhuma palavra no dicionário comece com um determinado prefixo, a função deverá retornar (-1, -1). Sua função deve respeitar a seguinte assinatura:

```
(int, int) SearchPrefix( String dict[N], String prefix )
```

Para a função acima, a eficiência é essencial. Desta forma, sua solução deverá obrigatoriamente ter complexidade inferior a $\Theta(N)$, em que N é o tamanho do vetor de entrada "dict".

Obs.1: Assuma que o vetor de entrada `dict` tem tamanho igual a `N` e que ele já está ordenado.
Obs.2: Para retornar um par de inteiros pode usar a seguinte sintaxe: `return (x,y);`
Obs.3: Para checar se uma String começa com um determinado prefixo assuma que a seguinte função já existe e pode ser reusada: `bool hasPrefix(String str, String prefix)`. Tal função recebe duas Strings `str` e `prefix`, retornando `true` se `str` começa com `prefix`; `false` caso contrário.
Obs.4: Os operadores de comparação `<`, `<=`, `>`, `>=`, `==`, `!=` podem ser usados nesta questão para comparar Strings de acordo com a ordem lexicográfica.

Questão 2: (3,0 pontos) Os EUA são a maior potência esportiva de toda a história dos jogos olímpicos modernos. Desde a primeira olimpíada moderna, em Atenas – 1896, os EUA já ganharam um total de 2523 medalhas olímpicas, sendo 1022 de ouro, 796 de prata e 705 de bronze. O Museu da História do Esporte Americano contratou você para organizar as réplicas de todas estas medalhas num grande mostruário aberto à visitação. O museu deseja exibir, primeiro, todas as medalhas de ouro, depois todas as medalhas de prata e, por fim, todas as medalhas de bronze. Além disso, as medalhas de ouro devem estar dispostas em ordem crescente de acordo com o ano em que foi recebida. Este mesmo critério deve ser seguido pelas medalhas de prata e de bronze. Diante desta solicitação, você deverá propor um algoritmo de ordenação baseado no *Insertion Sort* (Ordenação por Inserção) para resolver o problema do museu. Para projetar o seu algoritmo, considere que será manipulada a seguinte estrutura, a qual representa uma medalha:

```
Estrutura Medalha
{
      int valor; // valor pode ser OURO=1, PRATA=2, ou BRONZE=3
      int ano;
}
```
Além disso, seu algoritmo de ordenação deverá manipular um vetor de medalhas contendo N medalhas (índice começa em 0) seguindo a seguinte assinatura: `void Intercalar( Medalha v[N])`.

Questão 3: (2,0 pontos) A ordenação é um problema básico da computação, existindo inúmeros algoritmos com diferentes características. Tais características devem ser do conhecimento dos programadores que desejam desenvolver programas eficientes. Nesta questão, considere as seguintes características comuns a algoritmos de ordenação:

I. Implementa a estratégia dividir para conquistar.
II. Sempre divide os dados de entrada em duas partes de mesmo tamanho ou com diferença igual a 1.
III. Tempo de execução no pior caso é de ordem quadrática.
IV. Tempo de execução no melhor caso é de ordem $n*\log_2(n)$.
V. Em termos do tempo de execução, possui mesma complexidade assintótica no pior e melhor caso.

Para esta questão, você deverá associar a cada um dos algoritmos representados na tabela abaixo todas suas características, dentre as listadas acima. Todos os algoritmos abaixo têm ao menos uma das características listadas, podendo possuir mais do que uma.

Algoritmo	Característica
Quick Sort (Ordenação por Partição)[1]	
Merge Sort (Ordenação por Intercalação)	
Bubble Sort (Ordenação por Flutuação)[2]	
Insertion Sort (Ordenação por Inserção)	
Selection Sort (Ordenação por Seleção)	

[1]Para o algoritmo de ordenação por partição (*quicksort*), considere que é usada a estratégia mais simples de seleção do pivô vista em sala de aula.
[2]Para o algoritmo de ordenação por flutuação (*bubblesort*), considere a versão com a pequena otimização vista em sala de aula.

Questão 4: (1,0 ponto) O algoritmo de ordenação por partição, o *quicksort*, foi inventado em 1960 pelo cientista da computação Charles Anthony Richard Hoare. Em sala de aula, vimos que um dos pontos críticos para a garantia da boa eficiência do *quicksort* é a seleção do pivô. Neste contexto, responda:

(A) (0,5 ponto) Em qual(is) caso(s) a seleção do pivô faz com que o *quicksort* caia no seu pior caso?
(B) (0,5 ponto) Cite e explique brevemente uma estratégia para a seleção do pivô que diminui a probabilidade do *quicksort* cair no seu pior caso.

Questão 5: (1,0 ponto) Seja $V = \{a_1, a_2, ..., a_n\}$ um vetor de n números reais distintos. Considere o problema P como sendo o problema de determinar se é possível combinar dois números distintos em S de modo que a soma deles seja igual a 0 (zero). O algoritmo abaixo resolve o problema P:

```
P-Solution( V[n] ){
 FOR i = 0; i < n-1; i++ {
    FOR j = n-1; j>i; j-- {
     IF V[i] + V[j] == 0 {
      Return TRUE
     }
    }
  }
 Return FALSE
}
```

Qual a complexidade do algoritmo acima? Justifique brevemente sua resposta (não é necessário calcular a função de custo total).

Questão extra: (1,0 ponto extra) Projete outra solução para o problema da questão anterior com complexidade inferior à do algoritmo `P-Solution` acima. Assuma que o vetor de entrada `V` tem tamanho igual a `N` e que ele ainda não está ordenado.