Lista de exercícios No. 1

1. Resolva os seguintes exercícios do capítulo 1 do livro "Projeto de Algoritmos" de Nivio Ziviani:

- Exercício 2: O que significa dizer que uma função g(n) é O(f(n))?
- Exercício 3: O que significa dizer que um algoritmo executa em tempo proporcional a n?
- Exercício 4: Explique a diferença entre O(1) e O(2).
- Exercício 5: Qual algoritmo você prefere: um que requer n^5 passos ou outro que requer 2^n passos?
- Exercício 7: Prove que $f(n) = 1^2+2^2+...+n^2$ é igual a $n^3/3 + O(n^2)$
- Exercício 9: Indique se as afirmativas a seguir são verdadeiras ou falsas e justifique a sua resposta.
  a) $2^{n+1} = O(2^n)$
  b) $2^{2n} = O(2^n)$
  c) $f(n) = O(u(n))$ e $g(n) = O(v(n)) \Rightarrow f(n) + g(n) = O(u(n) + v(n))$
- Exercício 14: Considere o problema de encontrar a posição de inserção de um novo elemento em um conjunto ordenado:

  `A[1] > A[2] > A[3] > ... > A[n]`

  a) Apresente a situação e/ou entrada de dados em que ocorre o melhor caso e o pior caso.
  b) Apresente um algoritmo para resolver o problema acima.

- Exercício 16: Avalie as seguintes somas:
  a) $\sum_{i=1}^{n} i$
  b) $\sum_{i=1}^{n} a^i$
  c) $\sum_{i=1}^{n} ia^i$
  d) $\sum_{i=1}^{n} \log i$
  e) $\sum_{i=1}^{n} i2^{-i}$
  f) $1 + 1/7 + 1/49 + \cdots + (1/7)^n$

2. Resolva os seguintes exercícios do livro "Algoritmos" de Cormen *et al.*:

- Exercício 1.2-2 e 1-2-3

1.2-2
Vamos supor que estamos comparando implementações de ordenação por inserção e ordenação por intercalação na mesma máquina. Para entradas de tamanho n, a ordenação por inserção é executada em $8n^2$ etapas, enquanto a ordenação por intercalação é executada em $64n \lg n$ etapas. Para que valores de n a ordenação por inserção supera a ordenação por intercalação?

1.2-3
Qual é o menor valor de n tal que um algoritmo cujo tempo de execução é $100n^2$ funciona mais rápido que um algoritmo cujo tempo de execução é 2^n na mesma máquina?

- Exercício 2.2-3

2.2-3
Considere mais uma vez a pesquisa linear (ver Exercício 2.1-3). Quantos elementos da seqüência de entrada precisam ser verificados em média, supondo-se que o elemento que está sendo procurado tenha a mesma probabilidade de ser qualquer elemento no arranjo? E no pior caso? Quais são os tempos de execução do caso médio e do pior caso da pesquisa linear em notação Θ? Justifique suas respostas.

- Exercício 2.3-5

2.3-5
Voltando ao problema da pesquisa (ver Exercício 2.1-3) observe que, se a seqüência A estiver ordenada, poderemos comparar o ponto médio da seqüência com v e eliminar metade da seqüência de consideração posterior. A ***pesquisa binária*** é um algoritmo que repete esse procedimento, dividindo ao meio o tamanho da porção restante da seqüência a cada vez. Escreva pseudocódigo, sendo ele iterativo ou recursivo, para pesquisa binária. Demonstre que o tempo de execução do pior caso da pesquisa binária é $\Theta(\lg n)$.

- Exercício 2.3-7

2.3-7 *
Descreva um algoritmo de tempo $\Theta(n \lg n)$ que, dado um conjunto S de n inteiros e outro inteiro x, determine se existem ou não dois elementos em S cuja soma seja exatamente x.

