niversidade Federal do Rio Grande do Norte
Estrutura de dados I
Lista de exercícios - 1ª Unidade

IMD

Douglas Felipe de Lima Silva – 20220054131

Questão 01 - Determine o termo dominante e a complexidade Big-O das equações abaixo:

Expression	Dominant term(s)	$O(\ldots)$
$5 + 0.001n^3 + 0.025n$		
$500n + 100n^{1.5} + 50n \log_{10} n$		
$0.3n + 5n^{1.5} + 2.5 \cdot n^{1.75}$		
$n^2 \log_2 n + n(\log_2 n)^2$		
$n \log_3 n + n \log_2 n$		
$3 \log_8 n + \log_2 \log_2 \log_2 n$		
$100n + 0.01n^2$		
$0.01n + 100n^2$		
$2n + n^{0.5} + 0.5n^{1.25}$		
$0.01n \log_2 n + n(\log_2 n)^2$		
$100n \log_3 n + n^3 + 100n$		
$0.003 \log_4 n + \log_2 \log_2 n$		

1 – Termo Dominante = 0,001n^3 / O(...) = n^3

2 - Termo Dominante =100$n^{1.5}$ / O(...) = $n^{1.5}$

3 - Termo Dominante = 2,75$n^{1.75}$ / O(...) = $n^{1.75}$

4 - Termo Dominante = n^2 $\log_2$ n / O(...) = n^2 $\log_2$ n

5 - Termo Dominante = $n \log_3 n$ / O(...) = $n \log_3 n$

6 - Termo Dominante = $3 \log_8 n$ / O(...) = $\log_8 n$

7 - Termo Dominante = $0{,}01n^2$ / O(...) = n^2

8 - Termo Dominante = $100\ n^2$ / O(...) = n^2

9 - Termo Dominante = $0{,}5\ n^{1{,}25}$ / O(...) = $n^{1{,}25}$

10 - Termo Dominante = $n(\log_2 n)^2$ / O(...) = $n(\log_2 n)^2$

11 - Termo Dominante = n^3 / O(...) = n^3

12 - Termo Dominante = $0{,}003 \log_4 n$ / O(...) = $\log_4 n$

Questão 02 - Os algoritmos A e B gastam exatamente $Ta(n) = 0{,}1n^2 \log n$ e $Tb(n) = 2{,}5n^2$ unidades de tempo respectivamente, para um problema de tamanho n. Escolha o algoritmo que tem melhor desempenho na notação Big-O.

Resposta: De acordo com a notação Big-O, levando em consideração que os dois possuem termos n^2, entretanto Ta está sendo multiplicado também por log n, logo o algoritmo e $Tb(n) = 2{,}5n^2$ apresenta melhor desempenho.

Questão 03 - Como a notação Big-O é usada para descrever a complexidade de tempo dos algoritmos?

Resposta: A notação Big-O é descrita como uma função que representa a complexidade do algoritmo de acordo com o seu tempo de execução e a entrada fornecida e determinar seu comportamento assintótico, ou seja, sua aplicação e resultados de tempo de execução no pior caso possível, além dos cálculos de complexidade local que um algoritmo pode apresentar.

Questão 04 - Descreva resumidamente as principais características e o funcionamento dos algoritmos de ordenação bubble sort, selection sort, insertion sort, merge sort e quick sort.

Resposta:

- Bubble Sort: Funciona como um algoritmo de ordenação que em um dado vetor de elementos, compara os adjacentes e faz a troca de posição caso não estejam em ordem, repetindo os passos do algoritmo até que o vetor esteja ordenado. Funciona a partir do início do vetor, fazendo a comparação com o próximo e verificando se estão em ordem, se não reordena e avança para o próximo e repete o processo até que chegue ao final do vetor com ele ordenado. Possui complexidade $O(n^2)$.
- Selection Sort: Funciona como um algoritmo de ordenação que recebe um vetor como input e os organiza em ordem crescente ou decrescente. Ao percorrer todo o vetor, o algoritmo identifica o menor valor e o seleciona para colocar em ordem no vetor não ordenado e segue em loop executando o mesmo processo ignorando o que já foi ordenado, até finalizar com o vetor totalmente ordenado. Possui complexidade $O(n^2)$.
- Insertion Sort: Funciona como um algoritmo de ordenação que recebe um vetor como input e os organiza em ordem crescente ou decrescente. De início o algoritmo percorre o vetor a partir do segundo elemento e realiza comparações com os elementos antecessores, movendo para a direita quando a comparação for falsa e inserindo o elemento na posição correta quando a comparação for verdadeira, até que por fim o vetor esteja completamente ordenado. Possui complexidade $O(n^2)$.
- Merge Sort: O Merge Sort é um algoritmo que funciona através do princípio de “dividir para conquistar”, ou seja, recebe como input um vetor de determinado tamanho, divide em subvetores de tamanho igual e realiza a ordenação de forma recursiva em cada uma das partes, ao fim é realizada a junção (merge) das partes ordenadas. O algoritmo funciona comparando o menor elemento das partes e ordenando. Possui complexidade de tempo $O(n \log n)$ e de espaço $O(n)$.
- Quick Sort: Utiliza a mesma abordagem de “dividir para conquistar” do Merge Sort, mas nesse algoritmo um elemento do vetor (pivô) é selecionado e serve como parâmetro para dividir o vetor em dois, valores maiores que o pivô e menores que o pivô. O algoritmo é aplicado de forma recursiva em cada dos segmentos do vetor, até que esteja

