

Universidade Federal de Itajubá

-UNIFEI-

-Itajubá-

-2015-

-Pedro Henrique Dias Scarpioni – 31075

-Prof. Dr. Roberto Claudino – Algoritmos e Grafos

-Bacharelado em Sistemas de Informação

-Instituto de Matemática e Computação – IMC.

-Universidade Federal de Itajubá – UNIFEI.

-Itajubá – MG – Brasil.

=============================================================================

**Índice**

* + Atividade Proposta:...............................................................................................3
    - * Exercício 1:................................................................................................3
      * Exercício 2:................................................................................................3
      * Exercício 3:................................................................................................3
* Soluções das Atividades:.........................................................................................4
  + - * Exercício 1.................................................................................................4
        + Verificação de Corretude:...................................................................4
        + Cálculo de Complexidade:..................................................................4
      * Exercício 2:................................................................................................4
        + Verificação de Corretude:...................................................................4
        + Cálculo de Complexidade:..................................................................4

Melhor Caso:......................................................................4 e 5

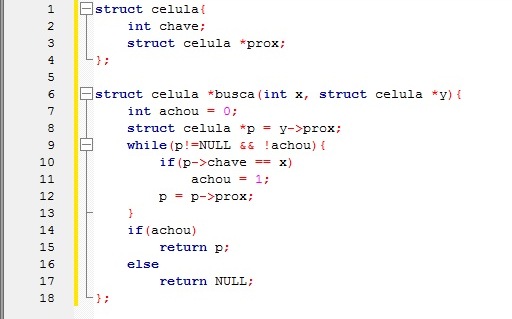
Pior Caso:................................................................................5

* + - * Exercício 3:................................................................................................5
        + Função:...........................................................................................6 e 7
        + Corretude e recorrência:......................................................................7

**Atividade Proposta**

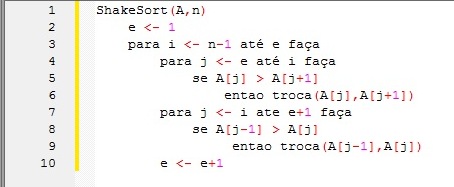
**Exercício 1**

A função abaixo recebe uma lista encadeada com cabeça e um inteiro x, e promete devolver p tal que p -> chave == x ou NULL se tal p não existir. Analise a função verificando sua correção e eficiência.



**Exercício 2**

Prove a correção e determine o tempo de execução, no pior e no melhor caso, para o algoritmo de classificação abaixo:



**Exercício 3**

Projete um algoritmo, prove sua correção e determine a complexidade de tempo e comportamento assintótico, para o problema

Dado um inteiro N, verifique se N é primo.

**Solução Atividades**

**Exercício 1**

**Verificação de Corretude**

As condições que garantem a parada do programa são se não houver mais elementos na lista ou se já foi encontrado o elemento. Dentro do loop, ele verifica se a chave é igual aos elementos da lista, fazendo a variável de controle receber 1, porém na próxima linha ele aponta o p para o próximo mesmo que o elemento já tenha sido encontrado, deveria haver uma condição, onde só mudaria o ponteiro caso a flag, ou seja, o achou fosse igual a zero, pois isso poderá gerar erro no programa, retornando um elemento diferente do que foi buscado ou que não aponte para nenhum espaço de memória.

**Cálculo de Complexidade**

f(n) = 1 + 1 + n \* ( 1 + 1) + 1 + 1

f(n) = 2n + 4

**f(n) = n + 2** → **O(n) = n (Complexidade Linear)**

**Exercício 2**

**Verificação de Corretude**

O algoritmo do *shakesort* está correto pois tem condições de paradas bem definidas, e ele ordena até colocar o maior elemento aonde é o seu lugar e depois em outra iteração ele executa a mesma operação com o menor elemento. Incrementando na segunda fazendo o elemento maior ir em frente na iteração e decrementando na terceira fazendo o menor ir para trás, ou seja, os ordenando em ordem crescente, ordenando em direção ao centro do vetor, pois ao final a variável *e* também é incrementado.

**Calculo de complexidade**

**Melhor caso**

f(n) = 1 + n \* (n \* (1) + n \* (1) + 1)

f(n) = 1 + n (2n + 1)

f(n) = 2n² + n + 1

**Ω(n) = n²**

**Pior caso**

f(n) = 1 + n \* ((1 + 1) + n \*( 1 + 1) + 1)

f(n) = 1 + n \*(4n + 1)

f(n) = 4n² + n + 1

**O(n) = n²**

Tanto no melhor como no pior caso, o algoritmo terá a complexidade O(n²), ou seja, ele terá complexidade quadrática. Porém no melhor caso, as funções de troca não seriam chamadas.

**Exercício 3**

**Função**

#include <stdio.h>

int verifica\_primo(n,meio){

int verifica = 0;

if(n > 1){

if(meio == 1)

return verifica;

else{

if(n % meio == 0){

verifica = 1;

return verifica;

}

verifica\_primo(n,meio - 1);

}

}

else{

if(n < -1){

if(meio == -1)

return verifica;

else{

if(n % meio == 0){

verifica = 1;

return verifica;

}

verifica\_primo(n,meio + 1);

}

}

}

}

///função principal:

int main(){

int n,meio,verifica;

printf("Digite o numero: ");

scanf("%d",&n);

if(n > 1 || n < -1){

meio = n/2;

verifica = verifica\_primo(n,meio);

if(verifica == 0)

printf("%d eh primo",n);

else

printf("%d nao eh primo",n);

}

return 0;

}

**Verificação da Corretude:**

As condições que garantem a parada do algoritmo são as condições da variável *meio* (assim chamada, pois ela começa valendo a metade do número que o usuário irá digitar para verificar se é primo, pois o máximo que algum número pode ser dividido e dar um resultado exato é por sua metade) serem maior que 1, pois todo o número é divisível por 1, mas se um usuário digitar um número negativo a condição é se o número é menor que -1.

E a *flag*, neste algoritmo chamado *verifica* começa recebendo zero, porém se algum número entre a metade do dividendo e 2 for divisível por n, ou seja, o resto da divisão n pelo elemento for igual a zero, o *verifica* irá receber 1, saindo da função, pois o número digitado pelo usuário não é um número primo.

Caso ainda permaneça na função, esta é chamada novamente por recursão, passando o *n* e subtraindo sempre 1 elemento da variável *meio.*

**Recorrência:**

f(n) = (└ n / 2 ┘) - 1 p/ n > 1

f(n) = (└ n / 2 ┘) + 1 p/ n < 1

**f(2) = - 1 \* (n - 1) → f(n) = Θ(n)**

**f(-2) = 1 \* (n - 1) → f(n) = Θ(n)**