Análisis y Diseño de Algoritmos (Algorítmica III) Lista de Ejercicios-Recurrencias

I. Método de Sustitución.

1. Demuestre que la solución de la recurrencia $T(n) = T(n/2) + 1$ pertenece a $O(\log n)$.
2. Demuestre que la solución de $T(n) = 2T(n/2) + n$ es $\Theta(n \log n)$.
3. Demuestre que la solución de $T(n) = T(n/2) + T(n/2) + n$ es $\Omega(n \log n)$.
4. Resuelva la recurrencia $T(n) = T(n/3) + T(2n/3) + 1$.

II. Método de Iteración y Árbol de Recurrencia.

1. Determine un buen limete superior asintótico para la recurrencia $T(n) = 3T(n/2) + n$ usando el metodo de iteración.
2. Determine un buen limete superior asintótico para la recurrencia $T(n) = T(n/3) + T(2n/3) + n$ es $\Theta(n \log n)$ utilizando el método del árbol de recur- rencia. No se preocupe por redondeos.
3. Dibuje el árbol de recurrencia para $T(n) = 4T(n/2) + n$ y obtenga la clase Θ a la que pertenece la solución.
4. Use el método de iteración para resolver la recurrencia $T(n) = T(n - a) + T(a) + a$ donde $a \geq 1$ es un entero positivo.
5. Use el metodo de arbol de recurrencia para resolver la recurrencia $T(n) = T(an) + T((1 - a)n + n$ donde $0 < a < 1$ es una constante.

III. Teorema Master.

1. Use el Teorema Master para resolver las recurrencias de abajo.

   (a) $T(n) = 4T(n/2) + n$

   (b) $T(n) = 4T(n/2) + n^2$

   (c) $T(n) = 4T(n/2) + n^3$

2. El tiempo de ejecución de un algoritmo A es descrito por la recurrencia $T(n) = 7T(n/2) + n^2$, otro algoritmo A' tiene complejidad de tiempo de- scrito

por $T'(n) = aT'(n/4) + n^2$. ¿Cuál es el mayor entero a tal que A' es asintóticamente mas rápido que A?

3. ¿El Teorema Master puede ser aplicado a la recurrencia $T(n) = 4T(n/2) + n^2 \log n$? Justifique su respuesta. Obtenga un buen limite superior asintótico para la recurrencia, sin usar el Teorema Master directamente.

4. Determine una expresión cerrada para cada una de las siguitenes recuerren- cias. [P11, Poggi, 18/03/2015, PUC-Rio]

a) T (1) = 1; T (2) = 6; T (n) = T (n – 2) + 3n + 4, ∀n ≥ 3.

b) T (1) = 1; T (2) = 6; T (n) = 2.T (n – 2) + 3, ∀n ≥ 3.

(c) $\sum_{i=1}^{n-1}(T(i) + T(n+i)) + 1, \ \forall n \geq 2.$

MIT, Fall 2005

IV. Problem 1-2. Recurrences

Give asymptotic upper and lower bounds for T(n) in each of the following recurrences. Assume that T(n) is constant for n<= 10. Make your bounds as tight as possible, and justify your answers.

1) $T(n) = 2T(n/3) + n \lg n$
2) $T(n) = 3T(n/5) + \lg^2 n$
3) $T(n) = T(n/2) + 2^n$
4) $T(n) = T(\sqrt{n}) + \Theta(\lg\lg n)$
5) $T(n) = 10T(n/3) + 17n^{1.2}$
6) $T(n) = 7T(n/2) + n^3$
7) $T(n) = T(n/2 + \sqrt{n}) + \sqrt{6046}$
8) $T(n) = T(n-2) + \lg n$
9) $T(n) = T(n/5) + T(4n/5) + \Theta(n)$
10) $T(n) = \sqrt{n}\, T(\sqrt{n}) + 100n$