- Problema 3-4

3-4 Propriedades da notação assintótica

Sejam $f(n)$ e $g(n)$ funções assintoticamente positivas. Prove ou conteste cada uma das seguintes conjecturas.

a. $f(n) = O(g(n))$ implica $g(n) = O(f(n))$.

b. $f(n) + g(n) = \Theta(\min(f(n), g(n)))$.

c. $f(n) = O(g(n))$ implica $\lg(f(n)) = O(\lg(g(n)))$, onde $\lg(g(n)) \geq 1$ e $f(n) \geq 1$ para todo n suficientemente grande.

d. $f(n) = O(g(n))$ implica $2^{f(n)} = O(2^{g(n)})$.

e. $f(n) = O((f(n))^2)$.

f. $f(n) = O(g(n))$ implica $g(n) = \Omega(f(n))$.

g. $f(n) = \Theta(f(n/2))$.

3. Existem algoritmos de força bruta que são ótimos? Exemplifique.

4. Diga se cada afirmativa abaixo é falsa ou verdadeira, justificando sua resposta:

   a) Um programa que é $O(n^2)$ em tempo de execução executa mais rápido que um programa que é $O(2^n)$.
   b) O MergeSort é um algoritmo de ordenação ótimo.
   c) Um programa é dito uma solução de compromisso entre tempo e espaço de armazenamento se utilizar o mínimo de memória possível.

5. Escrever uma solução para o problema de encontrar a moda de uma lista, utilizando as seguintes técnicas. Faça a análise das soluções.
   a) força bruta
   b) transformação

6. Qual das funções abaixo pode ser considerada uma solução de compromisso? Justifique.
   Obs: a função *alog* retorna o logaritmo de um número.

```
void PreencheVetor1(unsigned char *X, float *Y, int n)
{  for (int i=0; i<n; i++)
       if (X[i]==0) Y[i]=0.0; else Y[i]=alog(X[i]);
}

void PreencheVetor2(unsigned char *X, float *Y, int n)
{  float Z[256];
   for (Z[0]=0.0, int i=1; i<256; i++) Z[i]=alog(i);
   for (int i=0; i<n; i++) Y[i]=Z[X[i]];
}
```

7. Analise a seguinte versão não-recursiva do MergeSort, onde *n* é o tamanho do vetor a ser ordenado. A função *Merge(i1,i2,k)* intercala os subvetores de tamanho *k* que se iniciam nas posições *i1* e *i2* gastando para isso *2k-1* comparações entre elementos.

```
k=1
Enquanto k<n
    Repita para j=0 até n/(2*k) - 1
           P=j*2*k
           Merge(p,p+k,k)
    Fim repita
    k=k*2
fim enquanto
```

a) Forneça a função de custo que mede o número de comparações entre elementos do vetor em função do tamanho do vetor, *n*, para o pior caso.
b) Forneça a ordem de complexidade mais precisa para o pior caso, utilizando as notações O, Ω e Θ.
c) O algoritmo é ótimo? Justifique.

8. Determine a função de custo *f(n)* que determina o número de chamadas do procedimento *Processa,* no caso médio, para cada trecho abaixo. Determine a classe de comportamento assintótico em cada caso.

a)
```
Repita para  i=1 até n
        Repita para j=1 até i
                Processa;
```

b)
```
i=0;
  Enquanto i<=n faça
        Processa;
        Repita para j=i até n*n
                Processa;
        i=i+2;
  Fim enquanto
```

c)
```
i←1
  Repita até i>n            // n potência de 2
        Gere um número aleatório x entre 1 e 100;
        Se x mod 2 = 1 entrão
                Para j←i até n faça
                        Processa
        Senão se x > 50 então Processa
        Senão
                j←1
                Enquanto j<n/2 faça
                        Processa
                        j←j*2
                Fim enquanto
        Fim senão
        i←i+1
  Fim repita
```

9. Uma questão de uma prova de PAA tinha o seguinte enunciado: "Dado o programa 'Ordena' abaixo, que ordena grandes arquivos, onde cada registro possui 500Kb e as chaves são inteiros que ficam na variável *vetor*, em memória primária, determine sua ordem de complexidade, justificando sua resposta.". Ana respondeu *"$\theta(n)$"* enquanto José respondeu *"$\theta(n^2)$"*. Ambos acertaram a questão. Descreva a justificativa dada por cada aluno para que o professor considerasse as duas respostas como certas.

```
Ordena(arq,vetor, n)
Para i=1 até n-1
    Min=i
    Para j=i+1 até n
        Se vetor[j] < vetor[min] então min=j
    TrocaChaves(vetor,min,i)
    TrocaRegistros(arq,min,i)
```

