2. Análise de Desempenho – Respostas

Pergunta 1

Sabemos que a complexidade do algoritmo binarySearch é

Este algoritmo trata-se de um algoritmo recursivo que retorna quantos elementos

de b estão presentes em a.

Por isso temos a seguinte recorrência:

Temos se pois nesse caso só vai fazer uma comparação e retornar zero.

Para temos, à semelhança do caso anterior, o resultante da verificação do , o visto que a complexidade da pesquisa binária aumenta consoante o tamanho do array a, resultante da soma de 1 com a chamada recursiva e a chamada recursiva .

Para esta recorrência não podemos usar o teorema mestre visto que o algoritmo não é do tipo dividir para conquistar, sendo necessário usar o método de substituição neste caso.

Resolvendo pelo método de substituição temos:

...

Pergunta 2

2.1 – Para

Assim como na questão anterior este algoritmo é recursivo e como não é do tipo dividir para conquistar devemos usar o método de substituição na relação de recorrência. Este algoritmo conta o número de vezes que temos de dividir por até o quociente desta divisão ser, aproxidamente por defeito, zero.

Então temos a seguinte relação de recorrência:

Nesta relação de recorrência temos porque neste caso apenas vamos fazer um cálculo e uma comparação. No caso temos dois cálculos e uma comparação em e o custo da próxima chamada .

Resolvendo a relação de recorrência temos:

Como encontra-se uma divisão na próxima chamada recursiva então substituimos por visto que e desta forma podemos simplificar o cálculo da equação.

...

2.2 – Para

Na alínea anterior demonstramos que a complexidade, em função de , para é de , ou seja existe uma correlação entre a base do logaritmo e o valor de e temos:

Ou seja, quando o nosso algoritmo não vai chegar a uma condição de saída porque o quociente de qualquer número a dividir por 1 nunca é zero, e portanto a sua complexidade vai ser infinita (até os recursos do computador se esgotarem).

Logo a complexidade para é

Pergunta 3

Na pergunta 3 é nos apresentado um algoritmo de ordenação do tipo dividir para conquistar e portanto podemos usar o Teorema Mestre para resolver a equação de recorrência.

Ou seja, quando o algoritmo apenas vai realizar operações em tempo constante, não sendo por isso necessário usar uma chamada recursiva, e quando o algoritmo vai usar chamadas recursivas do tipo dividir para conquistar para ordenar todos os sub-arrays (ordenando assim o array).

Para resolver a equação de recorrência, como referido acima, vamos usar o Teorema Meste, e por isso temos que:

Portanto, para esta relação de recorrência aplica-se o primeiro caso do Teorema Mestre onde e , logo:

Então podemos concluir que este algoritmo de ordenação não é eficiente.

Mais tarde, pesquisando a complexidade num motor de busca, averigou-se que o algoritmo em questão trata-se do *Stooge Sort* que começa por ordenar os primeiros do array, de seguida ordena os últimos e no final volta a ordenar os primeiros , sendo que apesar da sua má eficiência é ainda melhor que o algoritmo *Slow Sort*.