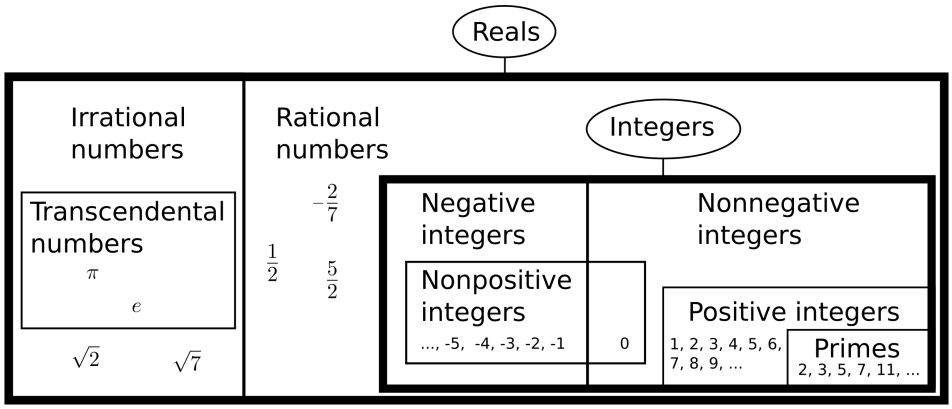
Universidade Federal de Goiás – UFG
Instituto de Informática – INF
Bacharelado em Sistemas de Informação

Algoritmos e Estruturas de Dados 1 – 2018/2

Lista de Exercícios nº 03 – Complexidade Assintótica de Algoritmos

(Turmas: INF0286, INF0061 – Prof. Wanderley de Souza Alencar)

Sumário



1 Enumerando os números racionais

 (++)

Os conjuntos numéricos como $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\Re$ e $\mathbb{C}$ são fundamentais em computação. Infelizmente não é possível que um computador digital os represente de maneira "*perfeita*", tendo em vista que todos eles possuem um número infinito de elementos.

Para alguns conjuntos, como os reais($\Re$), por exemplo, nem mesmo é possível representar, sempre, um único número de maneira perfeita, pois isto exigiria um número infinito de casas decimais de precisão. São os casos de π, $\sqrt{2}$, $\sqrt{3}$, etc.

Considere, neste momento, o conjunto dos números racionais $\mathbb{Q}$, que é formado por todos os números que podem ser expressos sob a forma de uma *fração* envolvendo dois números naturais, ou seja:

$$q = \frac{n}{d}, \text{ com } n \in \mathbb{N}, d \in \mathbb{N}^*$$

O método a seguir apresentado permite que se realize a *enumeração* de todos os números racionais entre 0 e 1, inclusive extremos.

```
rationalsEnumeration()
  integer d, n;
  begin
    for d = 1 to infinity do
        for n = 0 to d do
            if (gcd(n,d) = 1)
               then
                   print (n/d);
            end-if;
        end-for;
    end-for;
  end;
```

onde $gcd(n,d)$ representa o *máximo divisor comum* entre os números naturais n e d.

Por exemplo, a sequência a seguir pode ser gerada:

$$\frac{0}{1}, \frac{1}{1}, \frac{1}{2}, \frac{1}{3}, \frac{2}{3}, \frac{1}{4}, \frac{3}{4}, \frac{1}{5}, \frac{2}{5}, \frac{3}{5}, \ldots$$

Você deverá construir um programa para realizar a *enumeração* de números racionais.

Entrada

A entrada consiste de n linhas, com $n \in \mathbb{N}^*$, sendo que a última linha sempre conterá o valor 0 (zero), indicando o término da entrada. Cada linha apresenta um número natural k, $1 \leq k \leq 12.158.598.919$, que representa a posição do número racional desejado (representado por uma fração) na enumeração anteriormente apresentada.

Saída

A saída será, portanto, expressa por $(n-1)$ linhas, cada uma contendo um número racional expresso por sua fração representativa, ou seja, na forma $\frac{n}{d}$.

Exemplo

Entrada	Saída
1 2 3 12158598919 0	0/1 1/1 1/2 199999/200000

2 Café: Uma bebida muito interessante!

 (++)

Vasiliy gosta de, após realizar um trabalho árduo, descansar. Assim, você pode frequentemente encontrá-lo em alguma lanchonete nas proximidades de seu emprego. Como *todos os programadores*, ele adora a famosa bebida "café" (*coffee*, *café*, *caffè*, *Kaffee*, ...), que pode ser comprada em todas as lanchonetes da cidade.

