Tópicos Avançados em Estruturas de Dados

Atividade 4

Bruna Galastri Guedes	18.00189-0
Daniel Ughini Xavier	18.00022-3
Rodolfo Cochi Bezerra	18.00202-0
Vítor Martin Simoni	18.00050-9
Leonardo Cury Haddad	18.00442-3
Leonardo de Barros Rodrigues	18.02401-7

06/04/2020

Questão 1

O algoritmo a ser analisado, "merge sort", é o seguinte:

```
    void merge(int vetor[], int comeco, int meio, int fim) {
    int com1 = comeco, com2 = meio+1, comAux = 0, tam = fim-comeco+1;
    int *vetAux;
    vetAux = (int*)malloc(tam * sizeof(int));

    while(com1 <= meio && com2 <= fim){
        if(vetor[com1] < vetor[com2]) {
            vetAux[comAux] = vetor[com1];
            com1++;
        } else {
            vetAux[comAux] = vetor[com2];
            com2++;
        }
        comAux++;
    }

    while(com1 <= meio){  //Caso ainda haja elementos na primeira metade
        vetAux[comAux] = vetor[com1];
        comAux++;
        com1++;
    }

    while(com2 <= fim) {   //Caso ainda haja elementos na segunda metade
        vetAux[comAux] = vetor[com2];
        comAux++;
        com2++;
    }

    for(comAux = comeco; comAux <= fim; comAux++){    //Move os elementos de volta para o ve
        vetor[comAux] = vetAux[comAux-comeco];
    }

    free(vetAux);
}

void mergeSort(int vetor[], int comeco, int fim){
    if (comeco < fim) {
        int meio = (fim+comeco)/2;

        mergeSort(vetor, comeco, meio);
        mergeSort(vetor, meio+1, fim);
```

```
        merge(vetor, comeco, meio, fim);
    }
}
```

Analisando o algoritmo "merge sort", chegamos a conclusao que sua complexidade para todos os casos é O($n \cdot log_n$). O pior caso para esse algoritmo é quando todos os número do array terão de ser comparados entre si, no mínimo uma vez, mas sua complexidade continua a mesma.