MAC 338 - Análise de Algoritmos

Departamento de Ciência da Computação
Primeiro semestre de 2003

Simulado 1

1. (a) Seja $f : \mathbb{N} \to \mathbb{N}$. Dê a definição formal de $O(f(n))$.
   (b) Prove (usando a definição de $O(\cdot)$) que $5n + 20 = O(n^2)$.

2. Qual é a complexidade de tempo T(n) do seguinte trecho de programa? Justifique a sua resposta.

```
1.   i=n;
2.   while (i>1) {
3.     for (j=0; j<i; j++)
4.       /* corpo do while, que consome, digamos, tempo O(1) */
5.     i=i/2;
6.   }
```

3. Considere a variante do `mergesort` que ordena no lugar.

```
void mergesort_no_lugar(int v[], int ini, int fim) {
  int meio;
  if (ini<fim) {
    meio = (ini+fim)/2;
    mergesort_no_lugar(v,ini,meio);
    mergesort_no_lugar(v,meio+1,fim);
    intercala_no_lugar(v,ini,meio,fim);
  }
}
```

   É possível implementar o `intercala_no_lugar(v,ini,meio,fim)` de forma que ele execute no pior caso em tempo $\Theta(n^2)$, onde n é o número de elementos no vetor `v` entre `ini` e `fim`, ou seja, $n =$ `fim` $-$ `ini` $+ 1$.

   Seja $T(n)$ a complexidade de tempo no pior caso do algoritmo `mergesort_no_lugar`. Escreva a recorrência obtida do algoritmo para $T(n)$ (seja preciso: se necessário, use $\lceil\ \rceil$ e $\lfloor\ \rfloor$).

   Para simplificar, suponha que $n = 2^k$, para algum inteiro não-negativo k. Para estes valores de n, deduza da recorrência a ordem de grandeza de $T(n)$ (ou seja, deduza que $T(n) = \Theta(f(n))$, para uma função $f(n)$ tão simples quanto possível). Justifique a sua resposta.

4. Suponha que um vetor $v[0..n-1]$ está organizado como um *heap* decrescente (portanto $v[0]$ é um elemento máximo). Considere o seguinte problema:

   > dado um índice i, remover $v[i]$ e transformar o vetor restante em um *heap* $v[0..n-2]$.

   Descreva um algoritmo que resolva o problema em tempo $O(\log n)$. Justifique a sua resposta.

5. Dizemos que um vetor a de comprimento n é *quase ordenado com erros de tamanho* k para $k < n$ se, para quaisquer i e j, se $j - i > k$ então $a[j] \leq a[i]$. Assim o vetor não tem que estar completamente ordenado, mas quaisquer dois elementos no vetor que estejam fora de ordem crescente não podem distar mais do que k posições. Por exemplo, a lista

   $$5,\quad 8,\quad 1,\quad 6,\quad 15,\quad 12,\quad 11,\quad 20,\quad 19,\quad 25,\quad 30,\quad 35,\quad 32$$

   é quase ordenada com erros de tamanho 2. Em particular, $a[3] = 1 < 5 = a[1]$ e $3 - 1 = 2$, mas não há dois elementos fora de ordem distando três ou mais posições um do outro. Justifique suas respostas.

   a) Mostre como o `quicksort` pode ser modificado para produzir uma lista que é quase ordenada com erros de tamanho k. Qual é a complexidade de pior caso para tal versão modificada do `quicksort`? E de melhor caso?

   b) Se o vetor de entrada é quase ordenado com erros de tamanho k, qual é a complexidade do algoritmo de ordenação conhecido como `insertion sort`?

   c) Descreva um algoritmo $O(n)$ que, dado um vetor a de comprimento n e um inteiro positivo k, verifique se a é quase ordenado com erros de tamanho k.