Universidade Federal do Ceará
Departamento de Computação
Lista de Exercício

Professor Wladimir Araújo Tavares

1. (Jogo do Sapo) Em uma fase do jogo ”Pula Sapo”, seu objetivo é guiar um sapo através de uma sequência de canos de diferentes alturas até que ele chegue, em segurança, ao último cano à direita. No entanto, o sapo só sobrevive se a diferença de altura entre canos consecutivos for, no máximo, igual à altura máxima que ele consegue pular. Se a altura do próximo cano for muito alta, o sapo bate e cai. Se a altura for muito baixa, o sapo não suporta a queda. O sapo sempre começa no primeiro cano, à esquerda da sequência.

   A distância horizontal entre os canos é irrelevante, ou seja, o sapo sempre consegue alcançar o próximo cano com um único pulo, independentemente de sua posição.

   Sua tarefa é escrever um programa que, dado o número de canos e suas respectivas alturas, determine a menor altura do pulo que o sapo precisa para chegar ao último cano sem se machucar.

   Exemplo:

Entrada	**Saída**
10 1 3 6 9 7 2 4 5 8 3	5

   Implemente a sua solução como a seguinte função:

```
int jogo_sapo(int L[], int n) {

}
```

2. (Figurinhas da Copa) No álbum de figurinhas da Copa, cada figurinha possui um número único. Você já comprou várias figurinhas, mas algumas delas podem estar repetidas. Sua tarefa é determinar qual é o maior número de figurinhas repetidas que você possui de um mesmo número.

   Mais formalmente, dada uma lista de figurinhas já compradas, você deve identificar qual é o maior valor de repetição de qualquer figurinha na lista.

Entrada	**Saída**
10 1 2 3 4 2 2 5 3 1 4	3

Implemente a sua solução como a seguinte função:

```
int figurinhas(int L[], int n) {

}
```

Explicação:

A figurinha com o número 2 repete-se 3 vezes. O algoritmo pode fazer até n^2 passagens pelos vetor.

3. (Coca-Cola) Você tem uma quantidade inicial de garrafas de Coca-Cola e pode devolver um número específico de garrafas vazias para receber uma garrafa cheia em troca. O objetivo é calcular o número total de garrafas que você poderá beber, incluindo as obtidas através das trocas de garrafas vazias.

   Por exemplo, suponha que você comprou 10 garrafas de Coca-Cola, e a promoção diz que, a cada 3 garrafas vazias, você pode trocá-las por 1 garrafa cheia. Você começa com 10 garrafas cheias e bebe todas elas, ficando com 10 garrafas vazias. Com essas 10 garrafas vazias, você pode trocá-las por 3 garrafas cheias (10 / 3 = 3 garrafas), sobrando 1 garrafa vazia. Após beber as 3 garrafas cheias, você ficará com 4 garrafas vazias (3 novas + 1 que sobrou). Com 4 garrafas vazias, você pode trocá-las por mais 1 garrafa cheia (4 / 3 = 1 garrafa), sobrando 1 garrafa vazia. Após beber a última garrafa cheia, você ficará com 2 garrafas vazias, o que não é suficiente para mais trocas. O número total de garrafas consumidas será 10 (inicial) + 3 (primeira troca) + 1 (última troca) = 14.

Cheias	Vazias	Total Bebido
10	0	0
0	10	10
3	1	10
0	4	13
1	1	13
0	2	14

Entrada

A entrada é composta por dois inteiros N e M representando o número de garrafas de coca-cola compradas e M representando o número de garrafas vazias que podem ser trocadas por uma garrafa cheia.

Exemplo de Entrada e Saída

Entrada	**Saída**
10 3	14

Implemente a sua solução como a seguinte função:

```
int coca_cola(int N, int promo) {

}
```

4. (Remover Duplicados) Dado um vetor de n números inteiros, o objetivo é remover todas as ocorrências duplicadas, mantendo apenas a primeira ocorrência de cada elemento no vetor. O vetor resultante deve conter apenas elementos únicos, preservando a mesma ordem em que apareceram pela primeira vez no vetor original.

