Projeto e Análise de Algoritmos

Discente: PABLO VINICIOS DA SILVA ARAUJO : 5742297

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 Merge para realizar o Merge desses 3 sub-vetores ordenados em um único ordenado. Mostre o algoritmo recursivo desta versão do Merge-Sort junto com o Merge alterado. Realize a análise de complexidade desse algoritmo e explique se essa versão é mais eficiente do que a versão original.

```
R: funcaoMergeSort(A ,right, left)
        if(left < right)
        meio1 = left - (right - left)/3
        meio2 = left + 2*(right - left)/3

funcaoMergeSort(A,left,meio1)
funcaoMergeSort(A,meio1 + 1,meio2)
funcaoMergeSort(A,meio2+1,right)


Merge(A, left,meio1,meio2,right)
        i = left
        j = meio1 + 1
        k = meio2 + 1
        aux = [ ]
while i <= meio1 AND j <= meio2 AND k <= right:
        if A[i] <= A[j] AND A[i] <= A[k]:
                temp.append(A[i]) i += 1 elif A[j] <= A[i] AND A[j] <= A[k]:
                 temp.append(A[j]) j += 1 else: temp.append(A[k]) k += 1

while i <= meio1 AND j <= meio2:
         if A[i] <= A[j]:
                temp.append(A[i]) i += 1
        else:
         temp.append(A[j]) j += 1 while i <= meio1 AND k <= right:
        if A[i] <= A[k]:
                temp.append(A[i]) i += 1
        else:
                temp.append(A[k]) k += 1 while j <= meio2 AND k <= right:
        if A[j] <= A[k]:
        temp.append(A[j]) j += 1
```

```
else: temp.append(A[k]) k += 1 while i <= meio1:
temp.append(A[i]) i += 1 while j <= meio2:
temp.append(A[j]) j += 1 while k <= right:
temp.append(A[k]) k += 1


for index in range(0, len(temp)): A[left + index] = temp[index]
```

R: Analisando, o mergeSort tem por característica a questão de combinar 3 subvetores de tamanhos, aproximadamente, iguais. O número de níveis de recursão é dado por log n na base 3, devido a 3 divisões por cada nível de chamada.

Além disso, o merge combina os elementos dos 3 vetores, percorrendo-os por completo, dando como característica uma complexidade O(n).

Logo, temos T(n) = n logn na base 3.

Com relação ao mergeSort (versão anterior), a T(n) = n log n na base 2, como nas propriedades logarítmicas, diminuem com bases maiores, logn na base 3 é menor que log n na base 2, logo, a segunda versão é mais eficiente.