Lista de Exercícios 3

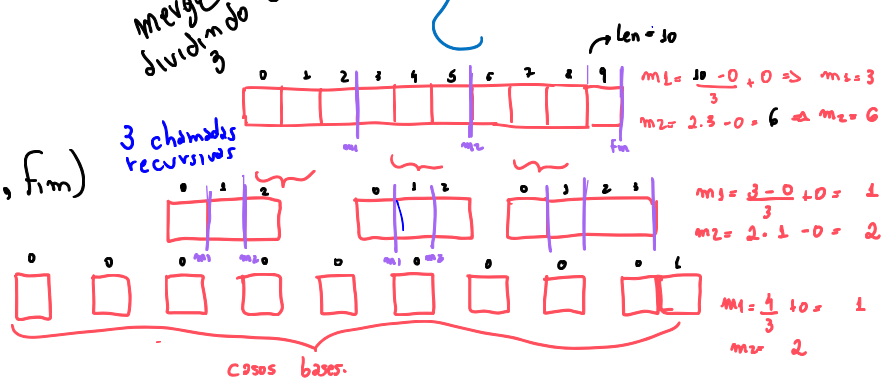
Universidade Federal do Ceará - Campus Quixadá
Projeto e Análise de Algoritmo — QXD0041 – 2023.2 Prof.
Fabio Dias

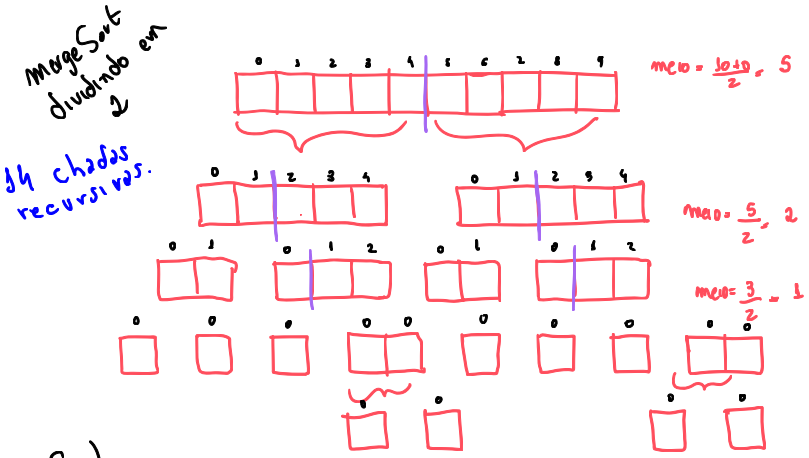
Divisão e Conquista

1. Vamos supor uma versão diferente do Merge-Sort, onde ele divide o vetor em 3 sub-vetores de tamanhos aproximadamente iguais. Depois ordena os 3 sub-vetores recursivamente e utiliza uma versão alterada do Intercala para realizar o Merge desses 3 subvetores ordenados em um único ordenado. Mostre o algoritmo dessa versão do Merge-Sort junto com o Intercala alterado. Realize a análise de complexidade desse algoritmo e explique se essa versão é mais eficiente do que a versão original.

```
mmerge (int *V, int inicio, int fim)
{
  Se (fim - inicio) <= 1
     retorna

  Se (fim - inicio) <= 2
     intercala (V, inicio, inicio + 1, fim-1, fim)
     retorna

  int m1 = (fim - inicio)/3 + inicio
  int m2 = 2. m1 - inicio

  mmerge(V, inicio, m1)
  mmerge(V, m1, m2)
  mmerge(V, m2, fim)

  intercala(V, inicio, m1, m2, fim)
}

intercala(int *V, inicio, m1, m2, fim)
{
  int *Vaux = new int [fim-inicio]
  int i=0, i1 = inicio, i2 = m1, i3 = m2

  enquanto (i1 < m1 e i2 < m2 e i3 < fim)
  {
    Se V[i1] <= V[i2] e V[i1] <= V[i3]
       Vaux[i++] = V[i1++]
    Se V[i2] <= V[i2] e V[i2] <= V[i3]
       Vaux[i++] = V[i2++]
    Se não
       Vaux[i++] = V[i3++]
  }
  enquanto(i1 < m1 e i2 < m)
  {
    Se V[i1] <= V[i2]  Vaux[i++] = V[i1++]
  } Se não  Vaux[i++] = V[i2++]
```

$$T(n) = \begin{cases} O(1) & \text{se } len < 3 \\ 3t\left(\frac{n}{3}\right) + O(n) & \end{cases}$$