Exemplo de Entrada e Saída

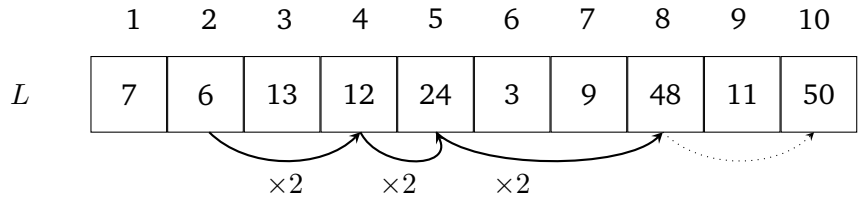
Entrada	**Saída**
7 1 2 2 3 4 3 5	5 1 2 3 4 5

```
int remover_duplicados(int L[], int n) {

}
```

5. (**dobrando os pares**)Faça um programa que recebe uma lista de tamanho N contendo N inteiros. Conte quantos números pares são iguais ao último número par que aparece antes deles (se houver algum).

   Considere a seguinte lista de inteiros:



   No exemplo acima, existem três números pares que são iguais ao dobro do último número par que apareceu antes deles (12, 24 e 48).

Entrada

A primeira linha da entrada contém um inteiro N representando o tamanho da lista. A segunda linha contém N inteiros representando os elementos da lista.

Saída

A saída é composta por uma única contendo a quantidade de números pares que são iguais ao dobro do último par que aparece antes dele.

Exemplos

Entrada	**Saída**
10 7 6 13 12 24 3 9 48 11 50	3

Dica

(a) Encontre o primeiro número par e use um dedo para apontar para essa posição.

(b) Use um outro dedo para percorrendo a lista. Para cada novo número par que encontrar, verifique se ele é igual ao dobro do último número par apontado pelo primeiro dedo.

(c) Se a condição for satisfeita, incremente o contador e atualize a posição apontado pelo primeiro dedo.

6. (**Removendo Duplicadas**)Faça um programa que recebe uma lista ordenada de tamanho N contendo N inteiros que pode conter elementos repetidos. Considere que a lista contém algumas posições vazias no final — (que contém 0's).

   A seguir, um exemplo de uma lista ordenada contendo 10 números inteiros com duas posições vazias no final.

	1	2	3	4	5	6	7	8	9	10
L	1	1	2	3	4	4	4	5	0	0

   A tarefa é remover os elementos duplicados, mantendo a ordem original da lista e preenchendo as posições restantes com zeros no final, sem alterar o tamanho da lista.

   Após a remoção dos elementos duplicados, obtemos uma lista ordenada de tamanho 10 com 5 posições vazias.

	1	2	3	4	5	6	7	8	9	10
L	1	2	3	4	5	0	0	0	0	0

   Note que o tamanho da lista permanece o mesmo, com os zeros preenchendo as posições que ficaram vazias após a remoção dos duplicados.

Entrada

A primeira linha da entrada contém um inteiro N representando o tamanho da lista. A segunda linha contém N inteiros representando os elementos da lista.

Saída

A saída é composta por uma única contendo N inteiros representando os elementos da lista ordenada após a remoção dos elementos duplicados com 0's no final.

Exemplos

Entrada	**Saída**
10 1 1 2 3 4 4 4 5 0 0	1 2 3 4 5 0 0 0 0 0

Dica

(a) O primeiro elemento da lista nunca será removido, pois não há duplicação antes dele.

(b) Use dedo1 para marcar a posição do último elemento único encontrado. Inicialmente, ela aponta para o primeiro elemento da lista.

(c) Use dedo2 para percorrer a lista a partir do segundo elemento. Sempre que o valor apontado por dedo2 for diferente do valor apontado por dedo1, copie esse valor para a posição imediatamente após dedo1 e atualize dedo1.

(d) Continue movendo dedo2 até o final da lista.

(e) Após o processo, preencha as posições restantes da lista com zeros.

