Universidade Federal do Ceará
Departamento de Computação

Construção e Análise de Algoritmos
Lista de exercícios 1

1. Um certo rapaz consegue saltar no máximo k degraus ao subir uma escada, onde $k \geq 1$ é um inteiro fixo. Escreva uma equação de recorrência para o número de modos distintos desse rapaz subir uma escada com n degraus. Note que o rapaz pode dar saltos menores que k também.

2. Prove as seguintes afirmações sobre notação assintótica:

(a) $n^3/100 - 25n^2 - 100n + 7$ é $\Theta(n^3)$.

(b) $n^7/150 - 99n^6 + 88n^4 - 77n^3 + 66n^2 - 55n + 44$ é $\Theta(n^7)$.

(c) $7n \log_{\sqrt{2}} n^2 - 16n + 9$ é $\Theta(n \log n)$.

(d) $n^3 \log_2 n - 11n^2 + 22n - 33$ é $\Theta(n^3 \log n)$.

3. Obtenha uma solução assintótica para as seguintes equações de recorrência:

(a) $T(n) = T(0.9 \cdot n) + 7$.

(b) $T(n) = 3 \cdot T(n/2) + n^2$.

(c) $T(n) = 4 \cdot T(n/2) + n^2$.

(d) $T(n) = 5 \cdot T(n/2) + n^2$.

(e) $T(n) = 2 \cdot T(n-2) + 1$.

(f) $T(n) = T(\sqrt{n}) + \log_2 n$.

(g) $T(n) = 2 \cdot T(\sqrt{n}) + \log_2 n$.

(h) $T(n) = 2 \cdot T(n/5) + 3 \cdot T(n/6) + n$.

4. Considere o algoritmo NumRecursivo(N) abaixo. Determine o valor retornado em notação assintótica.

Algorithm 1 inteiro NumRecursivo(inteiro N)

```
Valor = 0    //(Valor é uma variável local)
if N > 1 then
   Valor = Valor + NumRecursivo(⌊N/5⌋) + NumRecursivo(⌊N/5⌋)
   Valor = Valor + NumRecursivo(⌊N/6⌋) + NumRecursivo(⌊N/6⌋) + NumRecursivo(⌊N/6⌋)
   Valor = Valor + NumRecursivo(⌊N/10⌋) + N
end if
retorne Valor
```

5. Você está tentando escolher entre os três algoritmos A, B e C descritos a seguir. O **Algoritmo A** resolve o problema dividindo a entrada em cinco subproblemas com a metade do tamanho, resolve cada subproblema recursivamente e depois combina-os em tempo linear. O **Algoritmo B** resolve o problema dividindo a entrada em dois subproblemas de tamanho $n-1$ (onde n é o tamanho da entrada), resolve cada um recursivamente e combina-os em tempo constante. O **Algoritmo C** resolve o problema dividindo a entrada em nove subproblemas com um terço do tamanho, resolve cada subproblema recursivamente e depois combina-os em tempo quadrático. Qual o tempo de cada um em notação assintótica e qual você escolheria?

6. Considere vetores que satisfazem a propriedade: o subvetor dos índices ímpares está ordenado crescentemente e o dos índices pares está ordenado decrescentemente. Exemplo: A=[1 50 2 40 3 30 4 20 5 10]. Faça um algoritmo de tempo $O(\log n)$ que receba um vetor desse tipo e um inteiro x, e informe se x está no vetor, retornando sua posição, se for o caso.

7. Sobre o algoritmo HeapSort faça o que se pede. **(a)** Reescreva-o em pseudo-código para receber como entrada um vetor A e índices $p < r$ e ordenar o subvetor $A[p, \ldots, r]$. **(b)** Altere-o para utilizar um heap **mínimo** enraizado na última posição do vetor A, ao invés de um heap máximo enraizado na primeira posição.