Sabe-se que o preço de uma xícara café na lanchonete de número natural i $\left(1 \leq i \leq 10^3\right)$ é igual a x_i moedas $\left(1 \leq x_i \leq 10^3\right)$.

Vasiliy planeja comprar uma única xícara diária de sua bebida favorita por q $\left(1 \leq q \leq 10^3\right)$ dias consecutivos. Ele sabe que no i-ésimo dia, ele poderá gastar até m_i $\left(1 \leq m_i \leq 10^9\right)$ moedas.

Agora, para cada um dos dias, ele quer saber em quantas lanchonetes diferentes pode comprar seu adorado cafezinho.

Você, sendo estagiário na mesma empresa em que Vasiliy trabalha, foi incumbido de criar um programa para resolver este problema.

Entrada

A primeira linha da entrada contém um único número natural n – o número de lanchonetes na cidade que vendem a bebida favorita de Vasiliy $\left(1 \leq n \leq 10^3\right)$.

A segunda linha contém os n naturais x_i – preços da bebida, expresso em número de moedas, em cada uma das n lanchonetes da cidade.

A terceira linha contém o número natural q – o número de dias que Vasiliy planeja comprar a bebida.

Em seguida, são listadas q linhas, cada uma contendo um número natural m_i – o número de moedas que Vasiliy pode gastar no i-ésimo dia.

Saída

Imprima q números naturais, um por linha, onde o i-ésimo deles deve ser igual ao número de lanchonetes onde Vasiliy poderá comprar um café no i-ésimo dia.

Exemplo

Entrada	Saída
5 3 10 8 6 11 4 1 10 3 11	0 4 1 5

Observação:

Para o exemplo anterior, no primeiro dia, Vasiliy não poderá comprar a bebida em nenhuma das lanchonetes, pois apenas poderia gastar uma única moeda. Entretanto, em todas as lanchonetes o valor da xícara de café custa mais de uma moeda.
No segundo dia, Vasiliy pode comprar a bebida nas lanchonetes 1, 2, 3 e 4.
No terceiro dia, Vasiliy pode comprar a bebida apenas na lanchonete número 1.
Finalmente, no último dia, Vasiliy pode comprar a bebida em qualquer lanchonete.

3 Monk One: Monk Takes a Walk!

(+)

Today, *Monk* went for a walk in a garden. There are many trees in the garden and each tree has an English alphabet on it.
While *Monk* was walking, he noticed that all trees with vowels on it are not in good state. He decided to take care of them. So, he asked you to tell him the count of such trees in the garden.
Note: The following letters are vowels: 'A', 'E', 'I', 'O', 'U' ,'a','e','i','o' and 'u'.

Input

The first line consists of an natural number T $(1 \leq T \leq 100)$, denoting the number of test cases.
Each test case consists of only one string s of size $|\ s\ |$ $\left(1 \leq |\ s\ | \leq 10^4\right)$, each character of string denoting the alphabet (may be lowercase or uppercase) on a tree in the garden.

Output

For each test case, print the count in a new line.

Example

Entrada	Saída
2 amstrong AlanTuringPrize	2 6

4 Policiais *versus* Ladrões – 1

 (++)

Considere um "*minimundo*" quadrado, formado por $n \times n$ células, com $n \in \mathbb{N}^*$.
Neste "*minimundo*" são aplicadas as seguintes regras:

1. Cada célula pode conter tão somente um policial (P) ou um ladrão (L);
2. Um policial apenas pode prender um ladrão se ambos estiverem na mesma *linha* do "*minimundo*";
3. Um policial não pode prender um ladrão que está a mais de k células de distância dele;
4. Cada policial somente é capaz de prender um único ladrão.

Você deseja elaborar um programa para determinar o número máximo de ladrões ($\max_L$) que podem ser presos para uma determinada configuração (ou estado) do "*minimundo*" fornecido como entrada.