7. (**Cometa Halley**)O cometa Halley é um dos cometas de menor período do Sistema Solar, completando uma órbita ao redor do Sol aproximadamente a cada 76 anos. Na última vez em que o cometa foi visível da Terra, em 1986, diversas agências espaciais enviaram sondas para coletar amostras de sua cauda, a fim de confirmar teorias sobre sua composição química.

   Dado um ano atual, o objetivo é determinar o próximo ano em que o cometa Halley será visível da Terra. Caso o ano atual seja um ano de passagem do cometa, considere que o cometa já passou nesse ano e retorne o próximo ano de sua aparição.

   Por exemplo, se o ano atual for 2300, o próximo ano em que o cometa Halley será visível será 2366.

Entrada

A entrada consiste de uma única linha contendo um inteiro A que representa o ano atual.

Saída

Seu programa deve imprimir um único número inteiro, representando o próximo ano em que o cometa Halley será visível da Terra.

Exemplo de Entrada e Saída

Entrada	**Saída**
2300	2366

Explicação

A tabela abaixo ilustra os anos de passagem do cometa Halley mais próximos do ano 2300, para fins de exemplo:

Ano de Passagem	**Ano de passagem após 2300?**
1986	Não
2062	Não
2138	Não
2214	Não
2290	Não
2366	Sim

Como mostra a tabela, o próximo ano de visibilidade do cometa Halley após 2300 será 2366.

8. (**Maior Trecho Contíguo Ordenado Crescente**)Dado um vetor A de tamanho N, dizemos que ele está completamente ordenado de maneira crescente quando $A[i] \leq A[i+1]$ para todo $0 \leq i < N-1$. Um *trecho contíguo* dentro do vetor é formado por elementos consecutivos de A. Seu objetivo é encontrar o maior trecho contíguo dentro do vetor que esteja ordenado de forma crescente.

   Por exemplo, considere o vetor $A = [2, 3, 4, 1, 2, 3, 4, 6]$. Esse vetor possui dois trechos contíguos ordenados de forma crescente:

   - $[2, 3, 4]$, com tamanho 3;
   - $[1, 2, 3, 4, 6]$, com tamanho 5.

   O maior trecho contíguo ordenado é $[1, 2, 3, 4, 6]$, e seu tamanho é 5.

   Dica: Imagine que você começa um trecho crescente a partir do primeiro elemento do vetor, com o tamanho inicial da sequência igual a 1. Agora, posicione o *dedo* sobre o segundo elemento da lista. Se o elemento atual pode ser adicionado à sequência (ou seja, é maior ou igual ao anterior), atualize o tamanho da sequência e o valor da maior sequência encontrada. Se não, inicie uma nova sequência a partir do elemento onde o dedo está posicionado. Continue movendo o dedo até o final do vetor para encontrar o maior trecho.

Entrada

A primeira linha contém um número inteiro N, representando o tamanho do vetor A. A segunda linha contém N inteiros $A[0], A[1], \ldots, A[N-1]$, que são os elementos do vetor.

Saída

A saída deve consistir em uma única linha contendo um número inteiro, representando o tamanho do maior trecho contíguo que está ordenado de forma crescente.

Exemplos de Entrada e Saída

Entrada	Saída
8 2 3 4 1 2 3 4 6	5

Entrada	Saída
8 2 2 3 3 3 1 1 2	5

Explicação

No primeiro exemplo, o vetor $A = [2, 3, 4, 1, 2, 3, 4, 6]$ possui dois trechos ordenados crescentemente: $[2, 3, 4]$ e $[1, 2, 3, 4, 6]$, sendo o maior deles de tamanho 5.

No segundo exemplo, o vetor $A = [2, 2, 3, 3, 3, 1, 1, 2]$ possui trechos ordenados como $[2, 2, 3, 3, 3]$ e $[1, 1, 2]$, e o maior trecho tem tamanho 5.

