03ª)

$$T(n) = \begin{cases} \Theta(1) & n = 1 \\ 3T(\frac{n}{3}) + \Theta(1) + n & n > 1 \end{cases}$$

Teorema mestre.

2º caso: Se $f(n) = \Theta(n^{\log_a b})$, então $T(n) = \Theta(n^{\log_3 b} \log n)$

p/ todo $a \geq 1$ e $b > 1$

constantes.

Prova:

$a = 3$

$b = 3$

$f(n) = n$

$0 \leq c_1 \cdot n^{\log_3 b} \leq n \leq c_2 \cdot n^{\log_3 b}$

$0 \leq n$, como $n > 1$

logo $0 \leq n$ //

$\log_3 3 = 1$ //

$c_1 \cdot n \leq n \quad (\div n)$

$c_1 \leq 1$

$c_1 = 1$ //

$n \leq c_2 \cdot n \quad (\div n)$

$1 \leq c_2$

$c_2 = 2$ //

Logo, $\forall n \geq 2$, $c_1 = 1$ e $c_2 = 2$.

Por sua vez, a complexidade é $\Theta(n \log n)$

Como já é sabido, a versão da merge sort padrão tem complexidade $\Theta(n \log n)$ também.

Concluo que não há diferença entre a merge padrão e a merge dividindo em 3, pois $\Theta(n \log n) = \Theta(n \log n)$.

```cpp
#include <iostream>
#include <vector>
using namespace std;

void Intercala(vector<int>& v, int ini, int mid1, int mid2, int fim){
    //Calculando o tamanho dos 3 subvetores
    int tam1 = mid1 - ini + 1;
    int tam2 = mid2 - mid1;
    int tam3 = fim - mid2;

    vector<int> left (tam1);
    vector<int> medium (tam2);
    vector<int> right (tam3);

    for(int i = 0; i < tam1; i++){
        left[i] = v[i+ini];
    }
    for(int i = 0; i < tam2; i++){
        medium[i] = v[i + mid1 +1];
    }
    for(int i = 0; i < tam3; i++){
        right[i] = v[i + mid2 + 1];
    }
    //Agora eu tenho 3 subvetores e preciso organiza-los em de forma crescente no vetor original

    int i = 0; //Para subvetor left
    int j = 0; //Para o subvetor medium
    int k = 0; //Para o subvetor right
    int l = ini; //Para o vetor original

    while(i < tam1 && j < tam2 && k < tam3){
        if(left[i] <= medium[j] && left[i] <= right[k]){
            v[l] = left[i];
            i++;
        }else if(medium[j] <= left[i] && medium[j] <= right[k]){
            v[l] = medium[j];
            j++;
        }else{
            v[l] = right[k];
            k++;
        }
        l++;
    }

    if(i == tam1 && (j != tam2 || k != tam3)){
        while(j < tam2 && k < tam3){
            if(medium[j] <= right[k]){
                v[l] = medium[j];
                j++;
            }else{
                v[l] = right[k];
                k++;
            }
            l++;
        }
    }else if(j == tam2 && (i != tam1 || k != tam3)){
        while(i < tam1 && k < tam3){
            if(left[i] <= right[k]){
                v[l] = left[i];
                i++;
            }else{
                v[l] = right[k];
                k++;
            }
            l++;
        }
    }else if(k == tam3 && (i != tam1 || j != tam2)){
        while(i < tam1 && j < tam2){
            if(left[i] <= medium[j]){
                v[l] = left[i];
                i++;
            }else{
                v[l] = medium[j];
                j++;
            }
            l++;
        }
    }

    while(i < tam1){
        v[l] = left[i];
        i++;
        l++;
    }
    while(j < tam2){
        v[l] = medium[j];
        j++;
        l++;
    }
    while(k < tam3){
        v[l] = right[k];
        k++;
        l++;
    }

}


void TripleMerge(vector<int>& v, int ini, int fim){
    if(ini < fim){
        int n = fim - ini + 1; //calcula o tamanho do vetor
        int razao = n / 3; //calcula o tamanho que cada subvetor terá

        int mid1 = ini + razao - 1;
        int mid2 = ini + 2 * razao - 1;

        TripleMerge(v,ini,mid1); //Chamada recursiva para o primeiro subvetor
        TripleMerge(v,mid1+1,mid2);//Chamada recursiva para o segundo subvetor
        TripleMerge(v,mid2+1,fim);//Chamada recursiva para o terceiro subvetor

        Intercala(v,ini,mid1,mid2,fim); //Intercalando 3 subvetores em um vetor ordenado

    }
}


int main(){

    vector<int> vetor(9);
    for(int i = vetor.size()-1; i >= 0; i--){
        vetor[i] = i;
        cout << i << " ";
    }

    TripleMerge(vetor,0,vetor.size()-1);

    for(int i = 0; i < vetor.size(); i++)
        cout << vetor[i] << " ";
}
```