Raiz quadrada por busca binária:

Pensando da forma mais trivial possível, seria fácil descobrir a raiz, percorrermos todos os números menores que N(podendo ser reduzido pela metade de N), e verificar se a multiplicação de N1 * N1, acarreta na igualdade a n. Como exemplo, um algoritmo em python.

```
n = int(input())
res = 0
for (int i = 0; i <= n; i++):
        if i * i == n:
                res = i
                break
```

Basta pensar um pouco que já conseguimos imaginar os diversos problemas desse algoritmo trivial, além de ser lento pensando em possibilidades enormes de números ele também é falho quando lidamos com números que não possuem raiz exata como 33, nunca chegaríamos em uma possibilidade próxima da resposta já que não teríamos valor que suprisse a igualdade.

Por isso, decidimos usar o algoritmo de **busca binária** baseado em um conceito que é será apresentado de maneira aprofundado em estrutura de dados ao estudar Árvores de Busca Binária, este algoritmo elegante é muito útil quando vamos lidar com um vetor ordenado de números no nosso caso, por exemplo para um número 64, vetor = [1,2,3,4,5,6,7,8...64].

Assim como na árvore binária em nosso algoritmo, a cada passo que percorremos iremos eliminar de cara metade das possibilidades, ou seja, enquanto o algoritmo padrão tem complexidade N(com erros em vários valores), o nosso irá apresentar complexidade log 2 N.

Como exemplo, em um caso mais simples(64):

Mid = 32, como 32 * 32 > 64 end = 31
Mid = 15, 15 * 15 > 64 end = 14
Mid = 7, 7 * 7 < 64 start = 8 end = 14
Mid = (8+14)//2 = 11, 11 * 11 > 64 end = 10
Mid = 9, 9 * 9 > 64, end = 8

Porém, como já mencionamos antes existe o caso de números não que a raiz é fracionária como o numero 10 por exemplo, para solucionar isso trabalhos com um conceito de piso e teto, para controlar

quando devemos parar de verificar e retornar dessa forma um valor aproximado para o usuário, trecho do algoritmo em python.

```
while start <= end:

    mid = (start+end)//2
    # cont = cont + 1

    if  x == mid * mid:
        break
    elif x < mid * mid:
        end = mid - 1
    elif x > mid * mid:
        start = mid + 1

# print("cont: " + str(cont))
return end
```

Fazendo passo a passo:

[0,1,2,3,4,5,6,7,8,9,10] n = 10, onde 1 é o piso, 10 é o teto

10 // 2 → 5(mid)

5 * 5 > 10, logo teremos que nosso teto passará a ser

5 - 1 = 4.

[0,1,2,3,4], continuando:

4 // 2 → 2 (mid)

2 * 2 < 10, logo teremos que nosso teto irá continuar em 4, e nosso piso passa a ser mid + 1, logo 3.

[3,4], temos que;

(3+4) // 2 → 3, seguindo sabemos que 3 * 3 < 10, então nosso piso passa a ser 4.

[4,4], continuando:

(8)//2 → 4(mid)

4 * 4 > 10, portanto nosso teto passará a ser 3, e sairemos do loop pois o piso está maior que o teto.

Ficando assim com a resposta final aproximada armazenada no teto, que é o 3(valor arredondado para raiz de 10).