9. (**separação em ímpares e pares**)Faça um programa que recebe uma lista de tamanho N contendo N inteiros. O objetivo é mover todos os elementos ímpares para o fim da lista e os elementos pares para o início da lista.

   Considere a seguinte lista de inteiros:

	1	2	3	4	5	6	7	8	9	10	11	12
L	11	8	7	12	4	9	10	1	6	2	15	14

   Após o processo de separação, a lista pode ser reorganizada da seguinte forma:

	1	2	3	4	5	6	7	8	9	10	11	12
L	14	8	2	12	4	6	10	1	9	7	15	11

Entrada

A primeira linha da entrada contém um inteiro N representando o tamanho da lista. A segunda linha contém N inteiros representando os elementos da lista.

Saída

A saída deve ser uma lista de N números inteiros, na qual todos os números pares foram movidos para o início da lista e os números ímpares para o final.

Exemplos

Exemplo 1

Entrada	**Saída**
12 11 8 7 12 4 9 10 1 6 2 15 14	14 8 2 12 4 6 10 1 9 7 15 11

Exemplo 2

Entrada	**Saída**
5 5 4 1 8 2	2 4 8 1 5

Dica

Utilize dois dedos:

- Um dedo percorre a lista do início ao final, procurando um número ímpar fora do lugar.
- Outro dedo percorre do final ao início, procurando um número par fora do lugar.

Quando ambos encontram seus respectivos valores, troque-os de lugar. Repita o processo até que os dois dedos se encontrem.

10. (Comedores de Pipoca) A competição de comedores de pipoca consiste em N sacos de pipoca dispostos lado a lado, onde cada saco contém uma quantidade arbitrária de pipocas. A competição é realizada por equipes compostas por C competidores, e cada competidor pode consumir no máximo T pipocas. Para a edição de 2024, a comissão organizadora estabeleceu as seguintes regras:

    - Cada competidor deve consumir uma sequência contígua de sacos de pipocas.
    - Todas as pipocas de um mesmo saco devem ser consumidas por um único competidor.

    Por exemplo, considere $N = 5$, $C = 3$, $T = 13$, e a quantidade de pipocas nos sacos dada por $[5, 8, 3, 10, 7]$. Uma distribuição possível seria:

    - O primeiro competidor consome os sacos 1 e 2 (5+8 = 13 pipocas).
    - O segundo competidor consome os sacos 3 e 4 (3+10 = 13 pipocas).
    - O terceiro competidor consome o saco 5. (7 pipocas)

    Assim, o total de pipocas consumidas pelos três competidores seria 33 pipocas.

    Dica: Use um vetor totalPipocas para armazenar o consumo de cada competidor e uma variável competidorAtual para acompanhar quem está consumindo. Para cada saco, se $totalPipocas[competidorAtual] + sacos[i] <= T$, adicione as pipocas do saco ao total do competidor. Caso contrário, passe para o próximo competidor (competidorAtual++) e adicione as pipocas do saco a esse competidor. Lembre-se de checar se o competidorAtual não é igual a N.

    Entrada

    A primeira linha da entrada é composta por três números inteiros: N, representando o número de sacos de pipoca; C, representando o número de competidores; e T, representando o número máximo de pipocas que cada competidor pode consumir. A segunda linha é composta por N números inteiros, indicando a quantidade de pipocas em cada saco.

    Saída

    A saída é composta por um único número inteiro, representando a quantidade total de pipocas consumidas pela equipe de competidores.

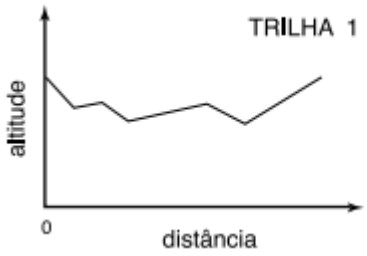
    Exemplo de Entrada e Saída:

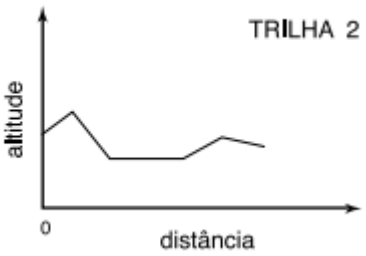
Entrada	**Saída**
5 3 13 5 8 3 10 7	33

Entrada	**Saída**
5 3 12 5 8 3 10 7	26

Entrada	**Saída**
3 3 4 1 1 5	2

11. (Trilha) Nos finais de semana, Fernanda costuma fazer longas caminhadas pelas belas trilhas que atravessam as matas vizinhas à sua cidade. Recentemente, Fernanda adquiriu um aparelho de GPS (Sistema de Posicionamento Global) e, com ele, mapeou as trilhas mais bonitas da região. Ela programou o GPS para registrar, em intervalos regulares, a altitude do ponto atual durante o trajeto. Dessa forma, após percorrer as trilhas com seu GPS, Fernanda consegue informações que permitem, por exemplo, a produção de gráficos, como os abaixo:





Fernanda já possui os dados de uma trilha que percorreu e deseja calcular o esforço de subida dessa trilha. O esforço de uma trilha é determinado pelo somatório dos esforços de cada trecho, onde o esforço de um trecho é proporcional ao desnível positivo entre os pontos consecutivos.

Por exemplo, considere uma trilha formada por 4 pontos com altitudes [498, 500, 498, 498]. O esforço dessa trilha será 2, pois há um desnível de 2 unidades entre o primeiro e o segundo ponto.

Entrada

A entrada é composta por duas linhas. A primeira linha contém um inteiro N, que indica o número de pontos da trilha. A segunda linha contém N inteiros representando as altitudes de cada ponto.

Saída

A saída consiste em uma única linha contendo o valor do esforço total da trilha.

Exemplo de Entrada e Saída:

Exemplo 1

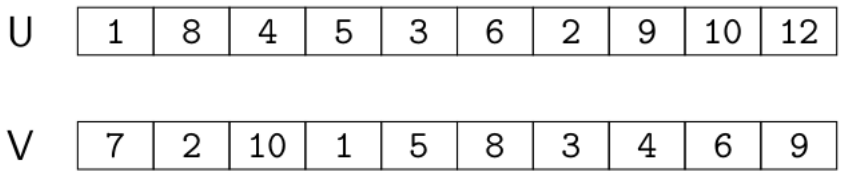
Entrada	**Saída**
4 498 500 498 498	2

Exemplo 2

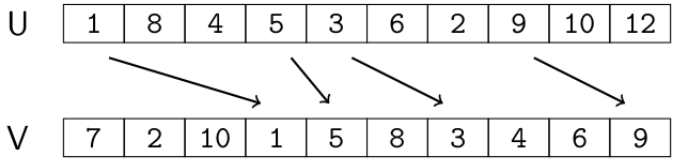
Entrada	**Saída**
4 1 10 6 8	11

12. Sejam duas listas U e V com tamanhos N e M, respectivamente, contendo números inteiros. Sua tarefa é verificar se todos os elementos ímpares da lista U estão presentes na lista V, e se, para cada um desses elementos, a sua posição em V é posterior à posição correspondente em U. Observe que podem existir mais elementos ímpares em V do que em U, mas o importante é garantir que os ímpares de U estão todos presentes e em ordem posterior.

    Considere o seguinte exemplo:



    Neste exemplo, todos os elementos ímpares de U aparecem em V, e cada um deles aparece mais adiante em V do que na lista U, portanto as listas passam no teste.




```
=> Sim
```

    Dica: Uma forma de resolver esse problema é percorrer a lista U da esquerda para a direita, verificando os números ímpares. Para cada ímpar encontrado em U, procure o mesmo número em V, garantindo que ele aparece mais adiante do que a sua posição em U.

    Entrada

    A primeira linha da entrada é composta por dois inteiros N e M representando o tamanho de U e V. A segunda linha da entrada é composta pelos elementos de U. A terceira linha da entrada é composta pelas linhas de V.

