2. (**5 pontos** – 3/1/1) Considere um arranjo `A` com `n` números distintos (sendo `n` um múltiplo de 2) não ordenados. O problema é encontrar o maior e o segundo maior valores dentre estes `n` elementos.

   (a) Então, pede-se que seja apresentado uma função para resolver o problema cujo custo seja ótimo – pode-se apresentar uma função não ótima com ônus. Uma sugestão para o protótipo da função:

   ```
   int Max1Max2(int* A, int n, int* pM1, int* pM2)
   ```

   (b) Qual é a função de complexidade do número de comparações no melhor e pior casos da função implementada?

   (c) Que configuração dos arranjos (vetores) de entrada levam a essas duas situações?

3. (**2 pontos**) Considere que você tenha dois algoritmos de ordenação. O primeiro algoritmo é quadrático tanto no pior caso quanto no melhor caso. Já o segundo algoritmo, é linear no melhor caso e cúbico no pior caso. Considerando que o melhor caso ocorre 90% das vezes que você executa o programa enquanto o pior caso ocorre apenas 10% das vezes, qual algoritmo você escolheria? Justifique a sua resposta em função do tamanho da entrada.

4. (**4 pontos** – 1/1/1/1) Seja a função (código) abaixo, com `a` = 2, e para análise de complexidade desta leve em consideração o número de comparações realizadas. Considere também que a chamada inicial à função é a seguinte `Misterio(A,0,n-1,y)`, em que `y` é um dos elementos do vetor `A`.

```
/* n uma potencia de a */
int Misterio(int A[n], int i, int j, int x)
{
  m = (i+j)/a;
  if ( A[m] == x )
    return m;
  else if (A[m] > x)
    return Misterio(A,i,m-1,x);
  else
    return Misterio(A,m+1,j,x);
}
```

   (a) O que a função acima faz?

   (b) Qual é a equação de recorrência da função? (estabeleça $T(n)$)

   (c) Apresente a fórmula fechada (elimine possíveis somatórios) para a função de complexidade.

   (d) Qual é a ordem de complexidade da função/programa? ($g(n) = O(T(n))$)