10. Para cada afirmativa abaixo, diga se é falsa ou verdadeira, justificando a sua resposta:

a) $f(n) = \begin{cases} n^2, n < 0 \\ n, n \geq 0 \end{cases}$ é $\theta\ (n^2)$

b) $f(n) = \log n^2$ é $O\ (\log n)$

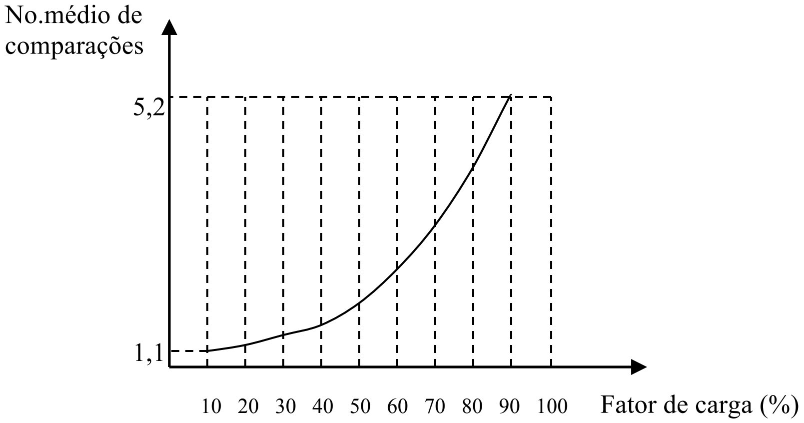
c) $f(n) = 2^{-n} + n$ é $\theta\ (n)$

d) $f(n) = n + \sqrt{n}$ é $O\ (n)$

e) Um algoritmo de ordenação ótimo será sempre mais eficiente que outro que não seja ótimo.

f) Se um algoritmo de ordenação tem o melhor caso igual a $\theta\ (nlogn)$ então ele é ótimo.

11. A eficiência de um método de pesquisa por hashing está relacionada ao fator de carga da tabela como indicado no gráfico a seguir. Por exemplo, se a tabela for dimensionada para ter 10 vezes mais entradas que o número de elementos a serem inseridos, o número médio de comparações necessárias para se recuperar um elemento é de 1,1. Em que situação o método pode ser considerado uma solução de compromisso? Justifique.



12. Você precisa avaliar um programa para o controle da temperatura de uma usina nuclear. O programa deve calcular o parâmetro de controle a cada 500 ms ou o reator corre o risco de explodir. Analisando o código, você percebe que a operação crítica é a multiplicação de números de ponto flutuante de precisão dupla. O fabricante informou apenas que o tempo médio de processamento do programa no processador de controle é de 10 ms. O pseudo-código do programa é dado a seguir, onde *n* é o número de sensores, que atualmente é em número de 1000, e *precisão* é uma variável entre 1 e 100 com igual probabilidade de assumir qualquer valor.

```
Função CalculaParâmetro (n: inteiro, V: matriz [1..n,1..n] de real, precisão: inteiro) retona real;
Inicio
  Controle = 0.0
  Se precisão <= 80 então
    Para i = 1 até n faça  controle = controle + V[i,i] * V[i,i]
  Senão se precisão <= 90 então
     Para i = 1 até n faça
       Para j = 1 até n faça controle = controle + V[i,j] * V[i,j]
   Senão
     Para i = 1 até n faça
       Para j = 1 até n faça
         Para k=1 até n faça controle = controle + V[i,k] * V[k,j]
   Retorne controle
  Fim
```

Baseado nos dados acima, você aprovaria o programa? Justifique, documentando sua resposta.