    Saída A saida é composta por uma única linha contendo ”Yes”se todos os elementos se todos os elementos ímpares de U estão em V e se cada um deles aparece mais adiante em V do que eles aparecem em U. Caso contrário, escreva ”No”.

Entrada	**Saída**
10 10 1 8 4 5 3 6 2 9 10 12 7 2 10 1 5 8 3 4 6 9	Yes

13. (**Martelo da Força**) O martelo de força é um equipamento utilizado para medir a força aplicada por um participante. Um prêmio é concedido ao participante dependendo da faixa de força que ele alcança. Considere que existem três faixas de força, descritas da seguinte forma:

Força (F)	**Premiação**
$F < 3$	1 ponto
$3 \leq F < 5$	4 pontos
$F \geq 5$	7 pontos

Note que as três faixas de premiação podem ser definidas por dois valores-limite $[3, 5]$, e a premiação correspondente a cada faixa é dada por três valores $[1, 4, 7]$.

Agora, considere que temos N faixas de premiação, definidas por $N - 1$ valores-limite, e N valores de premiação. Além disso, o participante utiliza o martelo da força M vezes, e a força alcançada em cada tentativa é fornecida. Sua tarefa é calcular a premiação total obtida pelo participante.

Entrada:

- Um número inteiro N representando a quantidade de faixas de premiação.
- Uma sequência de $N - 1$ números inteiros que definem os limites das faixas de força.
- Uma sequência de N números inteiros que representam a premiação para cada faixa.
- Um número inteiro M representando o número de tentativas.
- Uma sequência de M números inteiros, onde cada número $F_i (1 \leq F_1 \leq 1000000)$ representa a força alcançada em uma tentativa.

Saída: Imprima a soma total das premiações obtidas pelo participante.

Exemplo de Entrada e Saída:

Entrada	**Saída**
3 3 5 1 4 7 3 2 4 3	9

Explicação: Para a entrada fornecida, o participante realiza três tentativas com as forças 2, 4 e 3. Na primeira tentativa (força 2), ele recebe 1 ponto. Na segunda (força 4), recebe 4 pontos. Na terceira (força 3), também recebe 4 pontos. Portanto, a pontuação total é 1 + 4 + 4 = 9.

14. (Pão a Metro) O pão a metro é um tipo de sanduíche que pode ser comprado em diferentes tamanhos. Você está organizando uma festa e pretende encomendar sanduíches de N empresas, cada uma fornecendo um sanduíche de metragem M_i (em centímetros). Seu objetivo é cortar todos os pães em fatias de tamanho fixo K e distribuí-las entre os convidados.

    Dado dois inteiros N (número de sanduíches) e K (tamanho da fatia em centímetros), seguidos por N valores representando as metragens dos sanduíches, calcule o número total de fatias de tamanho K que podem ser obtidas.

    Entrada:

    - Um inteiro N representando o número de sanduíches.
    - Um inteiro K representando o tamanho desejado de cada fatia (em centímetros).
    - Uma sequência de N inteiros, onde cada inteiro M_i representa o tamanho de um sanduíche (em centímetros).

    Saída: Imprima o número total de fatias de tamanho K que podem ser cortadas a partir dos N sanduíches.

    Exemplo de Entrada e Saída:

Entrada	**Saída**
4 57 120 89 230 177	10

    Explicação: Para a entrada fornecida, temos os seguintes sanduíches com metragens: 120 cm, 89 cm, 230 cm e 177 cm. Desejamos cortar fatias de tamanho 57 cm.

    - O primeiro sanduíche (120 cm) gera 2 fatias.
    - O segundo sanduíche (89 cm) gera 1 fatia.
    - O terceiro sanduíche (230 cm) gera 4 fatias.
    - O quarto sanduíche (177 cm) gera 3 fatias.

    No total, são obtidas $2 + 1 + 4 + 3 = 10$ fatias.