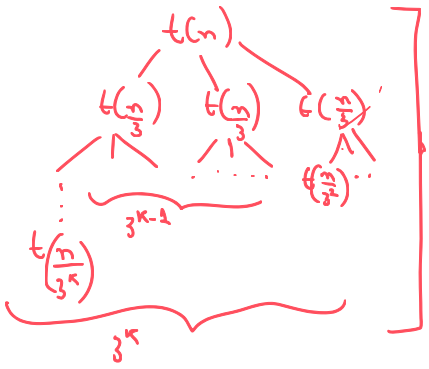




O mergeSorte dividindo em 3 realiza mais chamadas recursivas em cada interação e possui o caso base mais abrangente em relação ao mergeSorte convencional, assim atinge o caso base em menos interações recursivas. Mas em contra partida realiza mais verificações, afim de decidir a posição de um elemento. Em geral esse algoritmo realiza menos chamadas recursiva mas realiza mais comparações para cada chamada,

```
enquanto ( i1 < m1 e i3 < fim)
    . . .
enquanto (i2 < m2 e i3 < fim)
    . . .
enquanto (i1 < m1) ; enquanto (i2 < m2) ; enquanto (i3 < fim)
    . . .

i = 0

enquanto (inicio < fim)
{
  V[inicio++] = Vaux[i++]
}
}
```

assim a eficiência desse algoritmo está relacionado ao custo das comparações adicionais em relação ao maior número de chamadas recursivas da versão convencional.



$$t(n) \begin{cases} 3t\left(\frac{n}{3}\right) + O(n) \\ O(1) \end{cases}$$

Complexidade é $O(n \log_3 n)$

```
void intercala(int *V, int inicio, int m1, int m2, int fim)
{
   int * aux = new int[fim-inicio + 1];

   int i = 0;

   int i1 = inicio;
   int i2 = m1;
   int i3 = m2;

   while(i1 < m1 && i2 < m2 && i3 < fim)
   {
      if(V[i1] <= V[i2] && V[i1] <= V[i3])
         aux[i++] = V[i1++];
      else if(V[i2] <= V[i1] && V[i2] <= V[i3])
         aux[i++] = V[i2++];
      else if(V[i3] <= V[i1] && V[i3] <= V[i2])
         aux[i++] = V[i3++];
   }

   while(i1 < m1 && i2 < m2)
   {
      if(V[i1] <= V[i2])
         aux[i++] = V[i1++];
      else if(V[i2] <= V[i1])
         aux[i++] = V[i2++];
   }

   while(i1 < m1 && i3 < fim)
   {
      if(V[i1] <= V[i3])
         aux[i++] = V[i1++];
      else if(V[i3] <= V[i1])
         aux[i++] = V[i3++];
   }

   while(i2 < m2 && i3 < fim)
   {
      if(V[i2] <= V[i3])
         aux[i++] = V[i2++];
      else if(V[i3] <= V[i2])
         aux[i++] = V[i3++];
   }

   while(i1 < m1)
   {
      aux[i++] = V[i1++];
   }

   while(i2 < m2)
   {
      aux[i++] = V[i2++];
   }
   while(i3 < fim)
   {
      aux[i++] = V[i3++];
   }

   for(int j = inicio, i = 0; j<fim; j++, i++)
      V[j] = aux[i];
}
void merge(int *V, int inicio, int fim)
{
   if(fim <= inicio)
      return;
   if(fim - inicio <= 2)
   {
      intercala(V, inicio, inicio + 1, fim-1, fim);
      return;
   }

   int m1 = (fim - inicio)/3 + inicio;
   int m2 = 2 * m1 - inicio;


   merge(V, inicio, m1);
   merge(V, m1, m2);
   merge(V, m2, fim);

   intercala(V, inicio, m1, m2, fim);
}
```

```
void intercala(int *V, int inicio,int meio, int
fim)
{
   int *aux = new int[fim - inicio + 1];
   int i = 0;

   int i1 = inicio;
   int i2 = meio;

   while(i1 < meio && i2 < fim)
   {
      if(V[i1] < V[i2])
         aux[i++] = V[i1++];
      else
         aux[i++] = V[i2++];
   }

   while(i1 < meio)
      aux[i++] = V[i1++];

   while(i2 < fim)
      aux[i++] = V[i2++];

   i = 0;

   while(inicio < fim)
      V[inicio++] = aux[i++];
}

void merge(int *V, int inicio, int fim)
{
   if(fim - inicio <= 1)
      return;

   int meio = (fim + inicio)/2;

   merge(V, inicio, meio);
   merge(V, meio, fim);

   intercala(V, inicio, meio, fim);
}
```