Trabalho 1

Algoritmo e Estruturas de Dados - 4645G-04

Professor: Henry Cabral

Turma: 12

Pedro Henrique Nunes Oliveira

Problema 1:

Máquina de Café, beecrowd | 2670

```cpp
int main(){
    int menorTempo = 10000000;
    int funcionarios[3];
    // Pega valores
    for (int i = 0; i < 3; ++i){
        cin >> funcionarios[i];
    }
    // Testar todos os casos para todos os andares e verificar o menor
    for(int i = 0; i < 3; ++i){
        int tempo = 0;
        for (int j = 0; j < 3;++j){
            tempo += funcionarios[j] * abs(i - j) * 2;
        }
        if (tempo < menorTempo) menorTempo = tempo;
    }
    cout << menorTempo << endl;
    return 0;
}
```

Solução: Para encontrar o melhor andar para posicionar a máquina de café, o programa primeiro lê a quantidade de funcionários em cada andar e calcula, para cada possível posição (andar) da máquina, o tempo total que todos os funcionários gastariam para ir até a máquina e voltar ao seus respectivos andares.

O tempo é calculado fazendo um produto entre número de funcionários por andar, o módulo da diferença desses andares com o andar da máquina e tudo isso multiplicado por 2, visto que é um minuto para ir e um para voltar. A cada tempo calculado, o programa analisa se é menor que o tempo anterior e, se verdadeiro, atualiza esse valor.

Operações: Como o número de andar é fixo (3), o número total de operações e determinado pelos dois laços “for” aninhados, o laço externo percorre cada possível posição da máquina (3 andares) e, para cada posição, o laço interno percorre todos os andares para calcular o tempo total, resultando em 9 operações principais (3*3). Portanto, nesse problema, não existe pior nem melhor caso.

Complexidade: Como o número de andares é fixo, o número de operações não cresce com a entrada, sendo constante. Portanto, o algoritmo tem complexidade ***O(1).*** Se o número de andares fosse variável *(n)*, a complexidade seria $O(n^2)$ pois possui dois laços aninhados.

Problema 2:

Triângulo, beecrowd | 1929

```
int main(){
    int lados[4];
    for(int i = 0; i < 4; i++){
        cin >> lados[i];
    }

    // Testar todas as combinações possíveis de 3 lados;
    bool valido = false;
    for(int i = 0; i < 4; i++){
        for(int j = i + 1; j < 4; j++){
            for(int k = j + 1; k < 4; k++){
                // Aqui verifca todas as possibilidades
                int a = lados[i], b = lados[j], c = lados[k];
                // Se for triangulo valido (soma de 2 lados sempre maior que o
terceiro)
                if(a + b > c && a + c > b && b + c > a){
                    valido = true;
                    break;
                }
            }
        }
    }

    if(valido) cout << "S" << endl;
    else cout << "N" << endl;

    return 0;
}
```

Solução: Para verificar todas as possibilidades entre os 4 lados, buscando um triângulo válido, o programa usa 3 *loops for* os quais começam sempre uma unidade a mais do que o anterior para evitar comparação entre mesmo lado.

No último *loop* é feito uma verificação para ver se o triângulo é válido, isso é, a soma de dois lados quaisquer tem que ser maior que o terceiro lado. Se essa verificação for correta, o programa passa o valor *true* para a flag "válido". Por fim, existe um simples *if/else* para verificar se o programa achou uma possibilidade válida ou não.

Operações: Como o número de varetas é fixo (4), o número total de operações é determinado pelos três laços "for" aninhados, que percorrem todas as combinações possíveis de três varetas dentre quatro. O laço externo percorre cada vareta, o segundo percorre as varetas seguintes, e o terceiro percorre as restantes, resultando em 4 combinações principais ($C(4,3) = 4$). Para cada combinação, são realizadas três somas e três comparações para verificar se é possível formar um triângulo, totalizando 24 operações principais (4×6).

Complexidade: Como o número de varetas é fixo, o número de operações não cresce com a entrada, sendo constante. Portanto, o algoritmo tem complexidade $O(1)$. Se o número de varetas fosse variável (n), a complexidade seria $O(n^3)$, pois possui três laços aninhados para gerar todas as combinações de três elementos, sendo o melhor caso encontrar um triângulo válido na primeiro checagem e o pior caso não encontrar nenhum triângulo válido.

Problema 3:

O maior, beecrowd | 1027

```cpp
int main() {
    int a, b, c;
    cin >> a >> b >> c;

    int maiorAB = (a + b + abs(a - b)) / 2;
    int maior = (maiorAB + c + abs(maiorAB - c)) / 2;

    cout << maior << " eh o maior" << endl;

    return 0;
}
```

Solução: O programa lê três valores inteiros e utiliza uma fórmula matemática para encontrar o maior entre dois deles: (a + b + |a - b|) / 2. Primeiro, calcula o maior entre “a” e “b”, armazenando em “maiorAB”. Em seguida, aplica a mesma fórmula entre “maiorAB” e “c” para obter o maior dos três valores. Por fim, imprime o resultado seguido da mensagem "eh o maior".
Operações: O programa realiza apenas operações aritméticas simples e chamadas à função abs. São feitas duas aplicações da fórmula para encontrar o maior valor, cada uma envolvendo 2 somas, 1 subtração, 1 valor absoluto e 1 divisão. Portanto, o número total de operações principais é constante e pequeno, independentemente dos valores de entrada. Esse problema também não têm pior nem melhor caso.
Complexidade: Como o número de valores é fixo (3), o número de operações não cresce com a entrada, sendo constante. Portanto, o algoritmo tem complexidade O(1). Se o número de valores fosse variável (n), seria necessário adaptar a lógica, mas para este caso permanece O(1).

Conclusão:

Após analisar os 3 problemas, é possível perceber como a complexidade de um programa e o número de operações podem variar conforme cada algoritmo. Em todos os algoritmos acima, o número de operações é fixo para testagem, entretanto, caso esse número fosse variável, os programas poderiam se tornar bem complexos e até lentos de serem executados. Portanto, compreender a ordem de complexidade é fundamental para avaliar o desempenho das possíveis soluções e escolher a melhor abordagem para cada problema.