8. Elabore um algoritmo $O(n)$ de decomposição de um vetor S em três subvetores. Esse algoritmo recebe como entrada, além do vetor S, um valor piv pertencente a S, e os índices p e r, $1 \leq p \leq r$. O algoritmo deve rearrumar os elementos em $S[p \ldots r]$ e retornar dois índices q_1 e q_2 satisfazendo as seguintes propriedades: **(a)** se $p \leq k \leq q_1$, então $S[k] < piv$; **(b)** se $q_1 < k \leq q_2$, então $S[k] = piv$; **(c)** se $q_2 < k \leq r$, então $S[k] > piv$.

9. Seja $X[1 \ldots n]$ um vetor qualquer (os elementos desse vetor não são necessariamente inteiros ou caracteres; podem ser objetos quaisquer, como frutas ou arquivos executáveis). Suponha que você possui apenas um operador "=" que permite comparar se um objeto é igual a outro. Dizemos que X tem um elemento **majoritário** x se mais da metade de seus elementos são iguais a x. Escreva um algoritmo de tempo $\Theta(n \ log \ n)$ que diz se X possui ou não um elemento majoritário. Caso sim, devolva o seu valor. **Dica:** Se x é majoritário em X, então x é majoritário na primeira ou na segunda metade de X (explique porquê).

10. Sejam $X[1 \ldots n]$ e $Y[1 \ldots n]$ dois vetores ordenados. Escreva um algoritmo $\Theta(\log \ n)$ para encontrar a mediana de todos os $2n$ elementos nos vetores X e Y. Prove esta complexidade.

11. Faça um algoritmo de tempo $\Theta(n \log n)$ para resolver o seguinte problema: dado um vetor com n números inteiros positivos e um outro número inteiro positivo x, determine se existem ou não dois elementos cuja soma é igual a x.

12. Seja $X[1 \ldots n]$ um vetor de inteiros. Dados $i < j$ em $\{1, \ldots, n\}$, dizemos que (i, j) é uma inversão de X se $X[i] > X[j]$. Escreva um algoritmo $\Theta(n \ log \ n)$ que devolva o número de inversões em um vetor X. **Dica:** Tenta fazer essa contagem enquanto ordena o vetor no Merge-Sort.

13. Faça um algoritmo de divisão e conquista em pseudo-código para multiplicar duas matrizes quadradas (ou seja, o número de linhas é igual ao número de colunas), dividindo cada matriz em 9 submatrizes quadradas, assumindo que o número de linhas e colunas é uma potência de 3. Calcule a complexidade de tempo em notação assintótica.

14. Suponha que você tem k vetores ordenados de tamanho n e deseja combiná-los em um único vetor ordenado de tamanho kn. **(a)** Uma ideia é usar o algoritmo INTERCALA, intercalando o primeiro e o segundo, depois intercalando o resultado com o terceiro, depois com o quarto e etc... Qual a complexidade desse procedimento em termos de k e n? **(b)** Mostre uma solução mais eficiente usando divisão e conquista.

15. O algoritmo do k-ésimo mínimo visto em sala de aula ainda seria $\Theta(n)$ se tomássemos grupos de 3 elementos, ao invés de 5? E se fossem grupos de 7 elementos? Justifique pelo método da árvore de recursão.

16. Seja $X[1 \ldots n]$ um vetor de **números reais**. Dizemos que X tem um elemento **popular** x se mais de **um terço** de seus elementos são iguais a x. Escreva um algoritmo de tempo linear $\Theta(n)$ que diz se X possui ou não um elemento popular. Caso sim, devolva o seu valor. **Dica:** Use o algoritmo de Seleção do k-ésimo mínimo de tempo linear no pior caso.

17. Dizemos que um algoritmo é de **quase-ordenação** se, para qualquer vetor $A[1 \ldots n]$, o algoritmo rearranja os valores do vetor A de modo que $i < j$ implica $A[i] < A[j] + 0.1$. Por exemplo, o vetor $A = [1.5 \ 1.45 \ 2.4 \ 2.35 \ 3]$ está quase-ordenado. Sabendo que os valores do vetor de entrada são números reais menores que 100, faça um algoritmo de tempo $O(n)$ para quase-ordenar o vetor.