LISTA 5

1º)

```
maior_salto(L, inicio, fim):
    se inicio = fim:
        retorna 0, L[inicio], L[inicio]

    meio = inicio + (fim-inicio) / 2

    salto_esquerdo, esquerdo_minimo, esquerdo_maximo = maior_salto(L, inicio, meio)
    salto_direito, direito_minimo, direito_maximo = maior_salto(L, meio+1, fim)

    x = direito_maximo - esquerdo_minimo

    salto_maximo = max(salto_esquerdo, salto_direito, x)
    retorna (salto_maximo, min(esquerdo_minimo, direito_minimo), max(esquerdo_maximo, direito_maximo))
```

2º)

Infelizmente não consegui encontrar uma solução para O(n).

3º)

```
achar_par(L, n):
    para i de 0 até n-1:
        se L[i+1] - L[i] <= ((L[n-1] - L[0]) / (n-1))
            retorna (L[i], L[i+1])
    retorna (nil, nil)
```

A complexidade é O(n), onde n é o tamanho da lista.
Se ela precisasse ser ordenada, seria O(nlog n).

4:)

```
encontrar_elemento_majoritario(L, n):
    candidato = nil
    contagem = 0

    para i = 0 até n:
        se (contagem == 0):
            candidato = L[i]
            contagem = 1
        senão se (L[i]  == candidato):
            contagem++
        senão:
            contagem--

    contagem = 0
    para i = 0 até n:
        se (L[i] == candidato):
            contagem++
    se (contagem > n / 2):
        retorna candidato
    senão:
        retorna -1
```

Encontrei um algoritmo chamado Boyer-Moore Voting,
e ele percorre a lista 2 vezes: uma pra encontrar o candidato
e outra pra confirmar se o candidato é o elemento majoritário.
Tudo isso executa em O(n).

5:)

```
achar_elemento_falta(L, n):
    inicio, fim = 0, n -1
    enquanto inicio <= fim:
        meio = (inicio+fim) / 2
        se L[meio] == meio:
            inicio = meio + 1
        senão:
            fim = meio - 1
    retorna inicio
```

Esse algoritmo percorre em O(log n), pois a lista já está ordenada.

```
achar_elemento_falta(L, n):
    xor_lista, xor_total = 0, 0
    para i de 0 até n-1:
        xor_lista ^= i
    para i de 0 até n:
        xor_total ^= i
    retorne xor_lista ^ xor_total
```

Já esse algoritmo percorre em O(n).