Recorrências básicas

Considere os seguintes algoritmos:

```
public static int maxIterativo(int array[], int N){
   int max=Integer.MIN_VALUE;
   for(int i=0;i<N;i++)
        if(array[i]>max) max=array[i];
   return max;
}

public static int maxRecursivo(int array[], int N, int maxActual){
   if(N==0) return maxActual;
   if(array[N-1]> maxActual){
        return max(array,N-1,array[N-1]);}
   else{
        return max(array,N-1,maxActual);}
}
```


Recorrências básicas

na invocação do método **maxRecursivo** executa-se:

- algumas instruções (digamos **um número constante**);
- depois voltamos a chamar a mesmo método agora apenas com **N-1** objectos.

Número total de instruções executadas é

$C(n) = C(n-1) + O(1)$

uma recorrência!

Recorrências básicas

- Programa recursivo que, em cada passo, analisa um dado de entrada para eliminar um item
  - $C(N)= C(N-1)+ O(1)$ para $N>=2$ com $C(1)=O(1)$
  - Solução: $C(N)$ é aproximadamente N, i.e., $C=O(N)$

$$
\begin{aligned}
C(N) &= C(N-1) + O(1) \\
&= C(N-2) + O(1) + O(1) \\
&= \ldots \\
&= C(1) + \underbrace{O(1) + \ldots + O(1)}_{(N-1)*O(1)} \\
&= O(N)
\end{aligned}
$$

Recorrências básicas

Considere os seguintes algoritmos:

```
public static void paresIterativo(int array[],int N,int x){
   for(int i=0;i<N;i++)
      for(int j=i;j<N;j++)
        if(array[i]+array[j]==x)
         System.out.println(array[i] + "+ " + array[j] + " = " + x);
}

public static void paresRecursivo(int[] array,int x, int N){
   if(N==0)return;
   for(int i=0; i<N;i++){
      if( array[N-1] + array[i]==x)
        System.out.println(array[N-1]+ "+ " + array[i] + " = " + x);
   }
   paresRecursivo(array,x,N-1);
}
```

Recorrências básicas

- Programa recursivo que, em cada passo, analisa todos os dados de entrada para eliminar um item
  - $C(N)= C(N-1)+ O(N)$ **para** $N>=2$ **com** $C(1)=O(1)$
  - **Solução:** $C(N)$ **é aproximadamente** $N^2/2$, **i.e.,** $C(N)=O(N^2)$

$$
\begin{aligned}
C(N) &= C(N-1) + O(N) \\
&= C(N-2) + O(N-1) + O(N) \\
&= C(N-3) + O(N-2) + O(N-1) + O(N) \\
&= \ldots \\
&= C(1)+O(2)+\ldots+O(N-2)+O(N-1)+ O(N) \\
&= O(N(N+1)/2) \\
&= O(N^2/2)=O(N^2)
\end{aligned}
$$

Recorrências básicas

Considere os seguintes algoritmos:

```
/*Procura Binária Iterativa*/
public static int pBIterativa(int num,int[] array,int first,int last){
   while (last > = first){
        int medio = (last+first)/2 ;
        if (num == a[medio]) return medio;
        if (num <a[medio]) last = medio-1;
        else first = medio + 1;
   }
   return -1 ;
}
```

- Propriedade: A procura binária nunca examina mais do que $\lfloor \lg N \rfloor + 1$ números!

Recorrências básicas

Considere os seguintes algoritmos:

```
/*Procura binária recursiva*/
public static int pBRecursiva(int num,int[] array,int first,int last){
  int result=-1,mid;
  if(first>last) result=-1;
  else{
        mid=(first+last)/2;
        if(num==array[mid]) result=mid;
        else{
             if(num<array[mid])
                  result=pBRecursiva(num,array,first,mid-1);
              else
                   result=pBRecursiva(num,array,mid+1,last);
           }
      }
  return result;
}
```


Recorrências básicas

- Programa recursivo que, em cada passo, analisa divide em dois os dados de entrada
  - $C(N)= C(N/2)+ O(1)$ para $N>=2$ com $C(1)=O(1)$
  - **Solução: $C(N)$ é aproximadamente log N, i.e.,**

$$C(N)=O(\log N)$$

Seja $n=\log N$ (i.e., $N=2^n$). Então,

$$C(2^n)=C(2^{n-1}) + O(1)=C(2^{n-2}) + O(1) + O(1)$$
$$= \ldots$$
$$=C(2^0)+ O(n)$$
$$=O(n) + O(1)$$
$$=O(n)$$

