Avaliação Assincrona 1 - CANA

José Douglas G. Soares
485347

```
1. Proc acharSegmentos(bdUFC[1..n]){
2.     int i ← 1;
3.     segmentos ← [];
4.     Enquanto(i<=n){
5.         aux_array ← [bdUFC[i]];
6.         Se(i < n e bdUFC[i]+1 == bdUFC[i+1]){
7.             Enquanto(i < n e bdUFC[i]+1 == bdUFC[i+1]){
8.                 aux_array.addFinal(bdUFC[i+1];
9.                 i++;}
10.        }Senão Se(i<n e bdUFC[i]-1 == bdUFC[i+1]){
11.            Enquanto(i<n e bdUFC[i]-1 == bdUFC[i+1]){
12.                aux_array.addFinal(bdUFC[i+1]);
13.                i++;
14.            }
15.            aux_array[1..k] ← aux_array[k..1];
16.        }
17.        i++;
18.        segmentos.addFinal(aux_array);
19.        aux_array ← [];
20.    }
21.    retorna segmentos;
22. }
```

CONTINUA

1. Proc ordenaUFC(bdUFC[1..n]){
2. segmentos ← acharSegmentos(bdUFC[1..n]);
3. mergesort(segmentos, 1, tam(segmentos));
4. int K ← 0
5. PARA i de 1 até tam(segmentos){
6. Para j de 1 até tam(segmentos[i]){
7. bdUFC[K] ← segmentos[i][j];
8. K++;
9. }
10. }

Análise de Complexidade

Procedimento acharSegmentos.

Temos um loop maior na linha 4 que executa aproximadamente n vezes. O ponto chave é ver que não importa em qual condição (linhas 6 e 10) ele entre e nem em qual loop (linhas 7 e 11) ele fique preso, o i é sempre incrementado (linhas 9, 13 e 17). Isso quer dizer que o algoritmo executa aproximadamente n vezes → $\Theta(n)$.

Procedimento ordenaBD

Sabemos que a linha 2 executa em aproximadamente $\Theta(n)$.

Na linha 3, o tempo do mergesort é $n.\log(n)$, mas não temos na máxima $\sqrt{n}$ segmentos, pois foram realizados $\sqrt{n}$ flips. Ou seja, o nosso n é $\sqrt{n}$ e isso implica que a linha 3 executa em $\sqrt{n}.\log n$.

As linhas 5 até 7 iteram o vetor de vetores que possui n elementos inteiros → $O(n)$. Complexidade total = $n + \sqrt{n}.\log(\sqrt{n}) + n$ → $O(n)$.