Algoritmos e Programação I: Aula 11*

Faculdade de Computação
Universidade Federal de Mato Grosso do Sul
79070-900 Campo Grande, MS
https://ava.ufms.br

Sumário

1 Solucionando Problemas com Estrutura de Seleção

1.1 Calculando adiantamento de salários

Uma empresa tem funcionários cujos salários são mensais ou calculados pelo número de horas trabalhadas. Toda semana a empresa faz um adiantamento de salário aos funcionários. Se o número do funcionário for maior ou igual a 1000 o cálculo do adiantamento é dado pelo valor do pagamento do mensal dividido por 4. Se o número do empregado for menor que 1000 o cálculo do valor do adiantamento é dado pelo número de horas trabalhadas vezes o valor pago por cada hora trabalhada. Todas as horas trabalhadas acima de 40 horas recebem um adicional de hora extra de 50%. Projete e implemente um programa que leia o número do funcionário, e o salário mensal ou horas trabalhadas e o valor pago por hora trabalhada, calcule e imprima o valor do adiantamento semanal de cada funcionário.

Exemplo 1

```
# formato da entrada
1042 # número do funcionário
10000.00 # valor do salário mensal

# formato da saída
Funcionário no.: 1042
```

*Este material é para o uso exclusivo da disciplina de Algoritmos e Programação I da FACOM/UFMS e utiliza as referências bibliográficas da disciplina.

```
Valor do adiantamento semanal: R$ 2500.00
```

Exemplo 2

```
# formato da entrada
543 # número do funcionário
30.00 # valor da hora trabalhada
50 # números de horas trabalhadas

# formato da saída
Funcionário no.: 543
Valor do adiantamento semanal: R$ 1650.00
```

A descrição do problema fica:

```
# Este programa lê um número de um funcionário, e o salário mensal ou
# horas trabalhadas e o valor pago por hora trabalhada. Toda semana a
# empresa faz um adiantamento de salário aos funcionários. Se o número do
# funcionário for maior ou igual a 1000 o adiantamento é calculado por
# meio do valor do pagamento mensal dividido por 4. Se o número do
# empregado for menor que 1000 o valor do adiantamento é calculado por meio
# do número de horas trabalhadas vezes o valor pago por cada hora
# trabalhada. Todas as horas trabalhadas acima de 40 horas tem um
# adicional de hora extra de 50%. O programa calcula e imprime o valor do
# adiantamento semanal de cada funcionário.
```

Vamos agora descrever as especificações da entrada e saída, variáveis utilizadas e as pré e pós condições.

Entrada	**Saída**
um número inteiro representando o número do funcionário e três números reais representado o salário mensal, o valor da hora e o número de horas trabalhadas	valor do adiantamento semanal

e necessitamos das variáveis do tipo `int` para armazenar o valor do número do funcionário (`num_funcionario`) e do tipo `float` para armazenar o salário mensal (`salario_mensal`), o valor da hora trabalhada (`valor_hora`) e o número de horas trabalhadas (`numero_horas`). Quando da implementação do algoritmo pode surgir a necessidade da utilizarmos variáveis auxiliares para armazenar valores intermediários.

```
# descrição das variáveis utilizadas
# int num_funcionario
# float salario_mensal, valor_hora, numero_horas
# float valor_horasextras, adiantamento_semanal
```

Exemplo 1

```
# formato da entrada
1042 # número do funcionário
10000.00 # valor do salário mensal

# estado das variáveis após a leitura dos dados
num_funcionario == 1042
salario_mensal == 10000.00

adiantamento_semanal = salario_mensal / 4.0
adiantamento_semanal = 2500.00

# formato da saída
Funcionário no.: 1042
Valor do adiantamento semanal: R$ 2500.00
```

Exemplo 2

```
# formato da entrada
543 # número do funcionário
30.00 # valor da hora trabalhada
50 # números de horas trabalhadas

# estado das variáveis após a leitura dos dados
num_funcionário == 543
valor_hora == 30.00
numero_horas == 50

adiantamento_semanal = numero_horas * valor_hora
adiantamento_semanal = 50 * 30.00
adiantamento_semanal = 1500.00
valor_horasextras = (numero_horas - 40) * 0.5 * valor_hora
valor_horasextras = 10 * 0.5*30.00
valor_horasextras = 150.00
adiantamento_semanal = adiantamento_semanal + valor_horasextras
adiantamento_semanal = 1500.00 + 150.00

# formato da saída
Funcionário no.: 543
Valor do adiantamento semanal: R$ 1650.00
```

