Universidade Federal de São João del Rei
Departamento de Ciência da Computação
Curso de Ciência da Computação

Roteiro 1

Adélson de Oliveira Carmo Júnior
212050019

1 Encapsulamento

1.1

Código

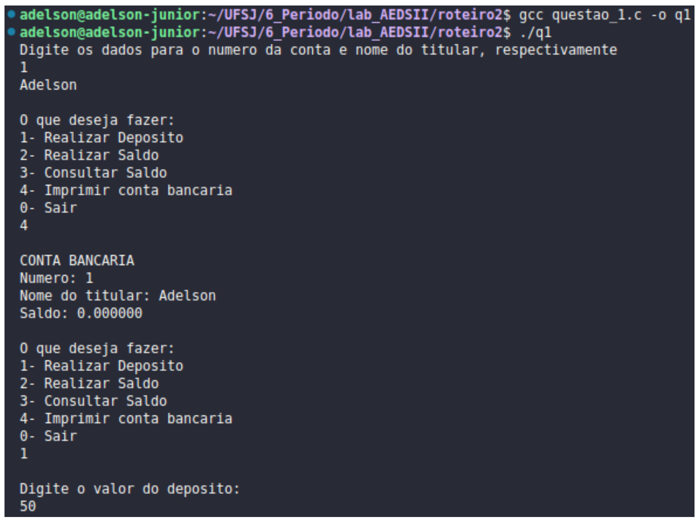
```c
#include <stdio.h>
#include <stdlib.h>

/* Struct da conta bancaria*/
typedef struct conta{
    int numero;
    char* titular;
    double saldo;
}ContaBancaria;


/* Cabecalho */
void criarConta(ContaBancaria* c, int numero, char *titular);
void depositar(ContaBancaria *c, double valor);
void sacar(ContaBancaria *c, double valor);
double consultarSaldo(ContaBancaria *c);
void imprimirInfo(ContaBancaria *c);

/* Funcao que cria uma nova conta bancaria com o numero e titular especificados e inicializa o saldo como zero */
void criarConta(ContaBancaria* c, int numero, char *titular){
    c->numero = numero;
    c->titular = titular;
    c->saldo = 0;
}

/* Funcao que deposita o valor especificado na conta */
void depositar(ContaBancaria *c, double valor){
    c->saldo = c->saldo + valor;
}

/* Funcao que realiza um saque da conta, desde que haja saldo suficiente */
void sacar(ContaBancaria *c, double valor){
    if(c->saldo < valor)
        printf("Saldo insuficiente.\n\n");
    else
        c->saldo = c->saldo - valor;
```

```c
}

/* Funcao que retorna o saldo atual da conta */
double consultarSaldo(ContaBancaria *c){
    return c->saldo;
}

/* Funcao que imprime as informacoes da conta, incluindo n mero, titular e saldo */
void imprimirInfo(ContaBancaria *c){
    printf("\nCONTA BANCARIA\nNumero: %d\nNome do titular: %s\nSaldo: %lf\n",c->numero,c->titular,c->saldo);
}

/* Main */
void main(){
    /* Variaveis que serao usadas */
    int numero, escolha;
    char titular[50];
    double valor;

    /* Interacao com o usuario */
    printf("Digite os dados para o numero da conta e nome do titular, respectivamente\n");
    scanf("%d %s",&numero,titular);

    /* Cria uma conta bancaria e chama a funcao para instanciar seus valores */
    ContaBancaria *c = (ContaBancaria*)malloc(sizeof(ContaBancaria));
    criarConta(c,numero,titular);

    /* Deixa o usuario escolher o que fazer com a conta ate que deseja encerrar parar */
    do{
        /* Interacao com o usuario */
        printf("\nO que deseja fazer:\n1- Realizar Deposito\n2- Realizar Saldo\n3- Consultar Saldo\n4- Imprimir conta bancaria\n0- Sair\n");
        scanf("%d",&escolha);

        /* realiza acao de acordo com a escolha*/
        switch (escolha) {
        case 0:
            break;
        case 1:
            printf("\nDigite o valor do deposito: \n");
            scanf("%lf",&valor);
            depositar(c,valor);
            break;

        case 2:
            printf("\nDigite o valor do saldo: \n");
            scanf("%lf",&valor);
            sacar(c,valor);
            break;

        case 3:
            printf("\nO valor do saldo eh: %f\n", consultarSaldo(c));
            break;

        case 4:
```

```c
            imprimirInfo(c);
            break;

        default:
            printf("Sem gracinhas , digite um numero valido.\n");
            break;
        }
    }while(escolha != 0);
}
```

