P1 Tópicos
Verão IME USP

Oseas Francisco De Lemos André

$\log_{10}^{n}$ é $O(\lg n)$

- prova

Por definição sabemos que $f(n)$ é $O(g(n))$ se existe $c>0$ e um valor n_0 tal que:

$n > n_0 \Rightarrow f(n) \leq c \cdot g(n)$

> de regra de logaritmos sabemos que se temos $\log_{10}^{n}$ e $\log_{2}^{n}$, temos a seguinte relação $\log_{10}^{n} = \frac{\log_2^n}{\log_2^{10}}$, $\log_2^{10} = r$, $r \in \mathbb{R}$

logo temos que $\log_{10}^{n} = \frac{1}{r} \cdot \log_2^n$

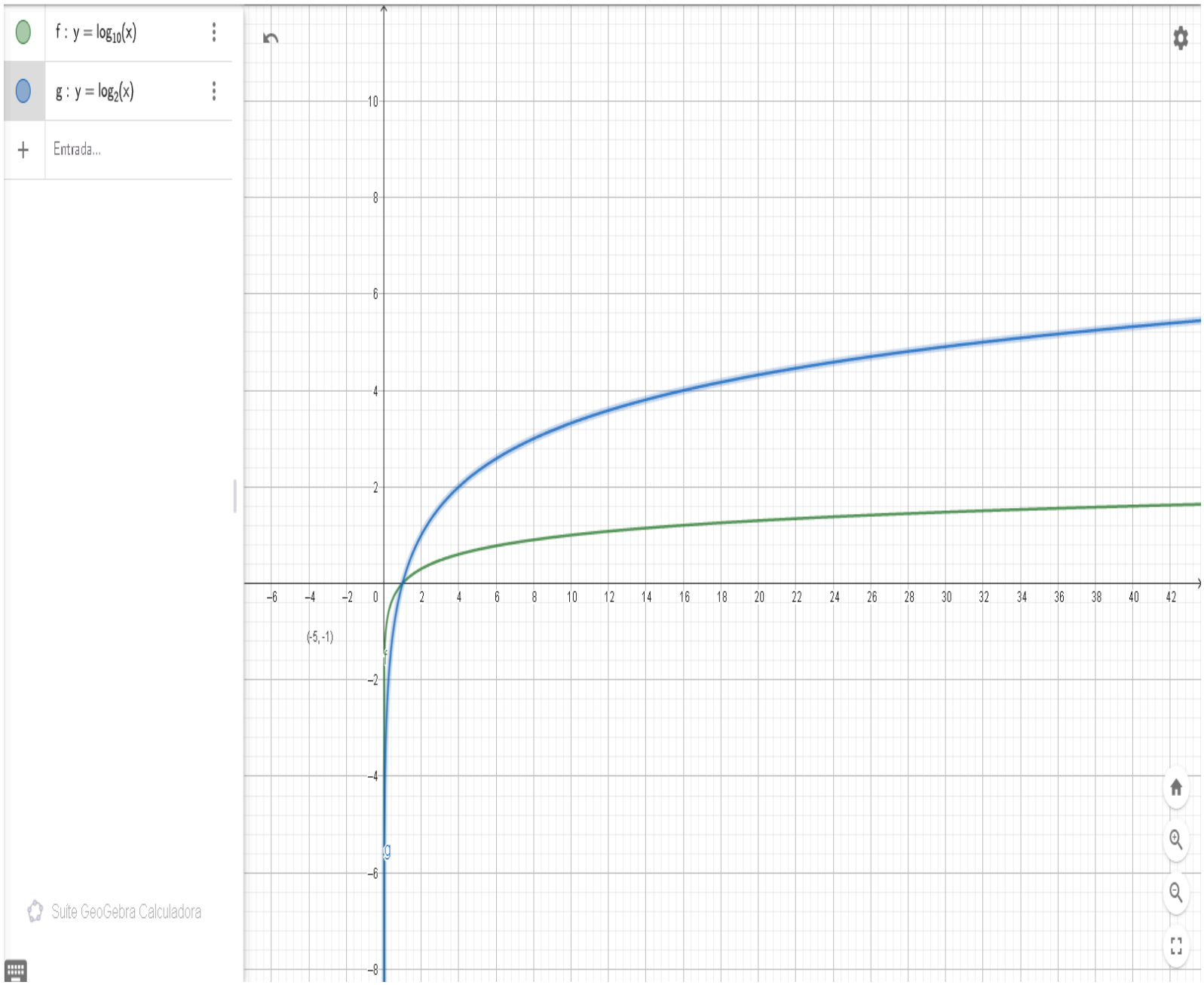
pelo gráfico sabemos que $n_0 = 1$ #

então concluimos que

$n > 1 \Rightarrow \log_{10}^{n} \leq \frac{1}{r} \log_2^n$ #

portanto $\log_{10}^{n}$ é $O(\log_2^n)$

Grafico:



Q2
A função abaixo pode ser testada com o código da questão 3.

```
// Encontrar o nó mais próximo do meio da lista
No* findMiddle(No* cabeca) {
    if (cabeca == NULL) {
        return NULL;
    }

    No* noDePassoUm = cabeca;
    No* noDePassoDois = cabeca;

    while (noDePassoUm != NULL && noDePassoDois->proximo != NULL) {
        noDePassoUm = noDePassoUm->proximo;
        noDePassoDois = noDePassoDois->proximo->proximo;
    }

    return noDePassoDois;
}
```

Q3

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

typedef struct no {
    struct no* anterior;
    int valor;
    struct no *proximo;

}No;

//Função para inserir apenas no inicio da lista
void insereNoInicio(No **cabeca, int valor){
    No *novoNo = (struct no*)malloc(sizeof(No));

    novoNo->valor = valor;
    novoNo->anterior = NULL;
    novoNo->proximo = (*cabeca);

    (*cabeca)->anterior = novoNo;

    (*cabeca) = novoNo;
}

//Função para inserir apenas no final da lista
void insereNoFinal(No *cabeca, int valor){
    No *novoNo = (struct no*)malloc(sizeof(No));

    No *noCorrent = cabeca;
    No *noAnterior = NULL;