Entrada

A primeira linha contém um número natural T $(1 \leq T \leq 10)$ que representa o número de casos de testes a serem fornecidos em seguida.
Cada grupo de $(n+1)$ linhas seguintes, representando um caso de teste, conterá:

- primeira linha: os números naturais n e k, com $1 \leq n \leq 1000$ e $1 \leq k \leq n^2$;
- próximas n linhas: conterá, cada uma, n caracteres separados por um único espaço em branco entre eles. Cada caractere pode ser um **P** – representando um policial – ou um **L** – indicando um ladrão.

Saída

A saída deverá conter n linhas, cada uma representando a quantidade máxima de ladrões ($\max_L$) que pode ser presa no caso de teste correspondente.

Exemplos

Entrada	Saída
1 3 1 P L P L P L L L P	3

Entrada	Saída
1 5 1 L P L L P L P L L L L L L P L L P L L L P L L L P	7

Entrada	Saída
1 7 2 L L L L L L P L L L L L L L L P L L L P L L L L L L L L L L L P L L L L L L L L P L P P P L L L L	7

Observação:

No primeiro caso de teste, note que há CINCO ladrões (um na 1ª linha, 2 na 2ª e 3ª linhas), como $k = 1$, significa que um policial não pode prender um ladrão que "*está a mais de uma célula de distância*" dele.

Note que o ladrão que está na 3ª linha e 1ª coluna não pode ser preso por nenhum dos policiais. Todos os demais ladrões são podem, potencialmente, serem presos, pois há um policial do lado de cada um deles.

Entretanto, deve-se lembrar que um policial somente pode prender um único ladrão que está na mesma linha que ele, o que faz com que um dos ladrões da 2ª linha não possa ser preso.

Resultado: no máximo TRÊS ladrões podem ser presos.

5 Policiais *versus* Ladrões – 2

 (++)

Considere que os policiais do "*minimundo*" anterior receberam um reforço: cada um agora é acompanhado de um cão da raça *rottweiler* especialmente treinado para capturar ladrões e, por isso, formam a dupla *policial+cão* (D).
As *novas* regras aplicáveis ao "*minimundo*" são as seguintes:

1. Cada célula pode conter tão somente uma dupla (D) ou um ladrão (L);
2. Cada dupla pode prender um ladrão se ambos estiverem na mesma *linha* ou na mesma *coluna* do "*minimundo*";
3. Uma dupla não pode prender um ladrão que está a mais de k células de distância dela;
4. Cada dupla somente é capaz de prender um único ladrão.

Você deseja elaborar um programa para determinar o número máximo de ladrões ($\max_L$) que podem ser presos para uma determinada configuração (ou estado) do "*minimundo*" fornecido como entrada.

Entrada

A primeira linha contém um número natural T $(1 \leq T \leq 10)$ que representa o número de casos de testes a serem fornecidos em seguida.
Cada grupo de $(n+1)$ linhas seguintes, representando um caso de teste, conterá:

- primeira linha: os números naturais n e k, com $1 \leq n \leq 1000$ e $1 \leq k \leq n^2$;
- próximas n linhas: conterá, cada uma, n caracteres separados por um único espaço em branco entre eles. Cada caractere pode ser um **D** – representando uma dupla – ou um **L** – indicando um ladrão.

Saída

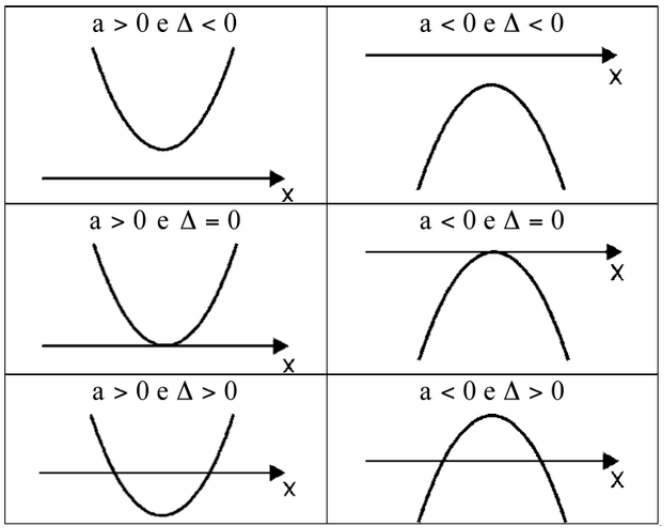
A saída deverá conter n linhas, cada uma representando a quantidade máxima de ladrões ($\max_L$) que pode ser presa no caso de teste correspondente.