Como vimos acima parte da entrada depende o valor no número do funcionário, e o mesmo ocorre com o cálculo do adiantamento semanal. As pré e pós-condições ficam:

```
# pré: num_funcionario (salario_mensal or valor_hora numero_horas)

# pós: num_funcionario, adiantamento_semanal
```

Os passos do Algoritmo são:

```
# Algoritmo: Empresa
# Passo 1. Leia as informações do salário dos funcionários
# Passo 1.1. Leia o número do funcionário
# Passo 1.2. Se funcionário mensalista (> 1000) leia salário mensal
# Passo 1.3. Caso contrário leia o valor da hora e número de horas
# Passo 2. Calcule o valor do adiantamento semanal
# Passo 2.1. Calcule o adiantamento para funcionários mensalistas
# Passo 2.2. Calcule o adiantamento para funcionários horistas
# Passo 2.2.1. Calcule o valor das horas extras
# Passo 3. Imprima os resultados
# fim do algoritmo
```

No Passo 2. temos duas possibilidades para o cálculo do valor do adiantamento semanal. Uma para os funcionários com salário mensal e outra para funcionários que recebem por hora trabalhada. No caso dos funcionários que recebem por hora trabalhada, deve ser levado em conta o adicional para as horas extras. No Passo 2.2. já foi calculado o valor do adiantamento levando em consideração o número de horas trabalhadas. Falta verificar se o funcionário trabalhou acima de 40 horas. Neste caso, deve ser calculado um acréscimo de 50% no valor de cada hora trabalhada acima das 40 horas.

Com isso, a codificação da solução do problema em Python fica:

```
1  # -*- coding: utf-8 -*-
2  # Programa: empresa.py
3  # Programador:
4  # Data: 03/04/2016
5  # Este programa lê um número de um funcionário, e o salário mensal ou
6  # horas trabalhadas e o valor pago por hora trabalhada. Toda semana a
7  # empresa faz um adiantamento de salário aos funcionários. Se o número do
8  # funcionário for maior ou igual a 1000 o adiantamento é calculado por
9  # meio do valor do pagamento mensal dividido por 4. Se o número do
10 # empregado for menor que 1000 o valor do adiantamento é calculado por
11 # meio do número de horas trabalhadas vezes o valor pago por cada hora
12 # trabalhada. Todas as horas trabalhadas acima de 40 horas tem um
13 # adicional de hora extra de 50%. O programa calcula e imprime o valor do
14 # adiantamento semanal de cada funcionário.
15 # início do módulo principal
16 # descrição das variáveis utilizadas
17 # int num_funcionario
18 # float salario_mensal, valor_hora, numero_horas
19 # float valor_horasextras, adiantamento_semanal
20
21 # pré: num_funcionario (salario_mensal or valor_hora numero_horas)
22
23 # Passo 1. Leia as informações do salário dos funcionários
24 # Passo 1.1. Leia o número do funcionários
25 print('Leia o número do funcionário: ')
26 num_funcionario = int(input())
27 # Passo 1.2. Se funcionário mensalista leia salário mensal
28 if num_Funcionario >= 1000:
```

```
   print('Leia o valor mensal do salário: R$ ')
   salario_mensal = float(input())
# Passo 1.3. Caso contrário leia o valor da hora e número de horas
else:
   print('Entre com o valor pago por cada hora: R$' )
   valor_hora = float(input())
   print('Leia o número de horas trabalhadas: ')
   numero_horas = float(input())
# Passo 2. Calcule o valor do adiantamento semanal
# Passo 2.1. Calcule o adiantamento para funcionários mensalistas
if num_Funcionario >= 1000:
   adiantamento_semanal = salario_mensal / 4.0
# Passo 2.2. Calcule o adiantamento para funcionários horistas
else:
   adiantamento_semanal = numero_horas * valor_hora
# Passo 2.2.1. Calcule o valor das horas extras
   if numero_Horas > 40:
      valor_horasextras = (numero_horas - 40) * 0.5 * valor_hora
      adiantamento_semanal = adiantamento_semanal + valor_horasextras
# Passo 3. Imprima os resultados
print('Funcionário no.: {0:d}'.format(num_funcionario))
print('Adiantamento Semanal: R${0:8.2f}'.format(adiantamento_semanal))

# pós: num_funcionario, adiantamento_semanal
# fim do módulo principal
```

Verificação e Teste:

Exemplo 1

```
# formato da entrada
Leia o número do funcionário:
1042
Entre com o valor do salário mensal do funcionário: R$
10000.00

# formato da saída
Funcionário no.: 1042
Valor do adiantamento semanal: R$ 2500.00
```

Exemplo 2

```
# formato da entrada
Leia o número do funcionário:
543
Entre com o valor pago por cada hora trabalhada do funcionário: R$
30.00
Entre com o número de horas trabalhadas pelo funcionário:
50
```

```
# formato da saída
Funcionário no.: 543
Valor do adiantamento semanal: R$ 1650.00
```

1.2 Ordenando três números

Em computação, busca e ordenação são duas das atividades mais utilizadas. O desenvolvimento de algoritmos eficiente para busca e ordenação é um tema de pesquisa contínuo.