UFOP, BRASIL

V. Lista de Exercícios sobre Recursividade

1) Dado os algoritmos recursivos abaixo, apresente suas funções de complexidade de tempo.

```
a)
void Pesquisa(int n)
{
   if (n > 1)
   {
      Inspecione n*n*n elementos; // custo n^3 Pesquisa
      (n/3);
```

```
   }
}
```

b)

```
void Pesquisa(n)
{
   if ( n <= 1 )
      return;
   else
   {
      obtenha o maior elemento dentre os n elementos;
      /* de alguma forma isto permite descartar 2/5 dos*/
      /* elementos e fazer uma chamada recursiva no resto*/ Pesquisa(3n/5);
   }
}
```

c)

```
void Pesquisa(int A[n], int n);
{
   if ( n <= 1 )
      return;
   else
   {
      ordena os n elementos;
      /* de alguma forma isto permite descartar 1/3 dos */
      /* elementos e fazer uma chamada recursiva no resto*/ Pesquisa (2n/3);
   }
}
```

d)

```
int fib(int n)
{
   if ( n == 0 )
      return 0;
   else if (n == 1) return
      1;
   else
      return Fib(n-1) + Fib(n-2);
}
```

e)

```
void    Proc( int n )
{
   if ( n == 0)
      return 1;
   else
      return Proc(n-1) + Proc(n-1);
}
```

f)

```
/* n é uma potencia de 2 */
void Sort (int A[n],int i, int j)
{
   if ( i < j )
   {
      m = (i + j - 1)/2;

      Sort(A,i,m);          /* custo = T(N/2) */ Sort(A,m+1,j);
                            /* custo = T(N/2) */ Merge(A,i,m,j); /*
      custo = N-1 comparacoes no */
      /* pior caso */
      /* Merge intercala A[i..m] e A[m+1..j] em A[i..j] */
   }
}
```

g)

```
/* n uma potencia de 3 */
void Sort2 (int A[n], int i, int j)
{
```

```
   if ( i < j )
   {
      m = ( (j - i) + 1 )/3;
      Sort2(A,i,i+m-1);
      Sort2(A,i+m,i+ 2m -1);
      Sort2(A,i+2m,j);
      Merge(A,i,i+m,i+2m, j);
      /* Merge intercala A[i..(i+m-1)], A[(i+m)..(i+2m-1) e
A[i+2m..j] em A[i..j] a um custo ( (5n/3) -2 ) */
   }
}
```

2) Implemente uma função recursiva que, dados dois números inteiros *x* e *n*, calcula o valor de x^n. Escreva e resolva a equação de recorrência dessa função. Qual é a ordem de complexidade da sua função? Qual seria a ordem de complexidade dessa mesma função implementada sem utilizar recursividade? O que você conclui?

3) Considere a função abaixo:

```
int X(int a)
{
  if ( a <= 0 )
    return 0;
  else
    return a + X(a-1);
}
```

a. O que essa função faz?

b. Calcule a sua ordem de complexidade. Mostre como você chegou a esse resultado.

c. Escreva uma função não-recursiva que resolve o mesmo problema. Qual é a ordem de complexidade da sua função? Explique.

d. Qual implementação é mais eficiente? Justifique.

4) Vários algoritmos em computação usam a técnica de “Dividir para Conquistar”: basicamente eles fazem alguma operação sobre todos os dados, e depois dividem o problema em sub-problemas menores, repetindo a operação. Uma equação de recorrência típica para esse tipo de algoritmo é mostrada abaixo. Resolva essa equação de recorrência.

```
T(n) = 2T(n/2) + n;
T(1) = 1;
```

5) Um problema típico em ciência da computação consiste em converter um número da sua forma decimal para a forma binária. Por exemplo, o número 12 tem a sua representação binária igual a 1100. A forma mais simples de fazer isso é dividir o número sucessivamente por 2, onde o resto da *i-ésima* divisão vai ser o dígito *i* do número binário (da direita para a esquerda).