    while (noCorrent != NULL)
    {
        noAnterior = noCorrent;
        noCorrent = noCorrent->proximo;
    }

    if(noCorrent == NULL){
        noAnterior->proximo = novoNo;
        novoNo->anterior = noAnterior;
        novoNo->valor = valor;
```

```
        novoNo->proximo = NULL;

    }else{
        noCorrent->proximo = novoNo;
        novoNo->anterior = noCorrent;
        novoNo->valor = valor;
        novoNo->proximo = NULL;

    }
}

//Funão para inserir em alguma posicão especifica da lista
//Considere que o primeiro item da lista fica na posição 0(zero)
void insereNoPosicao(No **cabeca, int valor, unsigned int posicao){
    No *novoNo = (struct no*)malloc(sizeof(No));
    No *noCorrent = *cabeca;
    No *noAnterior = NULL;
    int pos = 0;

    while (noCorrent != NULL && pos != posicao)
    {
        pos +=1;
        noAnterior = noCorrent;
        noCorrent = noAnterior->proximo;
    }

    if(noCorrent == NULL  && posicao>=pos){
        printf("Esta posicao nao existe na lista\n");
        return;
    }

    if(noCorrent != NULL && noAnterior == NULL){
        insereNoInicio(cabeca, valor);
    }else if( noCorrent == NULL && noAnterior != NULL){
        insereNoFinal(*cabeca, valor);
    }else if( noCorrent!=NULL && noAnterior!=NULL)
    {
        novoNo->valor = valor;
        novoNo->anterior = noAnterior;
        novoNo->proximo = noCorrent;

        noAnterior->proximo = novoNo;
        noCorrent->anterior = novoNo;
```

```
    }

}

void printLista(No *cabeca){
    No *aux = cabeca;

    while( aux != NULL){
        printf("%d ", aux->valor);

        aux = aux->proximo;
    }
    printf("\n");
}

void removerNo(No **cabeca, int chave){

    No * aux = *cabeca;
    while( aux != NULL && aux->valor!=chave){
        aux = aux->proximo;
    }

