INF 213 - Roteiro da Aula Pratica 1

Arquivos fonte e diagramas utilizados nesta aula:
https://drive.google.com/file/d/1127LdexMcuTr1H0Qc5zJ8WanNwrmLyaa/view?usp=sharing

Como nesta prática há algumas perguntas, faça uma cópia deste documento e acrescente as respostas nela.

Etapa 1

Considere o programa "complexidade1.cpp". Ele possui 5 funções, sendo que cada uma delas realiza uma quantidade diferente de somas.

Estude brevemente o código, compile-o e, a seguir, faça as atividades abaixo (o programa deve ser executado usando a sintaxe "./a.out N", onde N é o "tamanho da entrada") :

a) Meça o tempo de execução para diferentes tamanhos de entrada: 1 , 2, 3 , 4, 10, 11, 12, 13, 14, 50, 100, 500, 1000 (apenas entenda e observe os tempos -- não é preciso anotá-los aqui)

b) Meça os tempos para n=1000 e adicione-os à tabela abaixo. A seguir, tente ADIVINHAR o tempo para n=2000 (adivinhe antes de testar!!!). Finalmente, acrescente na última coluna o tempo (medido pelo programa para n = 2000). (não se preocupe -- você não perderá pontos se errar agora….)

N	1000	2000 (adivinhada)	2000
Funcao0	0.0266510	0.0266510	0.0271070
Funcao que executa n somas	0.0029790	0.0060000	0.0057950
Funcao que executa n^2 somas	2.2832990	8.9000000	9.1515690
Funcao que executa 2n^2 somas	2.5944400	10.0000000	10.4194810
Funcao que executa 4n^2 + n somas	2.6290100	12.0000000	10.7832350

c) O que podemos concluir sobre o tempo nas diferentes funções para valores pequenos de n ? (1, 2, 3, 10, 11, …)
Podemos concluir que entradas com valores pequenos são irrelevantes no quesito de tempo gasto, pois como estamos lidando com uma soma como operação básica, para atingir um tempo relevante é necessário um número de entrada(repetições) maiores.

d) O que podemos concluir sobre as diferenças de tempo para valores maiores ? (na verdade esse experimento simples não é suficiente para garantir que essa conclusão é correta -- mas ela é! Com o tempo veremos que isso é algo que realmente ocorre na prática)
Podemos concluir que quando tratamos de entradas com valores muito grandes o tempo de execução num algoritmos de complexidades $O(n^2)$, $O(n^3)$ tendem a ser mais demorados e para alguns chega a ser inviável a utilização dos mesmos.

e) Na função 1, por que a soma do ct é mais "importante" (crítica para analisar o algoritmo) do que o "i++"?
Nesse caso, apesar de ambas operações gastarem basicamente o mesmo tempo de processamento, a soma do ct é dita mais "importante", pois a variável i será sempre um inteiro nessas situações, o que nos faz julgar a outra operação como mais "importante" no quesito de analisar o seu tempo de processamento, pois uma operação de inteiros é a "menos cara" que outras operações.

Etapa 2

Considere o programa "complexidade2.cpp". Ele possui 4 funções, sendo que cada uma delas realiza uma quantidade diferente de operações.

a) Qual a operação básica em cada um delas?

Função 1	+
Função 2	+
Função 3	+
Função 4	+ e - (creio que não seja * ou / devido às chamadas da função 1)

b) Quantas vezes a operação básica é realizada em cada um ? (no caso da funcao4, pode ser uma resposta aproximada)

Função 1	n vezes
Função 2	$2n^2$ vezes
Função 3	$2n^2 + 2ni^2$ vezes
Função 4	$(4n^2 - 3i + 160)/2$ vezes

c) Meça o tempo de execução de cada função para os diferentes valores de N abaixo (coloque-os na tabela):

N	10	20	50	100	200
funcao1	0.0000810	0.0001810	0.0006020	0.0007760	0.0016620
funcao2	0.0007990	0.0031960	0.0075050	0.0324890	0.1158210
funcao3	0.0013550	0.0071230	0.1143890	0.9132310	7.3234720
funcao4	0.0102250	0.0217760	0.0601650	0.1397520	0.3604490

d) O que podemos concluir sobre os tempos ?
O que podemos concluir ao aumentar os valores de N, é que consequentemente os tempos de processamento do programa aumentam quase que proporcionalmente (depende do processador no momento de execução do programa).