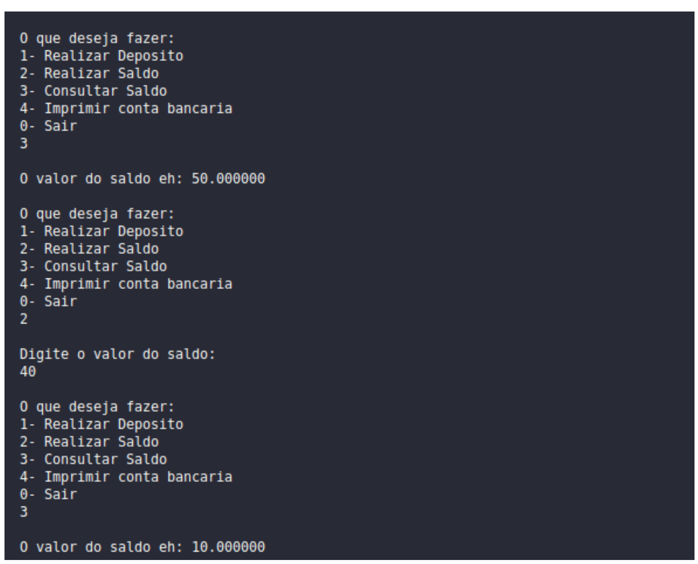
questao_1.c

Saída



```
adelson@adelson-junior:~/UFSJ/6_Periodo/lab_AEDSII/roteiro2$ gcc questao_1.c -o q1
adelson@adelson-junior:~/UFSJ/6_Periodo/lab_AEDSII/roteiro2$ ./q1
Digite os dados para o numero da conta e nome do titular, respectivamente
1
Adelson

O que deseja fazer:
1- Realizar Deposito
2- Realizar Saldo
3- Consultar Saldo
4- Imprimir conta bancaria
0- Sair
4

CONTA BANCARIA
Numero: 1
Nome do titular: Adelson
Saldo: 0.000000

O que deseja fazer:
1- Realizar Deposito
2- Realizar Saldo
3- Consultar Saldo
4- Imprimir conta bancaria
0- Sair
1

Digite o valor do deposito:
50
```

Figura 1: Questão 1.1 - Saida 1



```
O que deseja fazer:
1- Realizar Deposito
2- Realizar Saldo
3- Consultar Saldo
4- Imprimir conta bancaria
0- Sair
3

O valor do saldo eh: 50.000000

O que deseja fazer:
1- Realizar Deposito
2- Realizar Saldo
3- Consultar Saldo
4- Imprimir conta bancaria
0- Sair
2

Digite o valor do saldo:
40

O que deseja fazer:
1- Realizar Deposito
2- Realizar Saldo
3- Consultar Saldo
4- Imprimir conta bancaria
0- Sair
3

O valor do saldo eh: 10.000000
```

Figura 2: Questão 1.1 - Saida 2

2 Análise de Complexidade

2.1

O enunciado diz que deve-se encontra os valores para n para os quais o método da inserção é mais rápido. Para isso, é necessário que se compare as duas funções de complexidade das ordenações, afim encontrar o(s) momento(s) que a da inserção é menor que a da intercalação.

$$8 \cdot n^2 < 64 \cdot n \cdot \ln n$$
$$\cancel{(8 \cdot n)} \cdot n < \cancel{(8 \cdot n)} \cdot 8 \cdot \ln n$$
$$n < 8 \cdot \ln n$$

Agora, com a equação simplificada, basta analisar o gráfico de ambas as partes da igualdade e descobrir seus pontos de intersecção.