Exemplos

Entrada	Saída
1 3 1 D L D L D L L L D	4

Entrada	Saída
1 5 1 L D L L D L D L L L L L L D L L D L L L D L L L D	7

Entrada	Saída
1 7 2 L L L L L L D L L L L L L L L D L L L D L L L L L L L L L L L D L L L L L L L L D L D D D L L L L	8



6 Monk Two: The monk assists a polynomial!

 (++)

Nosso monge (da terceira questão, lembra?), enquanto passeava num parque, ficou perplexo ao encontrar um polinomial que estava no chão, e que tinha a forma $\left(A \cdot x^2 + B \cdot x + C\right)$. O polinomial estava *morrendo* e o monge, sendo sempre atencioso, tentou conversar com ele e revivê-lo. O polinomial disse a ele:
"*Eu já cumpri meu objetivo no mundo, e não necessito mais viver.*
Peço, apenas, que realize um último desejo meu:
Encontre pelo menos um número natural x_0*, de tal maneira que meu valor seja pelo menos k, ou seja, que a relação:*
$A \cdot x_0^2 + B \cdot x_0 + C \geq k$
seja válida."
Auxilie o monge a cumprir, integralmente, o último desejo do polinomial moribundo.

Entrada

A primeira linha contém um número natural T $(1 \leq T \leq 10)$ que representa o número de casos de testes a serem fornecidos em seguida.
As T linhas seguintes conterá, cada uma, os quatro números naturais necessários: A, B, C, e k. Sabe-se que $1 \leq A, B, C \leq 10^5$ e que $1 \leq k \leq 10^{10}$.

Saída

A saída deverá conter T linhas, cada uma representando um natural x_0 que satisfaça ao desejo do polinomial.

Exemplo

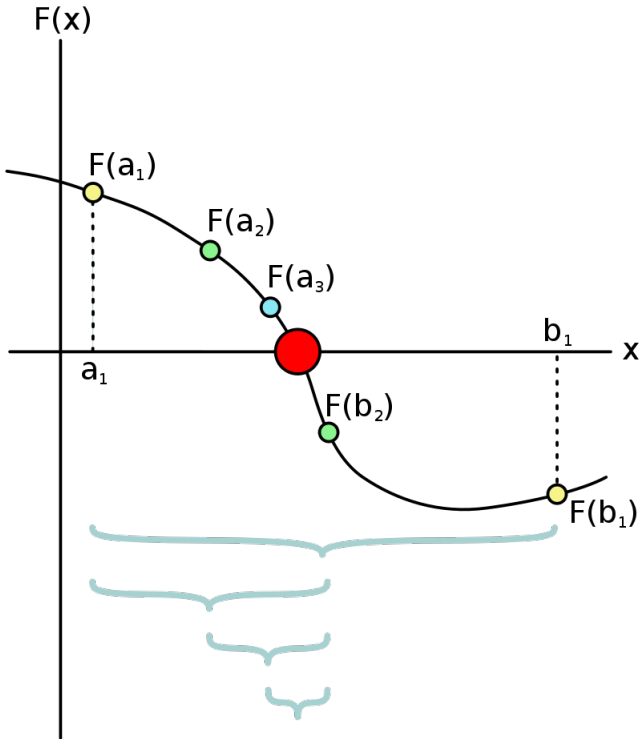
Entrada	Saída
3 3 4 5 6 3 4 5 5 3 4 5 150	1 0 7

Observação:

No exemplo anteriormente apresentado, o número natural x_0 identificado em cada caso é o *menor* possível. Evidentemente outros valores poderiam ser escolhidos.

Nos casos de teste que foram utilizados como *gabarito* no *Sharif Judge System*, a mesma regra foi aplicada. Assim, é possível que um determinado programa, corretamente elaborado, não acerte os casos de teste se não estiver aplicando o mesmo regramento.

O professor corrigirá, manualmente, cada um dos códigos submetidos ao *Sharif Judge System* para esta questão.