    if( aux != NULL && aux->anterior == NULL)
    {
        No* antigacabeca = (*cabeca);
        (*cabeca) = (*cabeca)->proximo;;
        free( antigacabeca );
    }
    else if( aux != NULL && aux->proximo != NULL)
    {
        aux->anterior->proximo = aux->proximo;
        aux->proximo->anterior = aux->anterior;
        free(aux);
    }else if( aux != NULL && aux->proximo == NULL)
    {
        aux->anterior->proximo = NULL;
        free(aux);
    }
}

int main()
{
    No *cabeca = (struct no*)malloc(sizeof(No));
```

```
    cabeca->valor = 12;
    cabeca->proximo = NULL;

    insereNoInicio(&cabeca, 24);
    insereNoInicio(&cabeca, 36);
    insereNoInicio(&cabeca, 48);
    insereNoInicio(&cabeca, 60);
    insereNoInicio(&cabeca, 72);


    printLista(cabeca);
    insereNoPosicao(&cabeca, -90, 5);
    printLista(cabeca);

    return 0;
}
```

Q4

```c
#include <stdio.h>

#define MAX_SIZE 100

typedef struct {
    int data[MAX_SIZE];
    int top;
} Pilha;

void inicializa(Pilha *pilha) {
    pilha->top = -1;
}

int verificaPilhaVazia(Pilha *pilha) {
    return pilha->top == -1;
}

int VerificaPilhaCheia(Pilha *pilha) {
    return pilha->top == MAX_SIZE - 1;
}

void push(Pilha *pilha, int value) {
    if (!VerificaPilhaCheia(pilha)) {
        pilha->data[++pilha->top] = value;
    } else {
        printf("pilha cheia!\n");
    }
}

int pop(Pilha *pilha) {
    if (!verificaPilhaVazia(pilha)) {
        return pilha->data[pilha->top--];
    } else {
        printf("pilha vazia!\n");
        return -1;
    }
}


int somarElementosDaPilha(Pilha *pilha) {
```

```
    Pilha tempPilha;
    inicializa(&tempPilha);
    int sum = 0;

    // Desempilha todos os elementos, da pilha original
    //empilha em uma pilha temporaria para coseguir manter a ordem no
momento de resturar
    while (!verificaPilhaVazia(pilha)) {
        int value = pop(pilha);
        sum += value;
        push(&tempPilha, value);
    }

    while (!verificaPilhaVazia(&tempPilha)) {
        push(pilha, pop(&tempPilha));
    }

    return sum;
}

int main() {
    Pilha pilha;
    inicializa(&pilha);

    push(&pilha, 12);
    push(&pilha, 24);
    push(&pilha, 36);
    push(&pilha, 48);
    push(&pilha, 60);
    push(&pilha, 72);

    printf("Soma: %d\n", somarElementosDaPilha(&pilha));

    return 0;
}
```

Q5

```c
#include <stdio.h>

#define MAX_SIZE 100

typedef struct {
    int data[MAX_SIZE];
    int top;
} Pilha;

void inicializa(Pilha *pilha) {
    pilha->top = -1;
}

int verificaPilhaVazia(Pilha *pilha) {
    return pilha->top == -1;
}

int VerificaPilhaCheia(Pilha *pilha) {
    return pilha->top == MAX_SIZE - 1;
}

void push(Pilha *pilha, int value) {
    if (!VerificaPilhaCheia(pilha)) {
        pilha->data[++pilha->top] = value;
    } else {
        printf("pilha cheia!\n");
    }
}

int pop(Pilha *pilha) {
    if (!verificaPilhaVazia(pilha)) {
        return pilha->data[pilha->top--];
    } else {
        printf("pilha vazia!\n");
        return -1;
    }
}


//Desempilha os elementos até chave(inclusive) e termina
```

```c
void removerElementoDaPilha(Pilha *pilha, int chave){
    int valor;
    while (!verificaPilhaVazia(pilha))
    {
        valor = pop(pilha);
        //printf("desempilhei: %d\n", valor);
        if( valor == chave ){
            break;
        }
    }

    if( verificaPilhaVazia(pilha) && valor != chave ){
        printf("erro\n");
    }

}

int main() {
    Pilha pilha;
    inicializa(&pilha);

    push(&pilha, 12);
    push(&pilha, 24);
    push(&pilha, 36);
    push(&pilha, 48);
    push(&pilha, 60);
    push(&pilha, 72);

    removerElementoDaPilha(&pilha, 13);

    return 0;
}
```