13. Considerando que a operação relevante é o número de vezes que a operação soma é executada, apresente a função de complexidade de tempo para:

a)
```
for i ← 1 to n do
   for j ← 1 to n do
      for k ← 1 to n do
         temp ← temp + i + j + k
```

b)
```
for i ← 1 to n do
   for j ← 1 to i do
      for k ← 1 to j do
         temp ← temp + i + j + k
```

c)
```
for i ← 1 to n do
   for j ← 1 to n do
      for k ← i to n do
         temp ← temp + i + j + k
```

d)
```
for i ← 1 to n do
   for j ← i to n do
      for k ← i to n do
         temp ← temp + i + j + k
```

e)
```
for i ← 1 to n do
   for j ← i to n do
      for k ← i to j do
      temp ← temp + i + j + k
```

14.
a) O que a função abaixo faz?
b) Qual é a operação relevante?
c) Qual é sua função de complexidade?

```
void p2 (int n)
{
   int i, j, x, y;

   x = y = 0;
   for (i=1; i<=n; i++) {
        for (j=i; j<=n; j++)
             x = x + 1;
        for (j=1; j<i; j++)
             y = y + 1;
   }
   cout<<x<<" "<<y;
}
```

15. Qual é a função de complexidade no pior caso para o número de atribuições ao vetor x?

```
void Exercicio3(int n){
   int i, j, a;

   for (i=0; i<n; i++){
      if (x[i] > 10)
         for (j=i+1; j<n; j++)
            x[j] = x[j] + 2;
      else {
         x[i] = 1;
         j = n-1;
         while (j >= 0) {
            x[j] = x[j] - 2;
            j = j - 1;
         }
      }
   }
}
```

16. Resolva as seguintes equações de recorrência:

a) $\begin{cases} T(n) = T(n-1) + c & \text{c constante, } n > 1 \\ T(1) = 0 \end{cases}$

b) $\begin{cases} T(n) = T(n-1) + 2^n & n \geq 1 \\ T(0) = 1 \end{cases}$

c) $\begin{cases} T(n) = cT(n-1) & \text{c, k constantes, } n > 0 \\ T(0) = k \end{cases}$

d) $\begin{cases} T(n) = 3T(n/2) + n & n > 1 \\ T(1) = 1 \end{cases}$

17. Use o teorema mestre para derivar um limite assintótico Θ para as seguintes recorrências:

a) $T(n) = 2T(n/2) + n - 1$
b) $T(n) = 3T(n/2) + n$
c) $T(n) = 4T(n/2) + n^2$
d) $T(n) = 4T(n/2) + n^3$

18. Apresente a complexidade de tempo para o procedimento abaixo:

```
PROCEDURE Pesquisa (n: integer);
 BEGIN
    IF n > 1 THEN
    BEGIN
       Inspecione n*n*n elementos;
       Pesquisa (2n/3);
    END;
 END;
```

19. Considere o algoritmo abaixo.

```
procedure Sort2 (var A: array[1..n] of integer; i,j: integer);
{-- n uma potencia de 3 --}
begin
   if i < j then
   begin
      k := ((j-i)+1)/3;
      Sort2(A,i,i+k-1);
      Sort2(A,i+k,i+2k-1);
      Sort2(A,i+2k,j);
      Merge(A,i,i+k,i+2k,j);
      { Merge intercala
        A[i…(i+k-1)], A[(i+k)..(i+2k-1)] e A[i+2k..j] em A[i..j]
   a um custo 5n/3-2}
   end
end
```

a) Escreva uma equação de recorrência que descreva este comportamento.
b) Converta esta equação para um somatório.

c) Dê a fórmula fechada para este somatório.

20. Dado o algoritmo abaixo, faça a análise em função do número de chamadas ao procedimento *Imprima*:

```
Permutação (v, i, n)
    Se i=n então imprima(v,n)
    senão para k:=1 até n-i+1 faça
            Permutação (v,i+1,n)
            Rotaciona (v,i,n)
```

21. O algoritmo de ordenação por inserção pode ser definido recursivamente da seguinte forma: para ordenar A[1..n], comece por ordenar A[1..n − 1], e depois insira A[n] no vetor A[1..n − 1] já ordenado.

    a) Descreva o tempo de execução desta implementação através de equações de recorrência.
    b) Desenhe as árvores de recursividade correspondentes ao melhor caso e ao pior caso. Utilize estas árvores para deduzir o comportamento assintótico deste algoritmo.

