CENTRO DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO E AUTOMAÇÃO

Avaliação tipo: A
Disciplina: Algoritmo e estrutura de dados II
Professor: Eduardo de Lucena Falcão
Discente: *Igor Michael Araujo de Macedo (20210094429)*

1. Sejam $f(n)$ e $g(n)$ funções assintoticamente não-negativas, e $h(n) = max(f(n), g(n))$. **Prove ou mostre um contra-exemplo** para a seguinte afirmação: $h(n) \in \Theta(f(n) + g(n))$. (2.0)
   Dadas duas funções, $f(n)$ e $g(n)$, não negativas, então uma função $h(n)$, determinada pela maior entre elas, é limitada superiormente e inferiormente pela soma das duas. Isso se dá ao fato de que, ao serem somadas, o maior peso se dá pela maior função, sendo assim, $max(f(n), g(n))$ sempre será limitada superiormente e inferiormente por $f(n) + g(n)$.
   Exemplo:
   Dados $f(n) = n^2$ e $g(n) = n \Rightarrow h(n) = max(f(n), g(n)) = n^2$
   então $h(n) \in \Theta(f(n) + g(n)) \Rightarrow n^2 \in \Theta(n^2 + n) = \Theta(n^2)$

2. **Analise detalhadamente** a complexidade do seguinte algoritmo de busca. (2.0)

```
int busca(int v[], int n, int val) {
  int ini = 0;          // 1 execução
  int fim = n - 1;      // 1 execução
  int meio = -1;        // 1 execução
  while (ini <= fim) {
  // analisando (ini <= fim):
    // assumindo pior caso no qual val não existe na lista e é menor que o menor elemento
    // fim ∈ {n − 1, (n − 3)/2, (n − 7)/4, (n − 15)/8,
    // fim ∈ {n, n/2, n/4, n/8, ..., 1}
    // Soma de termos da PG: a_n = a_1 · q^(x−1) ⇒ 1 = n · (1/2)^(x−1)
    ⇒ (1/2)^(x−1) = (1/n) ⇒ 2^(x−1) = n ⇒ log_2(2^(x−1)) = log_2(n)
    ⇒ x − 1 = log_2(n) ⇒ x = log_2(n) + 1
    // ou seja, o while executa O(log_2(n)), assim como todas as linhas dentro dele
    meio = (ini + fim) / 2;
    if (v[meio] == val) {
      return meio;
    }
    else if (val < v[meio])
      fim = meio - 1;
    else
      ini = meio + 1;
  }
```

```
  return -1;
}
```

Portanto, analisando o algoritmo no pior caso, quando o elemento buscado não existe na lista, o algoritmo tem complexidade $O(log_2(n))$. *No melhor caso, quando o elemento buscado está exatamente no meio, o algoritmo tem complexidade* $O(1)$.

3. Prove que $2n^2 + 24 = \Theta(n^2)$. (2.0)

   Se $f(n) = 2n^2 + 24$ *e* $g(n) = n^2$, *então* $\exists\, c_1,\ c_2$ *e* n_0 *positivos tal que*

   $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n) \Rightarrow c_1 \cdot n^2 \leq 2n^2 + 24 \leq c_2 \cdot n^2$

   para $c_1 = 3$ *e* $c_2 = 4$, *temos*:

   $3n^2 \leq 2n^2 + 24 \leq 4n^2$

   $\Rightarrow 3n^2 \leq 2n^2 + 24 \Rightarrow n^2 \leq 24 \Rightarrow -4.9 \leq n \leq 4.9$

   $\Rightarrow 2n^2 + 24 \leq 4n^2 \Rightarrow 2n^2 \geq 24 \Rightarrow n^2 \geq 12 \Rightarrow n \leq 3.46$ *e* $n \geq 3.46$

   $\Rightarrow 3.46 \leq n \leq 4.9$

   Sendo assim, para $c_1 = 3$, $c_2 = 4$ *e* $n_0 = 4$, *foi demonstrado que* $f(n) = 2n^2 + 24$ *é limitado superiormente e inferiormente por* $g(n) = n^2$.

4. Use o método da **iteração e/ou árvore** para determinar um bom limite assintótico superior na recorrência $T(n) = 2T(n/2) + n$, com $T(1) = 1$. Verifique a sua resposta usando o teorema mestre. (4.0)

   Método iterativo:

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

   $\quad T(n/2) = 2T(n/4) + n/2$

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

   $\quad T(n/4) = 2T(n/8) + n/4$

   $T(n) = 4(2T(n/8) + n/4) + 2n = 8T(n/8) + n + 2n = 8T(n/8) + 3n$

   $\quad T(n/8) = 2T(n/16) + n/8$

   $T(n) = 8(2T(n/16) + n/8) + 3n = 16T(n/16) + n + 3n = 16T(n/16) + 4n$

   $\Rightarrow T(n) = 2T(n/2) + n$

   $\Rightarrow T(n) = 4T(n/4) + 2n$

   $\Rightarrow T(n) = 8T(n/8) + 3n$

   $\Rightarrow T(n) = 16T(n/16) + 4n$

   $\Rightarrow$ *No nível i:* $2^iT(n/2^i) + in$

   $\Rightarrow$ *No nível* $i = log_2(n)$: $T(n) = 2^{log_2(n)}T(n/2^{log_2(n)}) + n \cdot log_2(n)$

   $= nT(1) + n \cdot log_2(n) = n + n \cdot log_2(n)$

   $\Rightarrow T(n) \in O(n \cdot log_2(n))$

Teorema mestre:

$a = 2;\ b = 2;\ f(n) = n$

$n^{log_b a} = n^{log_2 2} = n^1$

$f(n) = n = \Theta(n) \Rightarrow T(n) = \Theta(n \cdot log_2(n))$