Aluno: Thiago Fernandes Ribeiro RA:120160

1) **Identificar um algoritmo que comumente é executado de forma incremental e mostrar como o mesmo pode ser executado na abordagem dividir e conquistar.**

O algoritmo MinMax, utilizado para determinar o menor e o maior elemento de um vetor de números, pode ser utilizado da maneira incremental e da maneira recursiva.

Da maneira incremental, o algoritmo percorre o vetor comparando o elemento, i, com um valor min e um valor max, que são definidos, por convenção, como sendo o primeiro elemento do vetor.

Da maneira recursiva, o vetor é dividido ao meio, onde um elemento da metade mais à esquerda é comparado com um elemento do lado mais à direita e assim determinando o mínimo e o máximo.

Exemplo: Suponhamos que int vetor[4] = {2, 20, 1, 0}
dividindo ao meio e separando as partes:
maisEsquerda = {2, 20}
maisDireita = {1, 0}

#Retirando o mínimo das duas metades 2 < 20, min = 2

1 < 0, min = 0

#Minimo do vetor, comparação do mínimo da parte à esquerda com o mínimo da parte à direita

2 < 0, min = 0

#Retirando o máximo das duas metades 2 > 20, max = 20

1 > 0, max = 1

#Máximo do vetor, comparação do máximo da parte à esquerda com o máximo da parte à direita

20 > 1, max = 20

return min, max # 0, 20

**2) Usando a Figura do Slide 11, ilustre a operação de ordenação por interação para o arranjo A = {3, 41, 52, 26, 38, 59, 9, 49}**

Subdivide o vetor, até chegar no menor vetor possível, depois vai intercalando e tirando o mínimo entre as listas até juntar novamente no vetor original, já ordenado.

[3, 41, 52, 26] | [38, 59, 9, 49]
[3, 41] [52, 26] | [38, 59] [9, 49]
[3] [41] [52] [26] | [38] [59] [9, 49]
Intercalando
[3, 41][26, 52] | [38, 59] [9, 49]
[3, 26, 41, 52] | [9, 38, 49, 59]
[3, 9, 26, 41, 49, 52, 59]

**3) Descreva um algoritmo de tempo O (n log n) que, dado um conjunto S de n inteiro e um outro inteiro x, determine se existem ou não dois elementos em S cuja soma da exatamente x.**

```
unsigned int soma_igual(int vetor[], int x,
  unsigned int possivel, unsigned int tamanho_vetor)
{
  merge_sort(vetor);
  possivel = 0;
  esquerda = 1;
  direita = tamanho_vetor;

  while(esquerda < direita && vetor[esquerda] + vetor[direita] != x)
  {
      if(vetor[esquerda]+S[direita] > x)
      direita--;
      else esquerda++;
  }

  if(esquerda < direita) possivel = 1;

return possivel;
```

Inicialmente, se estabelece dois índices, o da esquerda e o da direita. O da esquerda estará inicialmente na primeira posição apontando, deste modo, o menor elemento do vetor, enquanto que o da direita estará inicialmente na última posição apontando, deste modo, o maior elemento do vetor. Se a soma do elemento do elemento da esquerda com o elemento da direita for igual a x, o elemento existe e o algoritmo termina, caso seja maior que x, o elemento da direita aponta para o seu elemento anterior e caso seja menor que x, o elemento da esquerda aponta para o seu sucessor. A complexidade do Merge Sort é O(n log n) e a complexidade das instruções abaixo do Merge Sort é O(n), e aplicando a propriedade de soma da Notação Big O: O(nlogn) + O(n) = O(2n + logn) = O(n + logn) Logo, o algoritmo é classificado como n log n.

**4) Fórmula de recorrência - a^n para n >= 0**

A fórmula de recorrência para o problema é:

$$\begin{cases} a(a^2)^{\frac{n-1}{2}}, n \mod 2 \neq 0 \\ (a^2)^{\frac{n}{2}}, n \mod 2 = 0 \end{cases}$$

**5) Fórmula de recorrência - MinMax**

A fórmula de recorrência é:

$$T(n) = \begin{cases} T\left(\lfloor \frac{n}{2} \rfloor\right) + T\left(\lceil \frac{n}{2} \rceil\right) + 2 & for\ n > 2 \\ 1 & for\ n = 2 \\ 0 & for\ n = 1 \end{cases}$$

Assumindo que n está na potência de 2, logo, n = 2k onde k representa a altura da árvore de recursão e a sua fórmula é:

$$T(n) = 2.T(\frac{n}{2}) + 2 = 2.\left(\ 2.T(\frac{n}{4}) + 2\ \right) + 2..... = \frac{3n}{2} - 2$$