15. (Pão a Metro 2) O pão a metro é um tipo de sanduíche que pode ser comprado em diferentes tamanhos. Você está organizando uma festa e pretende encomendar sanduíches de N empresas, cada uma fornecendo um sanduíche de metragem M_i (em centímetros). Seu objetivo é cortar todos os pães em fatias de tamanho fixo K e distribuí-las entre M convidados.

    Dado dois inteiros N (número de sanduíches) e M número de convidados, seguidos por N valores representando as metragens dos sanduíches, calcule o tamanho máximo da fatia para obter M pedaços com o mesmo tamanho.

    Entrada:

    - Um inteiro N representando o número de sanduíches.
    - Um inteiro M representando o número de convidados.
    - Uma sequência de N inteiros, onde cada inteiro M_i representa o tamanho de um sanduíche (em centímetros).

    Saída: Imprima o tamanho inteiro máximo da fatia que pode ser cortada.

    Exemplo de Entrada e Saída:

Entrada	**Saída**
4 10 120 89 230 177	57

    Explicação: Podemos retirar M fatias iguais de tamanho máximo 57, pois assim conseguimos 2 fatias no primeiro pão, 1 no segundo, 4 no terceiro e 3 no quarto, totalizando as 10 fatias necessárias. Se tentarmos cortar fatias de tamanho 58, só será possível obter 3 fatias do terceiro pão, totalizando 9 e, portanto, 57 é realmente o melhor que podemos obter. Note que não podemos usar duas ou mais fatias menores de diferentes pães para formarmos uma fatia do tamanho selecionado. (ficaria muito deselegante dar um lanche recortado às pessoas).

16. (O Sapinho no Poço) Um sapo começa no fundo de um poço com profundidade P centímetros. A cada salto, o sapo sobe S centímetros, mas, durante a recuperação para o próximo salto, ele escorrega E centímetros. A cada novo salto, o sapo se cansa, o que reduz a altura de seus saltos em 10 cm a cada tentativa subsequente. Se o sapo ficar abaixo da água (altura menor ou igual a zero) em qualquer momento, ele morrerá afogado.

    Sua tarefa é simular o movimento do sapo e determinar se ele conseguirá escapar do poço ou se morrerá afogado.

    Regras:

    - O sapo começa no fundo do poço ($H = 0$).
    - A cada salto, o sapo sobe S centímetros, mas o próximo salto é reduzido em 10 cm.
    - Após cada salto, ele escorrega E centímetros, exceto se o salto final fizer com que ele escape do poço (ou seja, se a altura acumulada com o salto for maior ou igual à profundidade P).
    - Se em qualquer momento o sapo ficar com altura menor ou igual a zero, ele afoga.

    Entrada: Três inteiros P, S, e E representando, respectivamente, a profundidade do poço, a altura inicial dos saltos, e a distância que o sapo escorrega.

    Saída: Determine se o sapo escapa do poço ou se morre afogado. Se ele escapa, imprima a quantidade de saltos necessários. Caso contrário, imprima "Afogado".

    Exemplo 1: Considere P = 800, S = 300 e E = 100:

H (Altura Atual)	S (Salto Atual)	$H + S \geq P$ (Escapa?)	Posição Final
0	300	Não	200
200	290	Não	390
390	280	Não	570
570	270	Sim	4

    Neste caso, o sapo consegue escapar após 4 saltos.

    Exemplo 2: Considere P = 100, S = 50 e E = 30:

H (Altura Atual)	S (Salto Atual)	$H + S \geq P$ (Escapa?)	Posição Final
0	50	Não	20
20	40	Não	30
30	30	Não	30
30	20	Não	20
20	10	Não	0
0	0	Não	Afogado

    Neste caso, o sapo morre afogado após não conseguir continuar subindo.

Entrada	**Saída**
800 300 100	4

Entrada	**Saída**
800 300 100	Afogado

17. (Sapinho Matemático) Um sapo começa no fundo de um poço com profundidade P centímetros. A cada salto, o sapo sobe S centímetros, mas, durante a recuperação para o próximo salto, ele escorrega E centímetros. A cada novo salto, o sapo se cansa, o que reduz a altura de seus saltos em 10 cm a cada tentativa subsequente. Se o sapo ficar abaixo da água (altura menor ou igual a zero) em qualquer momento, ele morrerá afogado.

