```c
#include <stdio.h>
#include <stdlib.h>
#include "racionais.h"
#define SEED 0
#define MAX 100

int gerar_aleat(struct racional r[], int n);
void imprime_vet(struct racional r[], int n);
void sem_invalido(struct racional r[], int *n);
void soma_tudo(struct racional r[], struct racional *soma, int n);
void trocar(struct racional *a, struct racional *b);
void bolha_ord(struct racional r[], int n);

int main()
{
    int n;
    struct racional r[MAX];
    struct racional soma;

    srand(SEED);

    do
    {
        scanf("%d", &n);
    } while (n <= 0 || n > 100);

    if (!(gerar_aleat(r, n)))
        return -1;

    imprime_vet(r, n);

    sem_invalido(r, &n);
    imprime_vet(r, n);

    bolha_ord(r, n);
    imprime_vet(r, n);

    soma_tudo(r, &soma, n);
    printf("a soma de todos os elementos eh: ");
    imprime_r(soma);
    printf("\n");

    return 0;
}
/*FIM DO MAIN*/
void soma_tudo(struct racional r[], struct racional *soma, int n)
{
    *soma = r[0];
    for (int i = 1; i < n; i++)
    {
        /*!!! Overflow em alguns casos !!!*/
        if (!(soma_r(*soma, r[i], soma)))
            return;
        if (!(simplifica_r(soma)))
            return;
    }
}

void sem_invalido(struct racional r[], int *n)
{
    for (int i = 0; i < *n; i++)
    {
        if (!(r[i].valido = valido_r(r[i])))
        {
            r[i] = r[*n - 1];
            printf("n: %d", *n);
            (*n)--;
        }
    }
}

int gerar_aleat(struct racional r[], int n)
{
    for (int i = 0; i < n; i++)
    {
        r[i] = sorteia_r(n);
    }
    if (!(simplifica_r(r)))
        return 0;
    return 1;
}
```

Cadê os comentários??? Ao menos para explicar o que cada função faz...

???

não!!!! Não imprima nada nessas funções, ainda mais se for informação de debugging!

??? aplica só em r[0]?

```c
void imprime_vet(struct racional r[], int n)
{
    for (int i = 0; i < n; i++)
    {
        imprime_r(r[i]);
        printf(" ");
    }
    printf("\n");

}

void trocar(struct racional *a, struct racional *b)
{
    struct racional mutz = *a;
    *a = *b;
    *b = mutz;
}

void bolha_ord(struct racional r[], int n)
{
    for (int i = 0; i < n - 1; i++)
    {
        for (int j = 0; j < n - i - 1; j++)
        {
            if (compara_r(r[j], r[j + 1]) > 0)
            {
                trocar(&r[j], &r[j + 1]);
            }
        }
    }
}
```

já reclamei disto...

pior implementação possível, vai executar (n-1)^2 passos, mesmo que o vetor já esteja ordenado...

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

int aleat(int max)
{
    int min;
    if (max > 0)
        min = max * (-1);
    else
    {
        min = max;
        max *= (-1);
    }

    return min + (rand() % (max - min + 1));
}

int mdc(int a, int b)
{
    if (b == 0)
        return a;
    return mdc(b, a % b);
}

int mmc(int a, int b)
{
    return ((a * b) / mdc(a, b));
}

struct racional cria_r(int numerador, int denominador)
{
    struct racional novo;
    novo.num = numerador;
    novo.den = denominador;

    novo.valido = valido_r(novo);

    return novo;
}

struct racional sorteia_r(int max)
{
    struct racional r;
    int denominador, numerador;

    numerador = aleat(max);
    denominador = aleat(max);

    r = cria_r(numerador, denominador);
    r.valido = simplifica_r(&r);
    return r;
}

int numerador_r(struct racional r)
{
    return r.num;
}

int denominador_r(struct racional r)
{
    return r.den;
}

int valido_r(struct racional r)
{
    return r.den;
}

void imprime_r(struct racional r)
{
    if (!valido_r(r))
        printf("INVALIDO");
    else if (r.den == r.num)
        printf("1");
    else if (!r.num)
        printf("0");
    else if (r.den == 1)
        printf("%d ", r.num);
    else
        printf("%d/%d", r.num, r.den);
}
```

```c
int compara_r(struct racional r1, struct racional r2)
{
    float n1, n2;
    n1 = r1.num / (float)r1.den;
    n2 = r2.num / (float)r2.den;
    if (n1 < n2)
        return -1;
    if (n1 > n2)
        return 1;
    return 0;
}

int simplifica_r(struct racional *r)
{
    if (!valido_r(*r))
        return 0;
    int div_comum;
    div_comum = mdc(r->num, r->den);
    r->num /= div_comum;
    r->den /= div_comum;

    if (r->den > 0)
        return 1;
    r->num *= -1;
    r->den *= -1;
    return 1;
}

int soma_r(struct racional r1, struct racional r2, struct racional *r3)
{
    long rnum, rden;

    if (!valido_r(r1))
        return 0;
    if (!valido_r(r2))
        return 0;
    if (!r3)
        return 0;

    rden = mmc(r1.den, r2.den);
    /*separei em duas linhas a mesma conta*/
    rnum = r1.num * (rden / r1.den);
    rnum += r2.num * (rden / r2.den);

    r3->num = rnum;
    r3->den = rden;
    r3->valido = 1;

    // printf("debug: soma entre ");
    // imprime_r(r1);
    // printf(" e ");
    // imprime_r(r2);
    // printf(" gera ");
    // imprime_r(*r3);
    // printf("\n");

    return 1;
}

int subtrai_r(struct racional r1, struct racional r2, struct racional *r3)
{
    long rnum, rden;

    if (!valido_r(r1))
        return 0;
    if (!valido_r(r2))
        return 0;
    if (!r3)
        return 0;

    rden = mmc(r1.den, r2.den);
    /*separei em duas linhas a mesma conta*/
    rnum = r1.num * (rden / r1.den);
    rnum *= r2.num * (rden / r2.den);

    r3->num = rnum;
    r3->den = rden;
    r3->valido = 1;
    return 1;
}
```

foi dito em sala para não fazer isso, pois pode dar resultado errado em alguns casos, por causa da precisão dos floats.

declarações de variáveis devem vir antes dos comandos.

```c
int multiplica_r(struct racional r1, struct racional r2, struct racional *r3)
{
    if (!valido_r(r1))
        return 0;
    if (!valido_r(r2))
        return 0;

    r3->num = r1.num * r2.num;
    r3->den = r1.den * r2.den;
    return 1;
}
void troca_var(struct racional *r)
{
    int aux;
    aux = r->den;
    r->den = r->num;
    r->num = aux;
}

int divide_r(struct racional r1, struct racional r2, struct racional *r3)
{

    if (!valido_r(r1))
        return 0;
    if (!valido_r(r2))
        return 0;

    troca_var(&r2);
    if (!valido_r(r2))
        return 0;

    if (!multiplica_r(r1, r2, r3))
        return 0;
    return 1;
}
```