Universidade Federal Rural do Semi-arido

Semestre: 2024.1

Professora: Rosana Cibely Rego Aula de Estrutura de dados I

Complexidade de algoritmos

Respostas da lista: Complexidade de algoritmos

Exercício 1: Analise de Complexidade de um Algoritmo Simples Considere o seguinte algoritmo:

```
int soma_numeros(int n) {
int soma = 0;
for (int i = 1; i <= n; i++) {
soma += i; }
return soma; }
```

Pergunta: Qual e a complexidade de tempo desse algoritmo? Justi que sua resposta.

Resposta:

A complexidade de tempo deste algoritmo é O(n). O algoritmo possui um único loop que itera n vezes, onde n é o valor de entrada. Dentro do loop, uma operação de adição é realizada. Portanto, o tempo de execução cresce linearmente com o tamanho de n.

Exercício 2: Comparação de Algoritmos Considere dois algoritmos que resolvem o mesmo problema:

Algoritmo A:

```
void algoritmo_a(int n) {
for (int i = 0; i < n; i++) {
 for (int j = 0; j < n; j++) {
printf("%d, %d\n", i, j);
} } }
```

Algoritmo B:

```
void algoritmo_b(int n) {
for (int i = 0; i < n; i++) { printf("%d\n", i); }
 for (int j = 0; j < n; j++) {
printf("%d\n", j);
```

} }

Pergunta: Determine a complexidade de tempo de cada algoritmo e identi que qual e o mais e ciente em termos de complexidade. Justi que sua resposta.

Resposta: A complexidade de tempo do algoritmo A, é $O(n^2)$. O algoritmo possui dois loops aninhados, cada um iterando n vezes. Portanto, o número total de iterações é n * n = n^2, resultando em uma complexidade quadrática.

A complexidade de tempo do algoritmo B, é O(n). O algoritmo possui dois loops que não são aninhados. Cada loop executa n vezes, portanto, o tempo total de execução é linear, ou seja, O(n + n) = O(2n) = O(n).

Nesse caso, O Algoritmo B é mais eficiente em termos de complexidade de tempo, pois tem complexidade linear O(n), enquanto o Algoritmo A tem complexidade quadrática $O(n^2)$.

Exercício 3: Analise de Complexidade Espacial

Considere o algoritmo a seguir:

int soma(int arr[], int n) {

int soma = 0;

for (int i = 0; i < n; i++) {

soma += arr[i]; }

return soma; }

Pergunta: Qual é a complexidade espacial desse algoritmo? Justi que sua resposta.

Resposta: A complexidade espacial deste algoritmo é O(1). O algoritmo utiliza um espaço constante para armazenar a variável soma, independentemente do tamanho do array arr. Portanto, o espaço adicional necessário não cresce com o tamanho da entrada.

Exercício 4: Analise de Algoritmo de Pesquisa

Considere o seguinte algoritmo de busca:

int busca_linear(int arr[], int n, int x) {

for (int i = 0; i < n; i++) {

if (arr[i] == x) {

return i; }

}

return-1; }

Pergunta: Qual é a complexidade de tempo desse algoritmo no pior caso? Como ela se compara com a complexidade de tempo de uma busca binaria em um array ordenado?

Resposta: A complexidade de tempo no pior caso deste algoritmo é O(n). No pior caso, o algoritmo precisará verificar todos os n elementos do array para encontrar o elemento x ou determinar que ele não está presente. Portanto, a complexidade é linear.

Exercício 5:

Complexidade de Ordenação

Considere o algoritmo de ordenação por seleção:

```
void selection_sort(int arr[], int n) {
int i, j, min_idx;
for (i = 0; i < n-1; i++) {
min_idx = i;
for (j = i+1; j < n; j++) {
 if (arr[j] < arr[min_idx]) {
min_idx = j;
} }
int temp = arr[min_idx];
arr[min_idx] = arr[i]; arr[i] = temp;
} }
```

Pergunta: Qual é a complexidade de tempo desse algoritmo? Existem algoritmos de ordenação mais e cientes? Se sim, cite um exemplo.

Resposta: A complexidade de tempo deste algoritmo é $O(n^2)$. O algoritmo de seleção possui dois loops aninhados. O loop externo executa n-1 vezes, e o loop interno executa n-i-1 vezes no pior caso. O produto dessas iterações resulta em uma complexidade de $O(n^2)$.

Sim, existem algoritmos de ordenação mais eficientes, como o Merge Sort e o Quick Sort, ambos com complexidade de tempo O(n log n) no pior caso (no caso do Quick Sort, a complexidade no pior caso é $O(n^2)$, mas em média é O(n log n)).