7 Método da Bissecção para Determinar as Raízes de Funções Reais

 (++)

Em Matemática, o *Método da Bisseção* (ou *da dicotomia*) é empregado para identificar a *raiz* de uma função contínua $f : [a,b] \rightarrow \Re \mid y = f(x)$ qualquer, desde que $f(a) \cdot f(b) < 0$, ou seja, o produto obtido pela aplicação da função aos extremos (esquerdo e direito) do intervalo em que se deseja verificar a existência da *raiz* seja um número negativo.
Partindo do intervalo inicial $I_0 = [x_a = a, x_b = b]$, o método, repetidamente, executa os passos seguintes:

1. encontra o *ponto médio* do intervalo atual, I_i, que é expresso por $y_i = \frac{(x_a + x_b)}{2}$;
2. bissecciona o intervalo atual, gerando os subintervalos $I_i^1 = [x_a, y_i[$ e $I_i^2 = [y_i, x_b]$;
3. avalia o valor da função f no ponto de bissecção, $f(y_i)$;
4. seleciona o subintervalo que contém a raiz da função (pode ser I_i^1 ou I_i^2);

A execução ocorre até que a raiz, com a precisão desejada, seja encontrada.

A grande qualidade é que este método é *robusto*, mas relativamente lento quando comparado a outros métodos como: (a) *Método de Newton-Raphson*, (b) *Método das Secantes* e (c) *Método da Razão Dourada*, etc.
Para garantir uma determinada precisão nas raízes encontradas, define-se que se $|f(x_i)| \leq \varepsilon$, então x_i é considerado raiz de $f(x)$.

Entrada

Foi designado a você que elabore um programa de computador que seja capaz de aplicar o *Método da Bissecção* para polinômios de grau $n \in \mathbb{N}^*$, ou seja:

$$y = f(x) = a_0 + a_1 \cdot x^1 + a_2 \cdot x^2 + \cdots + a_{n-1} \cdot x^{n-1} + a_n \cdot x^n \quad (1)$$

onde $a_0, a_1, \ldots, a_{n-1}, a_n \in \mathbb{Z}$.
A primeira linha contém o número natural n ($1 \leq n \leq 10^4$), que representa o grau do polinômio a ser fornecido e, separado por um único espaço em branco, o valor de $\varepsilon \in \Re$ (*epsilon*) considerado $\left(0 < \varepsilon < 10^{-6}\right)$.
A segunda linha contém os valores de a e b, com $a, b \in \mathbb{Z}$, delimitadores do domínio de f.
A terceira linha contém o valor dos $(n+1)$ coeficientes, nesta ordem: $a_0, a_1, \ldots, a_{n-1}, a_n$.

Saída

A saída deverá conter as raízes de $f(x)$, em ordem crescente de valores e expressas com seis casas decimais de precisão.

Exemplos

Entrada	Saída
3 0.000001 -10 10 -14 6 -3 1	2.698900

Entrada	Saída
3 0.000001 1 2 -2 -1 0 1	1.521380

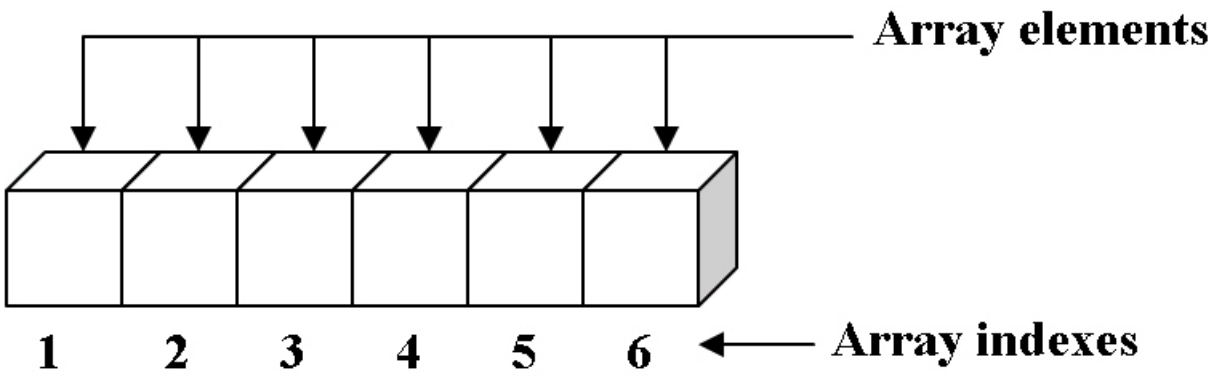
Entrada	Saída
5 0.001 -5 5 -1 0 -6 2 -2 1	2.301000

Observação:

Note que embora os exemplos apenas apresentaram polinômios com uma única raiz real, podem ocorrer aqueles que possuam duas ou mais raízes.

