INF 213 - Roteiro da Aula Prática

Aluno: Pedro Lucas Santos Ferreira (108196)

Etapa 1

a) Considere o programa "adivinharComplexidade.cpp". Compile-o (sem usar a flag -O3) e teste-o usando a sintaxe "./a.out N" (onde N é o tamanho da entrada).

Qual a complexidade do programa "adivinharComplexidade.cpp" ? Faça testes com N= 5,6,7,8,....13 e tente adivinhar a complexidade dele (não tente entendê-lo ou olhar na internet o que a função next_permutation faz).

A complexidade do programa é O(n!)

b) Considerando o programa "adivinharComplexidade2.cpp", Compile-o (sem usar a flag -O3) e teste-o usando a sintaxe "./a.out N" (onde N é o tamanho da entrada).

Teste o programa com vários valores de N e analise o código.

Por que as funções "dfjkhbjknbjkcjfhui" e "dfjkhbjknbjkcjfhui2", apesar de muito parecidas se comportam de forma tão diferente em relação ao tempo de execução?
Ambas as funções tendem a se estabilizarem com entradas maiores. Entretanto, a operação lógica da segunda função parece ser mais estressante para a máquina, logo seu tempo de execução é maior, já que é preciso realizar uma busca secundária no vetor além dos laço de repetição

Qual complexidade a função "find" (da STL do C++) parece ter? (descubra isso <u>apenas medindo os tempos de execução</u>, não tente entende-la, pesquisar na internet, perguntar ao Chat GPT, etc). <u>Note que queremos a complexidade DA FUNÇÃO find isolada (não do programa todo).</u>
Ao executar o programa, o tempo de execução da função 2 tende a crescer 4 vezes quando a entrada é dobrada. Dessa forma, a complexidade dessa função é O(n^2), mas queremos apenas a complexidade do método "find". Sendo assim, é possível perceber que esse método não interfere "muito" na complexidade da função, pois mesmo com ele a complexidade tende a O(n^2) como dito anteriormente. Logo conclui-se que a complexidade do "find" é menor que O(n^2), podendo ser O(n), O(log n) ou alguma derivação dessas duas.

Qual complexidade a função "log" (do C++) parece ter? Descubra isso apenas medindo os tempos de execução -- dica: teste com números muito maiores (exemplo: 500 milhões, 1 bilhão, 2 bilhões) e apague a chamada à segunda função para conseguir fazer essa medição apenas da primeira (caso contrário

não dará tempo do programa terminar antes do deadline desta prática, que é ainda neste século).
O tempo de execução está dobrando quando a entrada é dobrada. Logo a complexidade é O(n)

Etapa 2

Faça a análise de complexidade das funções presentes no arquivo analise1.cpp (tais funções podem nem compilar ou fazer sentido -- estamos interessados apenas na complexidade dos algoritmos).

Escreva suas respostas como comentários no topo das respectivas funções (veja o exemplo na primeira função de analise1.cpp). Lembre-se de sempre usar a notação “O”, simplificar ao máximo a resposta final (ou seja, em vez de O(3n^4 + n^3) a resposta deverá ser algo como O(n^4)) e utilizar a complexidade mais justa (conforme visto em aula).

Considere sempre o pior caso de cada função. (a não ser que dito o contrário nos comentários) Preste bastante atenção a todas funções!

Lembrem-se sempre de pedir ajuda ao professor se necessário (não fiquem em dúvida sobre a complexidade de alguma função).

Submissao da aula pratica:
A solucao deve ser submetida utilizando o sistema submitty (submitty.dpi.ufv.br). Envie analise1.cpp pelo submitty. Envie também um PDF deste documento após terminar as respostas da Etapa 1 (o nome do arquivo deverá ser roteiro.pdf).