    Sua tarefa é simular o movimento do sapo e determinar se ele conseguirá escapar do poço ou se morrerá afogado.

    Regras:

    - O sapo começa no fundo do poço ($H = 0$).
    - A cada salto, o sapo sobe S centímetros, mas o próximo salto é reduzido em 10 cm.
    - Após cada salto, ele escorrega E centímetros, exceto se o salto final fizer com que ele escape do poço (ou seja, se a altura acumulada com o salto for maior ou igual à profundidade P).
    - Se em qualquer momento o sapo ficar com altura menor ou igual a zero, ele afoga.

    Entrada: Dois inteiros P e E representando, respectivamente, a profundidade do poço, a altura inicial dos saltos, e a distância que o sapo escorrega.

    Saída: Determine o valor mínimo do salto para que o sapo saia do poço.

Exemplo de Entrada e Saída

Entrada	**Saída**
200 10	67

18. (Maior Elemento de U) Dadas duas listas de inteiros L e U, de tamanhos N e M, respectivamente, seu objetivo é encontrar o maior elemento de U que também aparece na lista L.

Exemplo de Entrada e Saída

Entrada	Saída
10 10 5 8 1 10 13 14 7 11 4 6 9 3 6 12 8 13 4 3 5 1	13

Implemente a seguinte função:

```
//devolva o maior elemento de U que está em L
int maiorU(int L[], int N, int U[], int M){
}
```

Busca Binária

19. Seja $X[0 \ldots n-1]$ um vetor ordenado de inteiros com n números distintos. Adapte o algoritmo de busca binária para determinar se existe um índice i tal que $X[i] = i$. Por exemplo, considerando $X = [-1, 1, 4, 5, 6, 7, 8]$, temos que $X[1] = 1$.
20. Considere dois vetores ordenados: $X[0 \ldots n]$ com $n+1$ elementos e $Y[0 \ldots n-1]$ com n elementos. O vetor X contém um único elemento extra em relação a Y. Adapte o algoritmo de busca binária para encontrar esse elemento extra. Por exemplo, considerando $X = [2, 4, 6, 8, 9, 10, 12]$ e $Y = [2, 4, 6, 8, 10, 12]$, o elemento extra é 9.
21.
22. Seja $X[0 \ldots n-1]$ um vetor que representa uma progressão aritmética, na qual exatamente um elemento está faltando. O elemento ausente será sempre um elemento do meio, ou seja, nem o primeiro nem o último da progressão. Adapte o algoritmo de busca binária para encontrar o elemento ausente. Por exemplo, considerando $X = [3, 7, 11, 15, 23, 27]$, o elemento 19 está faltando. Dica: Use o primeiro e o último elemento para calcular a razão da progressão aritmética.

Ordenação

23. (TapiocaSort)Dona Inês preparou uma pilha de n tapiocas, identificadas pelos tamanhos $p_1, p_2, \ldots, p_n$, e deseja ordená-la em ordem crescente de tamanho, com a menor tapioca no topo da pilha. Para isso, ela pode realizar apenas uma operação: inserir uma espátula entre duas tapiocas na pilha e virar todas as tapiocas acima da espátula. Considerando $p1$ como sendo o topo da pilha de tapiocas.

    Por exemplo, considere a pilha inicial $[2, 5, 3, 1, 7, 6, 4]$. Se Dona Inês inserir a espátula logo abaixo da tapioca de tamanho 1, a pilha resultante após o virar será $[1, 3, 5, 2, 7, 6, 4]$.

    Faça um programa que resolva o problema de Dona Inês utilizando exclusivamente a operação de virar para ordenar as tapiocas. Note que Dona Inês pode inspecionar a pilha de tapiocas antes de decidir a posição onde inserir a espátula. Considere que o valor em p_i representa o tamanho da tapioca.