   Por exemplo: 12 / 2 = 6, resto **0** (1º dígito da direita para esquerda), 6 / 2 = 3, resto **0** (2º dígito da direita para esquerda), 3 / 2 = 1 resto **1** (3º dígito da direita para esquerda), 1 / 2 = 0 resto **1** (4º dígito da direita para esquerda). Resultado: **12 = 1100**

   a. Escreva um procedimento recursivo `Dec2Bin(n: integer)` que dado um número decimal imprima a sua representação binária corretamente.

   b. Calcule qual é a ordem de complexidade do seu procedimento. Para isso, **determine e resolva** a equação de recorrência desse procedimento recursivo.

   c. Utilizando um dos tipos abstratos de dados vistos em sala, implemente um procedimento que faça a mesma coisa, ou seja, dado um número decimal imprima a sua representação binária.

6) Considere um sistema numérico que não tenha a operação de adição implementada e que vc disponha somente dos operadores (funções) sucessor e predecessor. Então, pede-se para escrever uma função recursiva que calcule a soma de dois números x e y através desses dois operadores: sucessor e predecessor.

7) O máximo divisor comum (***MDC***) de dois números inteiros ***x*** e ***y*** pode ser calculado usando-se uma definição recursiva:

$$MDC(x, y) = MDC(x - y, y), \text{ se } x > y \text{ .}$$

   Além disso, sabe-se que:

$$MDC(x, y) = MDC(y, x)$$

$$MDC(x, x) = x$$

   Exemplo:

$$MDC(10,6) = MDC(4,6) = MDC(6,4) = MDC(2,4) = MDC(4,2) = MDC(2,2) = 2$$

   Então, pede-se que seja criada uma função recursiva para descrever tal definição. Crie, também, um algoritmo que leia os dois valores inteiros e utilize a função criada para calcular o ***MDC*** de ***x*** e ***y***, e imprima o valor computado.

8) Pode-se calcular o resto da divisão, ***MOD***, de ***x*** por ***y***, dois números inteiros, usando-se a seguinte definição:

$$MOD(x, y) = \begin{cases} MOD(|x| - |y|, |y|), & \text{se } |x| > |y| \\ |x| & \text{se } |x| < |y| \\ 0 & \text{se } |x| = |y| \end{cases}$$

Então, pede-se que seja criada uma função recursiva para descrever tal definição. A função deve retornar -1 caso não seja possível realizar o cálculo. Além disso, crie um algoritmo que leia os dois valores inteiros e utilize a função criada para calcular o resto da divisão de ***x*** por ***y***, e imprima o valor computado.

9) Pode-se calcular o quociente da divisão, ***DIV***, de ***x*** por ***y***, dois números inteiros, usando-se a seguinte definição:

$$DIV(x, y) = \begin{cases} 1 + DIV(|x| - |y|, |y|), & \text{se } |x| > |y| \\ 0 & \text{se } |x| < |y| \\ 1 & \text{se } |x| = |y| \end{cases}$$

Então, pede-se que seja criada uma função recursiva para descrever tal definição. A função deve retornar -1 caso não seja possível realizar o cálculo. Além disso, crie um algoritmo que leia os dois valores inteiros e utilize a função criada para calcular o quociente de ***x*** por ***y***, e imprima o valor computado.

10) Seja a série de Fibonacci:

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

que pode ser definida recursivamente por:

$$Fib(n) = \begin{cases} 1 & \text{se} \quad n = 1 \vee n = 2 \\ Fib(n-1) + Fib(n-2) & \text{se} \quad n > 2 \end{cases}$$

Então escreva:

- Uma função recursiva que gere o termo de ordem n da série de Fibonacci.
- Um algoritmo que, utilizando a função definida acima gere a série de Fibonacci até o termo de ordem 20.