Portanto, $C(N)=O(\log N)$

Recorrências básicas

Considere os seguintes algoritmos:

```
public static void mergeSort(int[] a){
  if(a.length>=2){
     int meio=a.length/2; int[] frente=new int[meio];
     int[] cauda=new int[a.length-meio];
/*divide os elementos de a pelos dois arrays:frente e cauda */
     divide(a,frente, cauda);
     mergeSort(frente); /*ordenar o array frente*/
     mergeSort(cauda);/*ordenar o array cauda*/
     merge(a,frente,cauda);
         /*junta os arrays no array a, ordenando-os*/
   }
}
public static void divide(int[] a,int[] frente,int[] cauda){
    int i;
    for(i=0; i<frente.length;i++)  frente[i]=a[i];
    for(i=0; i<cauda.length;i++)   cauda[i]=a[frente.length+i];
    }
```


Recorrências básicas

Considere os seguintes algoritmos:

```
public static void merge(int[] a,int[] frente,int[] cauda){
   int indexF=0,indexC=0, indexA=0;
   while(indexF<frente.length && indexC<cauda.length){
       if(frente[indexF]<cauda[indexC]){
             a[indexA]=frente[indexF];indexA++; indexF++;
       }
       else{
            a[indexA]=cauda[indexC];indexA++;indexC++;
           }
   }
   while(indexF<frente.length){/*Se existir, copiar o resto de frente*/
        a[indexA]=frente[indexF];indexF++;indexA++;
   }
   while(indexC<cauda.length){/*Se existir, copiar o resto de cauda*/
        a[indexA]=cauda[indexC];indexC++; indexA++;
   }
}
```

Recorrências básicas

- Programa recursivo que, em cada passo, tem de examinar todos os dados de entrada antes, durante ou depois de os dividir em duas metades
  - **$C(N)= 2C(N/2)+ O(N)$ para $N>=2$ com $C(1)=O(1)$**
  - **Solução: $C(N)$ é aproximadamente $N \times \log(N)$, i.e., $C(N)=O(N \log(N))$**
  - Seja $n=\log N$ (i.e., $N=2^n$).
    Então, $C(2^n) = 2*C(2^{n-1}) + O(2^n)$;

$$
\begin{aligned}
C(2^n)/2^n &= C(2^{n-1})/2^{n-1} + O(1) \\
&= C(2^{n-2})/2^{n-2} + O(1)+O(1) \\
&= \ldots \\
&= C(2^0)/2^0 + O(n) \\
&= O(n) + O(1) \\
&= O(n)
\end{aligned}
$$

Portanto, $C(N)=O(N \log N)$

Cátia Vaz

Master theorem

- **Usado para recorrências do tipo dividir para conquistar** $T(n) = aT(n/b) + f(n)$,

$$a \geq 1, b > 1, \text{ e } f(n) > 0.$$

Case 1: $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$.
Solution: $T(n) = \Theta(n^{\log_b a})$.

Case 2: $f(n) = \Theta(n^{\log_b a})$.
Solution: $T(n) = \Theta(n^{\log_b a} \lg n)$.

Case 3: $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$ and $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n.
Solution: $T(n) = \Theta(f(n))$.

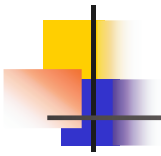
Identidades dos logaritmos e exponenciais

$$
\begin{aligned}
a &= b^{\log_b a} \\
\log_c(ab) &= \log_c a + \log_c b \\
\log_b a^n &= n \log_b a \\
\log_b a &= \frac{\log_c a}{\log_c b} \\
\log_b(1/a) &= -\log_b a \\
\log_b a &= \frac{1}{\log_a b} \\
a^{\log_b c} &= c^{\log_b a}
\end{aligned}
$$

$$
\begin{aligned}
a^{-1} &= 1/a \\
(a^m)^n &= a^{mn} \\
a^m a^n &= a^{m+n}
\end{aligned}
$$

Notações

$$
\begin{aligned}
\lg n &= \log_2 n && \text{(binary logarithm)} \\
\ln n &= \log_e n && \text{(natural logarithm)} \\
\lg^k n &= (\lg n)^k && \text{(exponentiation)} \\
\lg \lg n &= \lg(\lg n) && \text{(composition)}
\end{aligned}
$$



Resolução de Recorrências

- Métodos de resolução de recorrências:
  - Método de subtituição
    - chegar a uma possível solução recorrendo à tecnica de subtituições;
    - Provar por indução que a solução está correcta.
  - Teorema Mestre (Master Theorem)