MAC 5711 – Análise de Algoritmos
PRIMEIRO SEMESTRE DE 2004
Terceira Prova – 29 de junho

Nome: _____

Assinatura: _____

1. **(Valor: 2,0 pontos)** Digamos que um *ponto* é um certo tipo de objeto (a natureza exata desse objeto é irrelevante). Suponha que f é uma função que leva triplas (x, y, z) de pontos em números racionais.

   Suponha que $x_1, \ldots, x_n$ é uma seqüência de pontos tais que $f(x_1, x_2, x_i) \geq 0$ para todo $i \geq 3$.

   O algoritmo recebe um vetor x contendo $x_1, \ldots, x_n$ e devolve uma subseqüência $p_1, \ldots, p_t$, armazenada em um vetor p:

```
 1.   Algo (x) {
 2.     int t, i;
 3.     p[1] = x[1];
 4.     p[2] = x[2];
 5.     t = 2;
 6.     para i = 3 até n faça {
 7.       enquanto f(p[t-1],p[t],x[i]) < 0 faça
 8.         t = t - 1;
 9.       p[t+1] = x[i];
10.       t = t + 1;
11.     }
12.     devolva p, t;
13.   }
```

   Mostre que o algoritmo consome tempo $O(n)$ no pior caso (supondo que o cálculo de $f(u, v, w)$ consome tempo $O(1)$). Mostre exatamente onde na prova você usa cada hipótese que aparece no enunciado da questão.

2. **(Valor: 2,0 pontos)** Considere uma pilha implementada em um vetor com as operações usuais de `empilha` e `desempilha`. Para evitar *overflow*, ou seja, para lidar com a situação em que a pilha fica cheia, pode-se implementar uma operação que denominaremos de `realoca`, acionada pela operação `empilha` sempre que a pilha fica cheia. A operação `realoca` aloca um novo vetor de tamanho duas vezes maior que o vetor corrente, copia o conteúdo corrente da pilha para o novo vetor e desaloca o vetor corrente. Suponha que as operações de alocação e desalocação de memória (correspondente ao `malloc` e `free`, digamos) consumam tempo constante, ou seja, $O(1)$. Se a pilha está com s elementos, qual é o tempo de pior caso das operações `empilha` e `desempilha` neste caso? Faça uma análise amortizada e deduza o custo amortizado destas operações, ou seja, considere uma seqüência de n operações `empilha` e `desempilha`, realizadas sobre uma pilha inicialmente vazia, deduza o consumo de tempo das n operações no pior caso e depois deduza o custo amortizado de cada uma delas. (Ou apresente uma análise de créditos, deduzindo diretamente o custo amortizado de cada operação.) Suponha que o vetor alocado inicialmente para a pilha tem apenas uma posição. Sua análise deve ser o mais justa possível em termos assintóticos.

3. Considere a estrutura de dados para representação de uma coleção de conjuntos disjuntos conhecida como *union-find* e as implementações mais eficientes das rotinas `makeset`, `find`, `link` e `union`, que usam florestas enraizadas. (A rotina `union` recebe dois representantes de dois conjuntos diferentes e faz a união destes conjuntos, enquanto que a rotina `link` recebe dois elementos arbitrários e, usando a rotina `union`, faz a união dos dois conjuntos a que pertencem estes elementos, se eles estão em conjuntos distintos da coleção.) Denote por n o número de elementos, e por m o número de operações de uma seqüência de operações dada.

   (a) **(Valor: 1,5 pontos)** Nas duas primeiras linhas da tabela abaixo, preencha a complexidade de pior caso das duas rotinas, `find` e `union`, para as implementações em cada um dos casos abaixo. Nas duas últimas, preencha a complexidade amortizada de cada uma delas.

   A - com compressão de caminhos e heurística dos tamanhos;

   B - com apenas compressão dos caminhos;

   C - com apenas heurística dos tamanhos;

   D - sem nenhuma das duas heurísticas.

	A	B	C	D
`find` - pior caso				
`union` - pior caso				
`find` - custo amortizado				
`union` - custo amortizado				

   (b) **(Valor: 1,5 pontos)** Considere a melhor implementação do algoritmo de Kruskal que você conhece. Excluído o tempo gasto com a ordenação das arestas, qual é o tempo requerido pelas demais operações no algoritmo de Kruskal? Dê uma resposta o mais justa possível e explique de onde vem tal complexidade.

4. (a) **(Valor: 1,0 ponto)** Defina *algoritmo polinomial*. Defina *problema concreto de decisão*. Defina a classe P. Defina a classe NP.

   (b) **(Valor: 1,0 ponto)** Explique informalmente o que significa um problema ser NP-completo. Do ponto de vista **prático**, qual é a relevância de se determinar que um certo problema é NP-completo?

   (c) **(Valor: 1,0 ponto)** Considere que temos m máquinas idênticas e n tarefas, cada uma com uma duração. Um *escalonamento* é uma partição do conjunto $\{1, \ldots, n\}$ em m partes. Dados números $p_1, \ldots, p_n$ representando a duração de cada tarefa, o *makespan* de um escalonamento $M = \{X_1, \ldots, X_m\}$ é o número

$$\max\{\sum_{i \in X_j} p_i : j = 1, \ldots, m\}.$$

   Em palavras, cada máquina ficará ocupada por um tempo, que é a soma das durações das tarefas associadas àquela máquina. O *makespan* é o tempo de ocupação de uma máquina que ficará ocupada por mais tempo no tal escalonamento.

   > O problema ESCALONAMENTO em máquinas idênticas consiste no seguinte: dados m, n, $p_1, \ldots, p_n$ e um número t, existe um escalonamento com *makespan* menor ou igual a t?

   Sabe-se que o problema PARTIÇÃO, descrito abaixo, é NP-completo. Use este fato para mostrar que o problema ESCALONAMENTO é NP-completo.

   > O problema PARTIÇÃO consiste no seguinte: dada uma seqüência de números $a_1, \ldots, a_n$, existe um subconjunto S de $\{1, \ldots, n\}$ tal que $\sum_{i \in S} a_i = \sum_{i \notin S} a_i$?