11) O mínimo múltiplo comum (***M.M.C.***) entre dois números inteiros e positivos ***X*** e ***Y*** é definido como sendo o menor inteiro positivo, que seja múltiplo comum a ***X*** e ***Y***. Pede-se que seja criada uma função recursiva (não serão aceitas funções não recursivas) para o cálculo do ***M.M.C.***, onde a função deverá retornar 0 caso não seja possível computar o ***M.M.C.*** e o valor do ***M.M.C.*** entre ***X*** e ***Y*** em caso contrário. Então, apresenta-se a seguinte definição recursiva que deve ser implementada:

$$M.M.C.(X,Y) = \begin{cases} Z * M.M.C.(X/Z, Y/Z), & \text{se } X \bmod Z = 0 \text{ e } Y \bmod Z = 0 \text{ para } 1 < Z \le X, Y \text{ se} \\ Z * M.M.C.(X/Z, Y) & X \bmod Z = 0 \text{ e } Y \bmod Z \ne 0 \text{ para } 1 < Z \le X, Y \text{ se } X \\ Z * M.M.C.(X, Y/Z) & \bmod Z \ne 0 \text{ e } Y \bmod Z = 0 \text{ para } 1 < Z \le X, Y \end{cases}$$

$$M.M.C.(1,1) = 1$$

Escreva também um algoritmo para testar a função criada.

12) Implemente uma função recursiva `soma(n)` que calcula o somatório dos n primeiros números inteiros. Escreva e resolva a equação de recorrência dessa função. Qual é a ordem de complexidade da sua função? Qual seria a ordem de complexidade dessa mesma função implementada sem utilizar recursividade? O que você conclui?

Stanford 2017, EEUU

VI. Lista Stanford-Recuerrencias

1. In your pre-lecture Exercise for Lecture 3, you saw two different proofs that the solution to the recurrence relation $T(n) = 2 \cdot T(n/2) + n$ with $T(1) = 1$, was exactly $T(n) = n(1 + \log(n))$, when n was a power of two.

   (a) What is the exact solution to $T(n) = 2 \cdot T(n/2) + n$ with $T(1) = 2$, when n is a power of 2?

   (b) What is the exact solution to $T(n) = 2 \cdot T(n/2) + 2n$ with $T(1) = 1$, when n is a power of 2?

[We are expecting: Your answer, with a convincing argument (it does not need to be a formal proof). Notice that we want the exact answer, so don't give a $O()$ statement.]

2. Consider the recurrence relation $T(n) = T(n-1) + n$ with $T(1) = 1$. Your friend claims that $T(n) = O(n)$, and offers the following justification:

   Let's use the Master Theorem with $a = 1$, $b = \frac{n}{n-1}$, and $d = 1$. This applies since

$$\frac{n}{b} = n \cdot \left(\frac{n-1}{n}\right) = n - 1.$$

   Then we have $a < b^d$, so the Master Theorem says that $T(n) = O(n^d) = O(n)$.

   What's wrong with your friend's argument, and what is the correct answer?

[**HINT: It is totally fine to apply the Master Theorem when b is a fraction; that's not the problem.]**

[We are expecting: A clear identification of the faulty logic above; your solution to this recurrence (you may use asymptotic notation[1]) and a short but convincing justification.]

3. Use any of the methods we've seen in class so far to solve the following recurrence relations.[2]

   (a) $T(n) = T(n/3) + n^2$, for $n > 3$, and $T(n) = 1$ for $n \leq 3$.

   (b) $T(n) = 2T(n/2) + 10 \cdot n + 4$, for $n > 2$, and $T(n) = 1$ for $n \leq 2$.

   (c) $T(n) = T(n/2) + T(n/4) + n$ for $n > 4$, and $T(n) = 1$ for $n \leq 4$.

[We are expecting: The answer (you may use asymptotic notation) and a justification. You do not need to give a formal proof, but your justification should be convincing to the grader.]

4. Consider the function T (n) defined recursively by

$$T(n) = \begin{cases} 2T(n/2) + \frac{n}{\log(n)} & n > 2 \\ 1 & n \leq 2 \end{cases}$$

Fill in the blank: T (n) = Θ(--------). [HINT: It may be helpful that $\sum_{i=1}^{m} 1/i = \Theta(\log(m))$.]

[We are expecting: Your answer and a convincing justification. You do not need to write a formal proof; and you may assume that *n* is a power of *2* if it helps.]