Etapa 3

Compile o programa anterior utilizando a flag “-O3” do g++ (g++ -O3 complexidade2.cpp). Meça novamente os tempos para N = 200.

Essa flag ativa o nível máximo de otimização do compilador. Ela normalmente não reduz a complexidade dos algoritmos, mas acelera de forma considerável o tempo de processamento ao realizar diversos tipos de otimizações (nesta etapa não há nada a ser entregue/respondido)

Etapa 4

Considere o programa complexidade3.cpp . Veja o código-fonte e entenda o que ele faz.

A seguir, compile o programa e teste-o com o arquivo de teste entrada_5.txt (./a.out < entrada_5.txt > saida.txt). Para ver a saída, basta digitar “cat saida.txt” (no Linux)

Concentre-se apenas na função “encontraPosicoes” (nesta prática NÃO altere nada em outras partes do programa -- especialmente a parte que executa sua função 1 milhão de vezes).

Código original

Qual a operação básica da função encontraPosicoes?
A operação básica é a comparação (numeros[j] == i).

Quantas vezes essa operação é executada para uma entrada de tamanho N?

Essa operação é executada n^2 vezes.

Teste o desempenho do programa considerando os arquivos de teste disponibilizados (os números representam o tamanho da entrada)

Primeira melhoria

Note que, quando encontramos a posição de um determinado número “i”, não precisamos continuar procurando por esse número no array (podemos passar para o próximo número). Modifique seu programa utilizando essa estratégia para melhorar sua performance.

Considerando a nova versão do programa:
Quantas vezes a operação básica é executada para uma entrada de tamanho N?
Com essa nova versão do programa a operação básica é executada $n^2/2$ vezes

Teste o desempenho do programa considerando os arquivos de teste disponibilizados (os números representam o tamanho da entrada) Como esse tempo se compara com o obtido na versão original?
Comparando os tempos de execução da original e dessa melhoria podemos perceber que foi diminuído basicamente pela metade, o que já é uma grande diferença para entradas com valores um pouco maiores.

Segunda melhoria

Dada uma entrada pequena (por exemplo, os números 2,0,1,3,4,5) encontre a saída (para o problema tratado neste exercício) **utilizando uma folha de papel**. Pense em uma forma mais eficiente de resolver o problema e modifique “encontraPosicoes” para funcionar utilizando essa nova estratégia. Sua nova função deverá ficar MUITO mais eficiente.

Considerando a nova versão do programa:
Quantas vezes a operação básica é executada para uma entrada de tamanho N?
Com essa nova versão do programa a operação básica será executada N vezes com uma entrada de tamanho N.

Teste o desempenho do programa considerando os arquivos de teste disponibilizados (os números representam o tamanho da entrada). Como esse tempo se compara com o obtido na versão original?
Como nessa nova versão a operação básica só será executada n vezes, logo podemos perceber uma diferença gritante nos tempos de execução, comparando tanto com a original quanto com a primeira melhoria.

Submissao da aula pratica:
A solucao deve ser submetida ate as 18 horas da proxima Segunda-Feira utilizando o sistema submitty (submitty.dpi.ufv.br).

Deverão ser enviados (não envie mais arquivos .cpp):
1) Um PDF deste documento (contendo suas respostas para as perguntas), com nome roteiro.pdf
2) A versão final do programa complexidade3.cpp (ou seja, a versão com a segunda melhoria).