ordenado. Sua complexidade depende e varia de acordo com o pivô escolhido.

Questão 05 - Dado o vetor = {8, 9, 7, 9, 3, 2, 3, 4, 6, 1} explique o passo a passo executado pelo algoritmo bubble sort para ordenar de forma crescente (a resposta pode ser escrita ou através de diagramas).

Resposta: Uma iteração do algoritmo irá fazer uma comparação par a par dos valores no vetor, comparando 8>9 = true a posição dos valores continuaria a mesma, comparando 9>7 = false resulta na troca de posição entre 7 e 9, comparando 9>9 = false resultando numa troca equivalente, comparando 9>3 = false resulta na troca de posição entre 9 e 3 e assim por diante até chegar ao final do vetor e passar para a próxima iteração.

Questão 06 - Descreva resumidamente quais as principais características e diferenças entre os algoritmos de busca binária e de busca linear.

Resposta: A busca linear ou sequencial é um algoritmo simples de busca que funciona percorrendo todos os elementos de um determinado vetor input um vetor ordenado ou desordenado verificando se o valor desejado está presente ou ausente em determinada posição, possui complexidade O(n). Enquanto a busca binária é um algoritmo mais eficiente pois utiliza a estratégia de dividir para conquistar, recebendo como input um vetor já ordenado e dividindo o vetor de forma recursiva até que a posição do valor desejado seja encontrada, possui complexidade O(log n).

Questão 07 - O programa abaixo foi escrito de forma iterativa. Escreva esse algoritmo de maneira recursiva de forma que o resultado final seja o mesmo.

Resposta: De forma recursiva o algoritmo ficaria estruturado da seguinte forma na linguagem C:

```
int fiboRecursivo( int n )
 {
   if ( n <= 1 )
 {
      return n;
  }
else
```

```
{
    return fiboRecursivo(n-1) + fiboRecursivo(n-2);
 }
}
```

```
int fiboIterativo(int n){
    int a = 0, b = 1, c;
    int i = 2;
    if(n == 1){
        return a;
    }else{
        if(n == 2){
          return b;
        }else{
            while(i < n){
                c = a + b;
                a = b;
                b = c;
                i++;
            }
            return c;
        }
    }
}
```

Questão 08 - Explique porque, para problemas muitos grandes, não é recomendável utilizar soluções recursivas.

Resposta: A implementação de soluções recursivas para problemas muito grandes se deve ao fato da chamada recursiva demandar muita memória ao "empilhar" os dados durante sua execução, assim como a lentidão no tempo de execução da chamada recursiva que envolve a declaração e remoção de variáveis por recursão, além do fato de que para problemas muito grandes pode já existir algum algoritmo mais eficiente já implementado.

Questão 09 - Defina o que é caso base (condição de parada) de um algoritmo recursivo.

Resposta: O caso base de um algoritmo recursivo é essencial para a interrupção da recursão para evitar que ela continue indefinidamente gerando erros de Stack overflow ou comportamentos indesejados da função. Ao executar de forma recursiva o algoritmo a função vai empilhando seus valores até que chegue no caso base onde para a execução e retorna o que já foi calculado.

Questão 10 - Dado o algoritmo abaixo, calcule a complexidade local e a complexidade assintótica (O).

Resposta: Sabendo que a complexidade local do algoritmo é determinada pelo numero de operações realizadas pelo mesmo, sabendo que esse algoritmo realiza a busca na matriz por um determinado menor valor encontrado podemos dizer que sua complexidade local é O(n Linhas X n Colunas), além das complexidades de atribuição. Sabendo que a complexidade assintótica é determinada pelo comportamento do algoritmo a medida que seu input tende ao infinito temos também é O(n Linhas X n Colunas) sendo do tipo n^2 pois a quantidade de operações realizadas pelo mesmo cresce ao fazer a multiplicação de n colunas e n linhas de acordo a entrada.

```
int buscaMenor(int **matriz, int linhas, int colunas){
    int menor = matriz[0][0];
    for (int i = 0; i < linhas; i++) {
        for (int j = 0; j < colunas; j++) {
            if (matriz[i][j] < menor) {
                menor = matriz[i][j];
            }
        }
    }
    return menor;
}
```