SSC0503 - Introdução à Ciência de Computação II
2ª Lista
Professor: Claudio Fabiano Motta Toledo (`claudio@icmc.usp.br`)
Estagiário PAE: Jesimar da Silva Arantes (`jesimar.arantes@usp.br`)

---

1. Desenvolva algoritmos recursivos para os seguintes problemas:
   1. Impressão de um número natural em base binária.
   2. Multiplicação de dois números naturais, através de somas sucessivas (Ex.: $6.4 = 4+4+4+4+4+4$).
   3. Cálculo de $1 + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{N}$
   4. A partir de um vetor de números inteiros, calcule a soma e o produto dos elementos do vetor.
   5. Gerador de máximo divisor comum (mdc):
      - $mdc(x, y) = y$, se $x \geq y$ e $xmody = 0$.
      - $mdc(x, y) = mdc(y, x)$, se $x < y$.
      - $mdc(x, y) = mdc(y, xmody)$, caso contrário.
   6. Verifique se uma palavra é palíndromo.
   7. Dado um número n, gere todas as possíveis combinações com as n primeiras letras do alfabeto.

2. Usando o método da substituição, prove que:
   - $T(n) = T(n-1) + c$ é ???, c constante, $n > 1$ e $T(1) = 0$. Dica: expanda e conjecture, antes de verificar.
   - $T(n) = cT(n-1)$ para $n > 0$ é ??? com $T(0) = k$, c e k constantes. Dica: expanda e conjecture, antes de verificar.
   - $T(n) = 3T(n/2) + n$ para $n > 1$ é ??? com $T(1) = 1$. Dica1: resolva a recorrência para $n = 2^k$.
   - $T(n) = 3T(n/3) + 1$ para $n > 1$ é ??? com $T(1) = 1$. Dica1: resolva a recorrência para $n = 3^k$.
   - $T(n) = T(n-1) + n$ é $O(n^2)$.
   - $T(n) = T(\lceil n/2 \rceil) + 1$ é $O(lgn)$.
   - $T(n) = 2T(\lfloor n/2 \rfloor) + n$ é $\Theta(n\ lg\ n)$. Dica: Provamos em sala que $T(n) = O(n\ lg\ n)$.
   - $T(n) = T(\lfloor n/2 \rfloor + 17) + n$ é $O(n\ lg\ n)$.

3. Use árvore de recursão para determinar um bom limitante assintótico superior e o método da substituição para verificar a resposta.
   - $T(n) = 3T(n/2) + n$.

- $T(n) = T(n/2) + n^2$.
- $T(n) = 4T(n/2 + 2) + n$.
- $T(n) = 2T(n - 1) + 1$.
- $T(n) = T(n - 1) + T(n/2) + n$.
- $T(n) = 4T(n/2) + cn$ para c constante.

4. Considere o algoritmo a seguir. Suponha que a operação crucial é o fato de inspecionar um elemento. O algoritmo inspeciona os n elementos de um conjunto e, de alguma forma, isso permite descartar $2n/5$ dos elementos e então fazer uma chamada recursiva sobre os $3n/5$ elementos restantes.

```
void Pesquisa (int n)
{
 if (n <= 1)
    'inspecione elemento' e termine;
 else {
    para cada um dos n elementos 'inspecione o elemento';
    Pesquisa (3n/5);
    }
}
```

- Faça a análise de complexidade da função Pesquisa.

5. Considere o seguinte algoritmo recursivo que devolve a soma dos primeiros n cubos.

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

- Faça a análise de complexidade da função Cubo.

6. Considere a função do código fonte na Figura 6.

- Dê a relação de recorrência para o número de operações realizadas em uma entrada de tamanho n em função da chamada recursiva.
- Desenhe a árvore de chamadas recursivas para n =3 e n = 4.

```
unsigned long fatorial(int n) {
unsigned long resultado = 1;
if (n > 1) {
 resultado = n * fatorial(n-1);
}
return resultado;
}
```

- Resolva a relação de recorrência, obtendo uma função operações diretamente em função de f(n) que dê o número de n.

7. Considerando a função do código fonte na Figura 7.

```
double power(double val, unsigned int pow)
{
  if (pow == 0) /* pow(x, 0) returns 1 */
    return(1.0);
  else
 }
   return(power(val, pow - 1) * val);
}
```

   - Dê a relação de recorrência para o número de operações realizadas em uma entrada de tamanho n em função da chamada recursiva.
   - Desenhe a árvore de chamadas recursivas para n=3 e n=4.
   - Resolva a relação de recorrência, obtendo uma função operações diretamente em função de n.

8. Considere o código da Figura 1, sendo operações relevantes as atribuições e comparações.
   - Encontre a função g(n) que representa o número de operações realizadas pela função *preenche*.
   - Encontre uma relação de recorrência f(n) para a função *menores*, em termos da execução do laço mais externo, trocando essa repetição por uma recursão.
   - Encontre a forma fechada para f(n).
   - Escreva a função T(n) incluindo todas as operações da função *main*. A partir de T(n), encontre a função de eficiência assintótica usando as notações O, Ω e Θ. Use a definição formal e encontre as constantes.

9. Implemente as versões recursiva e iterativa da função para obter os números de Fibonacci em C .
   - Utilize a biblioteca time.h para medir e observar o tempo necessário para calcular n = 15, 30, 45 e 60 , utilizando as duas versões.
   - Faça a análise de complexidade da função iterativa: obtenha uma função f (n) pela contagem de operações de soma e atribuição e encontre as constantes de forma a mostrar o limite assintótico superior pela notação O. Imprima os resultados obtidos (contagem de operações e constantes) em um arquivo.

```
void preenche(int *v, int N) {
 int i;
 for (i = 0; i < N; i++) {
 *(values+i) = rand()%1000;
 }
}
```

```
void menores(int *v, int* lt, int N) {
   for (i = 0; i < N; i++) {
   for (j = 0; j <= i; j++) {
      if (v[j] < v[i])
          lt[i]++;
   }
   }
}
```

```
int main (void) {
 srand(NULL);
 int N;
 scanf("%d", &N);
 int *values = malloc(N*sizeof(int));
 int *lessthan = calloc(N,sizeof(int))
 int i,j;
 preenche(values,N);
 menores(values,lessthan, N);
 free(values);
 free(lessthan);
return 0;
}
```

Figure 1: Funções preenche e menores.

- Faça a análise de complexidade da função recursiva: obtenha uma relação de recorrência g(n) pela contagem das operações de soma e atribuição , expanda a equação e a partir da expansão, encontre a equação fechada para a qual a função alcance o caso base. A partir da equação fechada, encontre as constantes de forma a mostrar o limite assintótico superior pela notação O.