5. Consider the function T (n) defined by

$$T(n) = \begin{cases} 2T(\lceil n/2 \rceil) + n/2 & n > 1 \\ 1 & n = 1 \end{cases}.$$

Using an argument by induction (not using the Master Method), prove that T (n) = Ω(n log(n)).

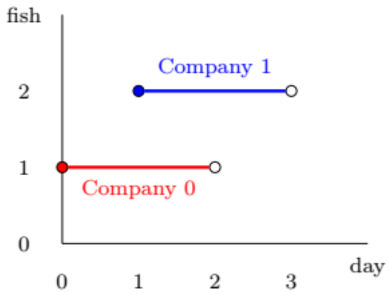
[We are expecting: A formal proof by induction. Make sure you explicitly state your inductive hypothesis, base case, inductive step, and conclusion.]

6. Plucky the Pedantic Penguin sometimes does consulting work on the side. (He points out indexing errors for bay area start-ups) There are n companies who are interested in Plucky's work. Plucky can work for at most one company during a given day. Each company has a range of times when they are interested in Plucky's work, and an amount they are willing to pay: between a_i and b_i (including a_i and not including b_i), Company i is willing to pay Plucky f_i fish. Here, a_i, b_i, f_i are all positive integers and a_i b_i. Each day, Plucky chooses to work for the highest bidder. If there is no company interested in Plucky's work on a day, Plucky gets zero fish that day.

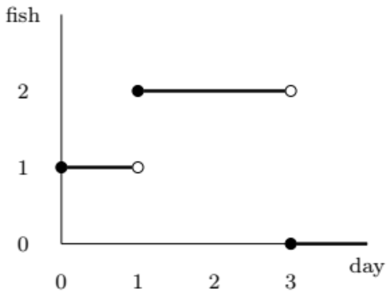
   Plucky gets the bids (a_i, b_i, f_i) as inputs, and wants to make a plot of how many fish he will receive each day. To understand the format he wants the output in, see the example below.

Example: Suppose that $n = 2$. As input, Plucky would get the following data, which can be visualized as the graph below.

Company i	a_i	b_i	f_i
Company 0	0	2	1
Company 1	1	3	2



In this example, Plucky would work for Company 0 at day 0, and receive one fish. He'd work for Company 1 on days 1, 2, and receive two fish on each of those days. On days 3 and onwards, no company was interested in Plucky's work, so he works for no company and receives zero fish. So his output plot would look like this:



To return this plot, Plucky will return a sequence $(t_0, f_0), (t_1, f_1), \ldots$, with $t_i \leq t_{i+1}$, which we interpret as meaning "starting on day t_i and ending on day $t_{i+1} - 1$, Plucky makes f_i fish. In the example above, the return value would be $(t_0 = 0, f_0 = 1), (t_1 = 1, f_1 = 2), (t_2 = 3, f_2 = 0)$.

Notes:

- The last f-value will always be 0.
- In the example above, it would also be correct to return $(t_0 = 0, f_0 = 4), (t_1 = 0, f_1 = 1), (t_2 = 1, f_2 = 2), (t_3 = 3, f_3 = 0)$; that is, adding extraneous intervals of length 0 is still correct.
- In the example above, it would also be correct to return $(t_0 = 0, f_0 = 1), (t_1 = 1, f_1 = 2), (t_2 = 2, f_2 = 2), (t_3 = 3, f_3 = 0)$; that is, breaking an interval into two smaller intervals is still correct.

In this problem you'll design an algorithm for Plucky. Your algorithm should take as input a list of n bids (ai, bi, fi), one for each company i ∈ {0, . . . , n-1} , and return a list ***fishPlot*** of (ti, fi) pairs as described in the example above.

a) Describe a simple $O(n^2)$-time algorithm for Plucky.
[We are expecting: Pseudocode, and a short English description explaining the main idea of the algorithm. No justification of the correctness or running time is required.]

b) Design a divide-and-conquer algorithm that takes time $O(n \log(n))$.
[We are expecting: Pseudocode, and a short English description explaining the main idea of the algorithm. We are also expecting an informal justification of correctness and of the running time.]