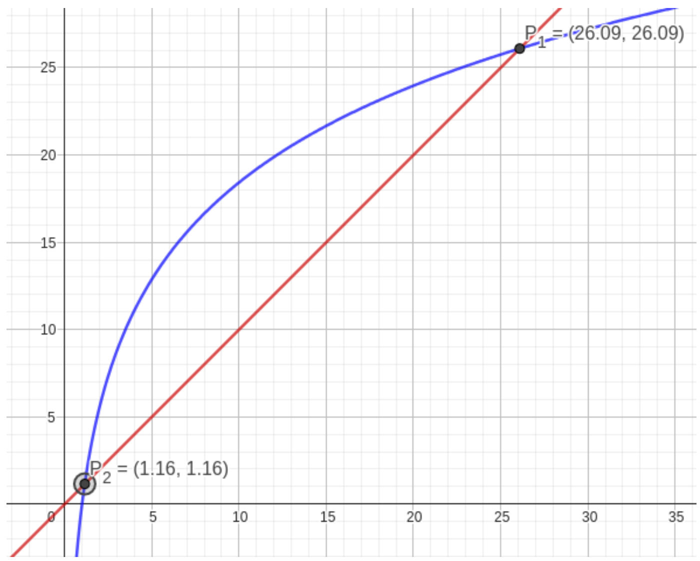


Figura 3: Questão 2.1 - Gráfico

Observe que no intervalo $1.16 < n < 26.09$, a função de ordenacação por intercalação, representado pela cor azul, cresce mais rápido que a função de ordenação por inserção, representada pela cor vermelha. Com isso, é possível afirmar que o método da inserção é mais rápido para $2 < n < 26$, intervalo de números inteiros

2.2

De forma semelhante ao anterior, é necessário que se compare os algortimos entre si afim de determinar quando um supera o outro em eficiência. Assim, escreve-se a seguinte inequação:

$$100 \cdot n^2 < 2^n$$

Devido a complexidade dessa inequação, pode-se traçar o gráfico e, dessa forma, encontrar os pontos de intercecção que entre as funções. Sendo a função de cor vermelha o algortimo de tempo 2^n, e a funcão de cor azul, o algoritmo de tempo $100 \cdot n^2$.

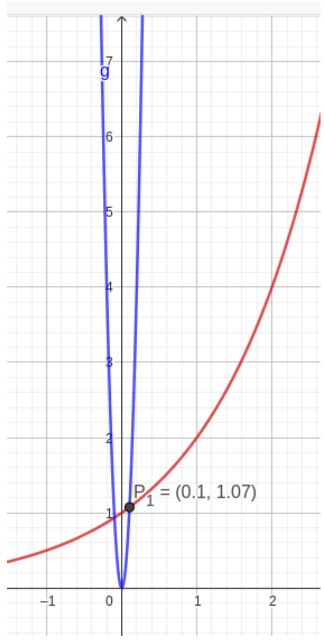


Figura 4: Questão 2.1 - Gráfico aproximando de uma intersecção

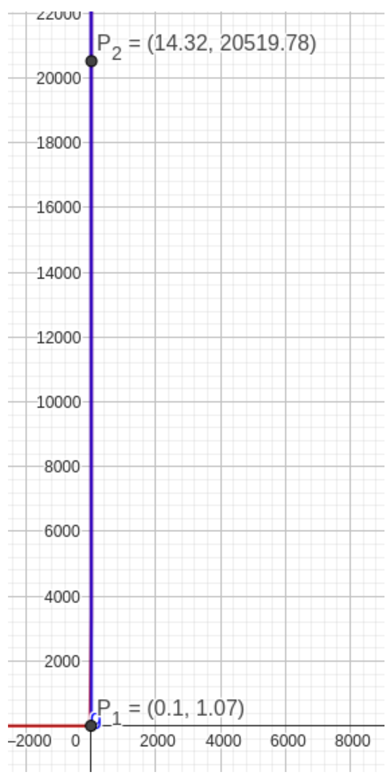


Figura 5: Questão 2.1 - Gráfico com as duas intersecções

Analisando o comportamento das funções no gráfico, é notável que para os intervalos $[0, 0.1]$ e $[14.32, \infty]$ a função azul a que cresce menos, sendo assim, a mais eficiente. Contudo, como não números naturais entre 0 e 0.1, é dito que para valores de n maiores ou iguais a 15, o algoritmo de tempo $100 \cdot n^2$ é mais eficiente que o algortimo de tempo 2^n, .

