UNIVERSIDADE FEDERAL DE SERGIPE
CENTRO DE EXATAS E TECNOLOGIA
DEPARTAMENTO DE COMPUTAÇÃO
PROFA. LEILA MACIEL DE ALMEIDA E SILVA

PRIMEIRA PROVA DE PROJETO E ANÁLISE DE ALGORITMO

INSTRUÇÕES: Esta prova tem **2h de duração e 45 min de tolerância** para o envio pelo **Google Classroom**. Gere um **único arquivo** contendo as respostas textuais de todas as questões. Caso opte por fazer à mão, certifique-se da legibilidade das fotos ou digitalização. Insira seu **nome completo** e **matrícula** no cabeçalho da sua resposta. O arquivo com as soluções deve ser em formato **PDF**. O nome de seu arquivo deve possuir o formato **SeuNomeUltimoSobrenome-P1.pdf**. Por exemplo, para o meu nome seria LeilaSilva-P1.pdf. Provas entregues com atraso, serão descontadas em 2,0 pontos a cada 5 minutos de atraso.

Nome do Aluno:
Matrícula:

1. Dado um vetor *A* de tamanho *n*, constituído de palavras de tamanho *m*, ordenadas em ordem lexicográfica (a ordem dos dicionários), e uma palavra *p* de tamanho maior ou igual a *m*, determine o índice da maior palavra menor ou igual a *p* no vetor, se existir. Caso não exista, retorne *-1*. Duas *strings* podem ser comparadas por igualdade (=) e desigualdade (< , >), em tempo *O(i)*, onde *i* é o tamanho da menor *string*, através de uma função pré-definida, que você pode usar diretamente, `compareString`. Esta função recebe duas *strings* `s` e `t` e devolve `-1` se `s < t`, `0` se `s = t` e `1` se `s > t`, na ordem lexicográfica. Por exemplo, `compareString("casa","caro")` devolve `1`, pois `casa` é lexicograficamente maior que `caro`.
   A complexidade de seu algoritmo deve ser ***O(log n)***. A sua solução deve incluir:
   (a) (1,0) estruturação da solução por indução;
   (b) (1,0) algoritmo recursivo derivado do item (a), em pseudo-linguagem;
   (c) (0,25) estabelecimento da fórmula de recorrência que expressa a complexidade do algoritmo em (b);
   (d) (0,75) cálculo explícito da fórmula de recorrência pelo método de substituição visto nos slides. Observe que como *m* é extremamente menor que *n* no pior caso, ao realizar a conta você pode considerar que *m* funciona como uma constante em relação à *n.*

   Por exemplo, para *A = [“arte”, “cafe”, “caju”, “casa”]* e *p=”cama”* a resposta seria *3*, pois *A[3] = “caju”* e *“caju”* é a maior palavra menor ou igual a *“cama”* no vetor. Para *A = [“arte”, “cafe”, “caju”, “casa”]* e *p=”amor”* a resposta seria *-1*, pois não existe nenhuma palavra menor ou igual *“amor”* no vetor.

2. Considere um vetor de tamanho *n*, em que cada elemento representa uma de três cores possíveis: azul, amarelo ou verde. Os elementos do vetor estão inicialmente em ordem arbitrária. Dê uma representação adequada para as cores e elabore um algoritmo *in loco* (sem espaço adicional) com **complexidade de tempo e de espaço *O(n)*** para que no vetor resultante os elementos de cor azul precedam os elementos de cor amarela e estes precedam os de cor verde. Para a sua solução faça:
   (a) (0,5) descreva a ideia da solução;
   (b) (1,0) escreva o algoritmo que implementa a ideia em pseudo-linguagem;

(c) (0,5) argumente porque sua complexidade atende ao exigido na questão.

Para o vetor [A,A,V,A,V,A] a resposta seria [A,A,A,A,V,V].

3. Um texto é dito **relevante** para uma busca, se o número de vezes em que cada palavra-chave da busca ocorrer for maior ou igual que um valor inteiro *k*. Suponha um texto já processado, representado por um vetor *P* das *n* palavras do texto, em ordem arbitrária. Elabore um algoritmo determinar se o texto, representado pelo vetor *P*, é relevante para uma busca informada. A entrada do algoritmo é o vetor *P* de tamanho *n*, um vetor *CH* de palavras-chave de tamanho *m* e um inteiro *k* que mede o grau de relevância. Suponha que as palavras estão sem acentuação, na língua inglesa. Por exemplo, para
P = [sorting, heapsort, heapsort, sorting, tree, sorting, heapsort, list, tree]
com *CH = [heapsort]; k=3*, obtém-se *Resposta: "Texto relevante"*
mas com *CH = [tree, heapsort]; k=3*, obtém-se *Resposta: "Texto irrelevante"*
Na sua solução as *strings* podem ser comparadas usando a função `compareString` da questão 1. Para responder a questão faça:
   (a) (0,5) explique sua ideia de solução;
   (b) (1,5) elabore o algoritmo em pseudo-linguagem;
   (c) (0,5) calcule detalhadamente a complexidade de tempo e espaço do seu algoritmo, no pior caso. Soluções quadráticas **não serão** aceitas.

4. Seja *P* uma cadeia de caracteres de tamanho *m* consistindo de letras e de no máximo um asterisco (*). O asterisco funciona como um caractere curinga que pode casar com uma sequência arbitrária de texto, inclusive com a sequência vazia. Além disso, o asterisco não ocorre nem no início nem no final do padrão. Por exemplo, se *P=¨refres*cante¨* e *T=¨super refresco com gosto marcante para aplacar o calor causticante¨*, existe um casamento possível iniciando no segundo *r* e terminando no *e* de *marcante* e um outro casamento iniciando no mesmo local que o anterior e terminando no *e* de *causticante*. Suponha ainda que cada caractere do seu texto tem um valor associado que é a posição do caractere no texto. Por exemplo, o valor do *r* de *refresco* em *T* é 7, enquanto que do *c* de *com* é 16. A soma dos pesos da cadeia *refres* no texto é 7+8+9+10+11+12 = 57. Elabore um algoritmo baseado no KMP para determinar o casamento de um padrão do tipo *P* em um texto *T*, de tamanho arbitrário *n*, de tal forma a minimizar a soma dos pesos do pedaço do texto que casou com o padrão, ou identifique que tal casamento não é possível. Para responder a questão faça:
   (a) (0,5) explique sua ideia de solução;
   (b) (1,5) elabore o algoritmo em pseudo-linguagem;
   (c) (0,5) discuta a complexidade do seu algoritmo.