Lembre-se também mesmo que você tenha elaborado um programa correto, devido a divergência de arredondamentos, o *Sharif Judge System* pode indicar casos de testes como incorretos.

O professor corrigirá, manualmente, cada um dos códigos submetidos ao *Sharif Judge System*.



One-dimensional array with six elements

8 Que Número Mínimo é Esse?

 (+++)

Godofredo, um estagiário na área de tecnologia da informação da empresa `TSE - Turing Software Development Enterprise`, recebeu uma tarefa para hoje:
Há uma vetor A que possui n números naturais estritamente positivos, com $n \in \mathbb{N}^*$. Sua tarefa é *atualizar* todos os elementos deste vetor para algum valor mínimo, x_{min}, ou seja, após a atualização teremos que: $A[i] = x_{min}$, com $1 \leq i \leq n$.
Isto deve feito de modo tal que o produto de todos os elementos do *novo vetor* A, que chamaremos de A', seja estritamente maior que o produto de todos os elementos do vetor A original.
Observe que todos os x_{min} devem ser tão pequenos quanto possível, desde que satisfaçam à condição estabelecida.

Entrada

A primeira linha contém um número natural $n \in \mathbb{N}^*$, $1 \leq n \leq 10^5$, que representa o total de elementos do vetor A.
A segunda linha contém os n elementos de A, $1 \leq A[i] \leq 10^5$ separados por espaços em branco.

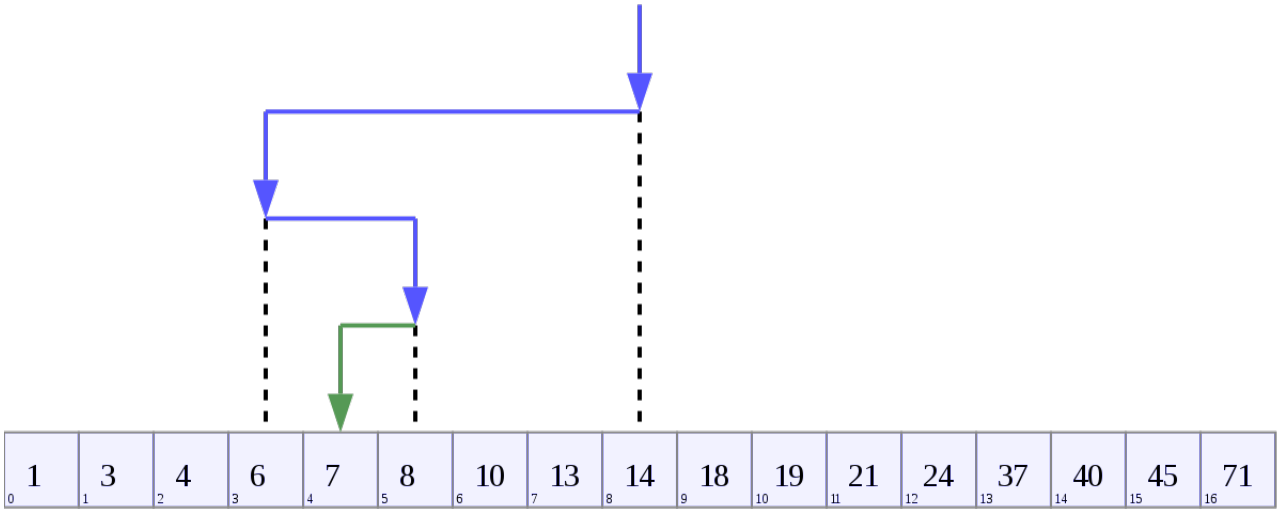
Saída

A saída deverá conter uma única linha, com os *novos valores* de cada dos elementos de A, do primeiro ao n-ésimo, nesta ordem.