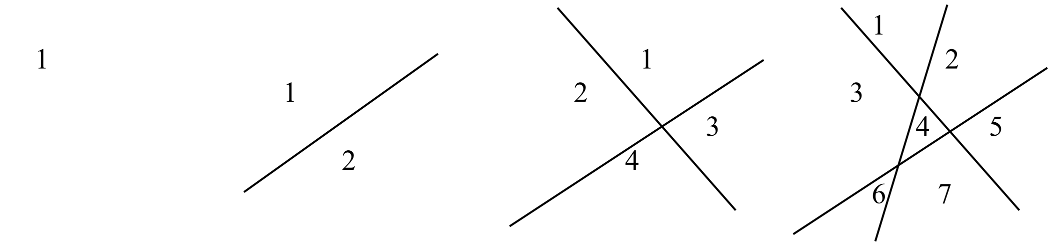
22. Analise as seguintes soluções para o problema da busca binária. Considere o vetor indexado de 1 a *n*, onde *n* é da forma 2^m-1, na primeira solução, e 2^m na segunda:

```
PesquisaA(item, vetor, i, j)
k=(i+j)/2
Se vetor[k]=item então retorne (k) /* encontrado */
Senão se i=j então retorne (-1)  /* não encontrado */
        Senão se item>vetor[k]
                então PesquisaA(item, vetor, k+1, j)
                senão PesquisaA(item, vetor, i, k-1)
```

```
PesquisaB(item, vetor, i, j)
Se i=j então
  Se vetor[i]=item então retorne (i) /* encontrado */
  Senão retorne (-1)  /* não encontrado */
Senão
    k=(i+j)/2
    Se item>vetor[k]
          então PesquisaB(item, vetor, k+1,j)
          senão PesquisaB(item, vetor, i, k)
```

a) Qual solução executa o menor número de comparações com o item pesquisado no melhor caso? Qual a ordem de complexidade? Mostre como chegou a esta conclusão.
b) b) Qual solução executa o menor número de comparações com o item pesquisado no pior caso? Qual a ordem de complexidade? Mostre como chegou a esta conclusão.
c) O caso médio de uma das soluções é trivial. Qual é ele? Como poderia ser feita a análise do caso médio para a outra solução?

23. Para o problema a seguir apresente a recorrência e a forma fechada. **Linhas no plano ou Cortando a sua pizza favorita**. Quantas fatias de pizza uma pessoa pode obter ao fazer *n* cortes retos com uma faca? Ou, expressando de outra forma, qual é o número máximo de regiões L_n determinado por *n* retas no plano? Lembre-se que um plano sem nenhuma reta tem uma região, com uma reta tem duas regiões e com duas retas tem quatro regiões, conforme mostrado na figura 2.



$L_0 = 1$ $L_1 = 2$ $L_2 = 4$ $L_3 = 7$

24. Dado o algoritmo abaixo:

```
void Processa(int v[], int esq, int dir)
{
   int *p = malloc(sizeof(int)*(dir - esq));

   if (esq < dir - 1)
   {
     int m=(esq+dir)/2;
     Processa(v, esq, m);
     Processa(v, m+1, dir);
   }
    for (int i=esq+1; i<=dir; i++) *(p+i-esq-1)=v[i]+v[i-1];
    for (int i=esq+1; i<=dir; i++) v[i]=*(p+i-esq-1);
}

Primeira chamada: Processa(v,0,n-1);
```

a) faça a análise em função do custo de memória (maior número possível de bytes alocados em um determinado instante, no pior caso) , em função de n (número de elementos de v). Considere que não existe coletor de lixo. Sugestão: Use uma árvore de recorrência para ilustrar o problema.
b) faça a análise em função do número de atualizações de elementos do vetor v, em função de n.

25. O algoritmo abaixo é uma alteração do anterior:

```
void Processa(int v[], int esq, int dir)
{
   int *p = malloc(sizeof(int)*(dir – esq));

   if (esq < dir - 1)
   {
     int m=(esq+dir)/2;
     Processa(v, esq, m);
     Processa(v, m+1, dir);
   }
    for (int i=esq+1; i<=d; i++) *(p+i-esq-1)=v[i]+v[i-1];
    for (int i=esq+1; i<=d; i++) v[i]=*(p+i-esq-1);
    free(p);
}
```

Faça a análise em função do custo de memória (maior número possível de bytes alocados em um determinado instante, no pior caso) , em função de n (número de elementos de v).