2.3

Dizer que uma função $g(n)$ é $O(f(n))$ é dizer que $g(n)$ é dominado assintoticamente por $f(n)$. Existe uma constante positiva c que multiplicada a $f(n)$, faz com que ela seja sempre maior que $g(n)$, $g(n) \leq c \cdot f(n)$. Em outras palavras, $g(n)$ nunca será pior, no que diz respeito à eficiencia, que $f(n)$.

2.4

Dizer que uma função $g(n)$ é $\Omega(f(n))$ é dizer que $g(n)$ domina assintoticamente $f(n)$. Existe uma constante positiva c que multiplicada a $f(n)$, faz com que ela seja sempre menor que $g(n)$, $g(n) \geq c \cdot f(n)$. Em outras palavras, $g(n)$ nunca será melhor, no que diz respeito à eficiencia, que $f(n)$.

2.5

Afirmar que o tempo de execução de um algoritmo é no mínimo $O(n^2)$ é errado pelo fato que a notação $O(n)$ denota que a função que domina assintóticamente o algorimo é n^2. Em razão disso, o algoritmo nunca pode ser mais eficiente que essa função e, portanto, ela é o "teto"do tempo de execução desse algoritmo. Contudo, caso deseja-se simbolizar que seu tempo será no mínimp n^2, basta usar $\Omega(n^2)$, já que esta denota que o algoritmo sempre será mais eficiente que essa função.

2.6

Para resolver determinar quando um algoritmo é melhor que outro, será necessário igualar suas funções afim de encontrar os pontos de intersecção:

$$n^2 - n + 500 = 47 \cdot n + 47$$

$$n^2 - 48 \cdot n + 453 = 0$$

Com uma nova equação montada, é possível encontrar seu delta utilizando a fórmula de Bháskara:

$$\Delta = 48^2 - 4 \cdot 1 \cdot 547 = 492$$

Por fim, utilizando dos valores obtidos previamente, encontra-se as raízes:

$$n = \frac{48 \pm \sqrt{492}}{2 \cdot 1}$$

$$n_1 = 12.90$$

$$n_2 = 35.09$$

Com isso, é possível dizer que o intervalor natural que faz o algoritmo A mais eficiente que o B é $[13, 35]$

2.7

Devemos levar em consideração que os três loops *for* geram dois somatórios devido ao fato que usam uma variavel que está sendo incrementada para o funcionamento do *for*. Em razão disso, é possível escrever essa equação da seguinte forma

$$\sum_{i=0}^{n-2}\sum_{i=1}^{n-1} i = \sum_{i=0}^{n-2}(\sum_{i=1}^{n} i - n) = \sum_{i=0}^{n-2}(\frac{n\cdot(n+1)}{2} - n) =$$

$$\sum_{i=0}^{n-2}(\frac{n^2+n-2n}{2}) = \sum_{i=0}^{n-2}(\frac{n\cdot(n-1)}{2}) = \frac{n^3-2^2+n}{2}$$

Com isso é possível afirmar que a operação $(s = 1)$ tem tempo de execução igual a $\frac{n^3-2^2+n}{2}$, ou simplesmente, que ela é $O(n^3)$.

2.8

Nesse algoritmo, ao considerar apenas a operação $(v[i] > MAX)$, leva-se em consideração toda a operação *if*, que provoca um mesmo comportamento assitótico independemente da entrada. Isso ocorre pois toda vez que entrar no *for*, será feita a comparação no *if*, e, portanto, o número de comparações se manterá estático para qualquer entrada n. Sendo assim ele domina e é dominado assintoticamenete por uma função n, logo o algoritmo é $O(n)$ $\Omega(n)$ $\Theta(n)$.