Exemplos

Entrada	Saída
5 4 2 1 10 6	4 4 4 4 4

Entrada	Saída
10 1 2 4 8 16 32 64 128 256 512	23 23 23 23 23 23 23 23 23 23



9 Análise Empírica da Complexidade de Tempo – 1

 (++)

Vamos, empiricamente, realizar testes a respeito da *Complexidade de Tempo* de um algoritmo.
Considere que você tenha um vetor V_n de números inteiros, com $1 \leq n \leq 2^{20}$, já ordenados em ordem crescente a partir de sua primeira posição, e sobre o qual você deseja realizar a busca pelo valor k, $0 \leq k \leq \left(2^{32}-1\right)$.
Conceba, e implemente, um algoritmo que seja capaz de realizar a *pesquisa binária* sobre V.

Entrada

Não há entradas fornecidas diretamente pelo usuário, pois todas elas serão geradas, aleatoriamente, pelo próprio programa, da seguinte maneira:

1. Gere o valor de n, com $1 \leq n \leq 2^{20}$;
2. Gere o valor de k, $0 \leq k \leq \left(2^{32}-1\right)$.
3. Gere os valores dos elementos do vetor V, desde que ao final os elementos estejam ordenados de maneira crescente a partir de sua primeira posição. Cada elemento de V deve estar no intervalo inteiro $[0, \left(2^{32}-1\right)]$.

Saída

A saída deverá conter:

1ª linha
o valor de n gerado;

2ª linha
o valor de k gerado;

$\lceil n/10 \rceil$ **próximas linhas**
os valores dos elementos de V, sendo 10 em cada linha e separados por espaços em branco;

penúltima linha

a posição p na qual a primeira ocorrência de k foi encontrada, ou -1 para indicar que k não está presente em V;

última linha

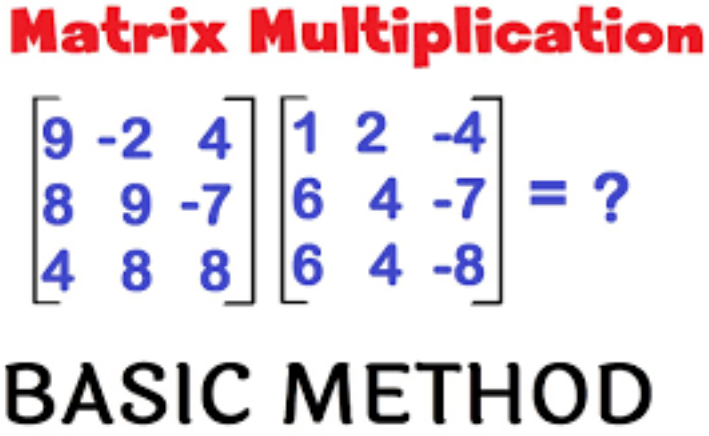
o número de passos que o algoritmo executou até encontrar, ou não, k em V.

Observação:

Para este problema, o professor corrigirá, manualmente, cada um dos códigos submetidos ao *Sharif Judge System*.

10 Análise Empírica da Complexidade de Tempo – 2

 (++)



Vamos, empiricamente, realizar testes a respeito da *Complexidade de Tempo* de um algoritmo. Considere que você tenha duas matrizes quadradas $A_{n\times n}$ e $B_{n\times n}$ de números inteiros, com $1 \leq n \leq 100$. Conceba, e implemente, um algoritmo que seja capaz de calcular a *matriz produto* $A \times B$.

Entrada

Não há entradas fornecidas diretamente pelo usuário, pois todas elas serão geradas, aleatoriamente, pelo próprio programa, da seguinte maneira:

1. Gere o valor de n, com $1 \leq n \leq 1000$;
2. Gere os valores dos elementos das matrizes A e B, sendo que cada elemento x está no intervalo inteiro $[-1000, +1000]$.

Saída

A saída deverá conter $(3n+2)$ linhas, sendo:

1ª linha
o valor de n gerado;

n **próximas linhas**
os valores dos elementos de A naquela linha (separados por espaços em branco);

n **próximas linhas**
os valores dos elementos de B naquela linha (separados por espaços em branco);

n **próximas linhas**
os valores dos elementos de $A \times B$ naquela linha (separados por espaços em branco);

linha $(3n+2)$
o número de passos que o algoritmo executou para gerar a matriz produto.

Observação:

Para este problema, o professor corrigirá, manualmente, cada um dos códigos submetidos ao *Sharif Judge System*.