PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
LISTA DE EXERCÍCIOS 1 – ANÁLISE DE ALGORITMOS - 17/09/2018

1- Resolva os itens abaixo:
   (a) Suponha $b^x = a$, qual o valor de *x* em termos de *a* e *b*?
   (b) Usando o item (a), mostre que $\log_c(ab) = \log_c a + \log_c b$
   (c) Mostre que $a^{\log_b n} = n^{\log_b a}$
2- Arranje a lista de funções abaixo em ordem crescente de taxa de crescimento. Isto é, se a função $g(n)$ segue imediatamente a função $f(n)$ então deve ser válido que $f(n)$ é $O(g(n))$.

$$f_1(n) = 10^n \quad , \quad f_2(n) = n^{1/3} \quad , \quad f_3(n) = n^n \quad \text{e} \quad f_4(n) = \log_2 n$$

3- Utilizando a definição de notação assintótica, apresente limites apertados e o mais próximo possível para as funções abaixo.
   (a) $g(n) = 2n^3 - 9n^2 + 8$
   (b) $g(n) = 3n^3 - 7n + 5$.
4- Resolva as seguintes recorrências: assuma que $T(n)$ é constante para $n \leq 2$. Para cada $T(n)$ abaixo, dê a expressão da fórmula fechada em notação *O* que mais se aproxima de $T(n)$. Isto é, $T(n) = 2T\left(n/2\right) + n$ resolve para $O(n \log n)$. Utilize o método da expansão.
   (a) $T(n) = (T(n-1))^2$
   (b) $T(n) = T(n/2) + bn \log n$
   (c) $T(n) = 2T(n/3) + n^{1/3}$
   (d) $T(n) = \sqrt{n}\, T(\sqrt{n}) + n$
   (e) $T(n) = 4T(n/2) + n^2$
   (f) $\begin{cases} T(a) = \Theta(1), \ a \geq 1 \text{ é uma constante} \\ T(n) = T(n-a) + T(a) + n, \ \text{se} \ n > a \end{cases}$
5- Resolva a recorrência abaixo, utilizando o método da substituição (indução). Lembre-se que é necessário "sugerir" uma possível função. Assuma que $T(n)$ é constante para *n* suficientemente pequeno. $T(n) = T\left(n/2\right) + T\left(n/4\right) + T\left(n/8\right) + n$
   .
6- Sejam as recorrências $S(n) = 2S\left(n/4\right) + 2n^2\sqrt{n}$ e $T(n) = 9T\left(n/2\right) + 3n^2$.
   Utilizando o teorema Máster, diga quais afirmações abaixo são verdadeiras.
   $S(n) = O(T(n))$
   $T(n) = O(S(n))$
   $S(n) = \Omega(T(n))$
   Também utilizando o teorema Máster, resolva a recorrência abaixo, assumindo que n é uma potência de 2 e c é uma constante.

$$\begin{cases} T(1) = 1 \\ T(n) = 4T\left(n/2\right) + n^c, \ \text{se} \ n > 1 \end{cases}$$

7- O elemento majoritário de um vetor de comprimento *n* é o elemento que aparece estritamente mais que $\left\lfloor n/2 \right\rfloor$ vezes no vetor. Observe que se este elemento existe, é único. Um algoritmo para encontrar este elemento com tempo de execução $\Theta(n^2)$ é imediato. Seria possível obter este elemento em tempo $\Theta(n)$? Neste caso não se sugere a técnica divide-and-conquer, mas o uso de estruturas de dados como pilha ou fila.

8- Suponha o algoritmo de busca abaixo, onde *n* é um inteiro positivo:

```
            pesquise(n)
if  n≤1
     then "inspecione o elemento e termine"
     else
          for  i←1  to n
                "inspecione o elemento"
          pesquise( n/3 )
```

(a) Explique o que está sendo feito neste algoritmo, supondo que o tempo de "inspecione o elemento" seja constante.
(b) Como este é um algoritmo recursivo, seu tempo é expresso por equações de recorrência. Apresente as equações.
(c) Resolva a recorrência pelo método da expansão.

9- Aplique a técnica "divide-and-conquer" para resolver o problema de calcular a *n-ésima* potência de *a*, isto é, a^n . Apresente os passos da técnica, o algoritmo, a recorrência e o tempo do algoritmo utilizando algum método para resolver a recorrência. Avalie o tempo de um algoritmo que não utiliza a técnica.

10- Suponha que dado um vetor A de *n* inteiros e um outro inteiro *x*, deseja-se determinar se existe ou não dois elementos (não necessariamente distintos) tais que a soma destes é igual a *x*. Um algoritmo com tempo de execução $\Theta(n^2)$ é imediato. A seguir são apresentadas duas versões para resolver este problema, utilizando ordenação. Pede-se então que você as termine, detalhando as partes vagamente descritas. Você deve tentar obter o melhor tempo possível para cada versão e, além disso, este deve ser melhor que $\Theta(n^2)$. Por isso, em seguida, apresente o tempo de cada versão, justificando.

(a) Ordene A utilizando mergesort. Para cada elemento $A[i]$ em A, faça $y=A[i]-x$ e procure *y* em A.
(b) Ordene A utilizando mergesort. Em seguida, faça *i* ser a primeira posição de A e *j* a última, então verifique se $A[i]+A[j]=x$. Caso a igualdade não seja verdadeira, continue o processo de verificação, por incrementar i ou decrementar j, dependendo do resultado de $A[i]+A[j]$.

11- Dado um vetor de *n* elementos não ordenados, encontre os *k* menores números e como saída apresente estes elementos de forma ordenada. Descreva os algoritmos que implementam os métodos abaixo, com melhor tempo assintótico para o pior caso e apresente os tempos em função de *n* e *k*.

(a) Ordenação dos *n* elementos e (b) Utilizando fila de prioridades.