Como no caso anterior, vamos projetar um algoritmo que utiliza comparações entre dois elementos de um conjunto para ordenar esse conjunto. Não temos como projetar um algoritmo com essas características sem utilizar a estrutura de controle de seleção. Existem algoritmos de ordenação que não utilizam comparação entre os elementos, mas não fazem parte do programa desta disciplina.

Consideraremos o problema de ordenar o conjunto (lista) das notas obtidas numa dada disciplina. A avaliação da disciplina de Algoritmos e Programação I é feita por meio de 3 provas. Projete um algoritmo para ler a lista das três notas, obter uma nova lista com as notas ordenadas em ordem não decrescente e imprimir a nova lista.

Exemplo

```
# formato da entrada
7.5 8.2 6.4

# formato da saída
6.4 7.5 8.2
```

A descrição detalhada desse problema é imediata:

```
# Este algoritmo lê uma lista com três notas entre 0.0 e 10.0, ordena a
# lista de notas em ordem não decrescente e imprime o resultado.
```

As especificações de entrada e saída são:

Entrada	**Saída**
Uma lista com 3 números reais $nota1$, $nota2$ e $nota3$	lista com 3 números reais $nota1' \leq nota2' \leq nota3'$

Nesse exemplo as variáveis são compostas de tipos básicos. Necessitamos do tipo `float` para armazenar os valores da lista das três notas, `nota1`, `nota2`, `nota3` e `aux` (armazena um valor temporário das notas).

```
# descrição das variáveis utilizadas
# float nota1, nota2, nota3, aux
```

sendo que as pré e pós-condições ficam:

```
# pré: para i = {1, 2, 3}:nota[i] | 0.0 =< nota[i] <= 10.0

# pós: para i = {1, 2}:nota[i] | nota[i] <= nota[i+1]
```

Exemplo

```
# formato da entrada
7.5 8.2 6.4

# estado das variáveis após a leitura dos dados
nota1 == 7.5
nota2 == 8.2
nota3 == 6.4

# formato da saída
6.4 7.5 8.2
```

Vamos agora descrever os passos do Algoritmo:

```
Algoritmo: OrdenaNotas
# Passo 1. Leia as notas
# Passo 2. Ordene as notas nota1, nota2, nota3
# Passo 3. Imprima as notas ordenadas
# fim do algoritmo
```

O Passo 2 necessita um melhor detalhamento. Necessitamos descrever como faremos a ordenação. Uma solução ingênua para esse problema é usar a estrutura de seleção e descrever todas as possibilidades para imprimir as três notas ordenadas.

```
if (nota3>=nota2) and (nota2>=nota1):
   print(nota1, nota2, nota3)
if (nota2>=nota3) and (nota3>=nota1):
   print(nota1,nota3, nota2)
if (nota1>=nota3) and (nota3>=nota2):
   print(nota2, nota3, nota1)
if (nota3>=nota1) and (nota1>=nota2):
   print(nota2, nota1, nota3)
if (nota1>=nota2) and (nota2>=nota3):
   print(nota3, nota2, nota1)
if (nota2>=nota1) and (nota1>=nota3):
   print(nota3, nota1, nota2)
```

Essa abordagem, apesar de solucionar o problema não é eficiente, pois ela depende do número de permutações do conjunto de entrada e esse número é $n!$.

Temos que explorar outras alternativas para obter uma ordenação que não dependa das permutações. Existe um número muito grande de algoritmos de ordenação. Na nossa disciplina estudaremos os algoritmos mais básicos.

Um desses algoritmos é o algoritmo da ordenação por seleção e consiste em selecionar o menor elemento e armazenar no primeiro elemento, o segundo menor no segundo elemento, e assim sucessivamente. Nesse caso, sempre teremos uma única impressão da saída e o algoritmo se encarregará de armazenar o elementos nas variáveis adequadas.

Inicialmente colocamos a menor nota na variável `nota1`. Nesse caso, temos três possibilidades: a menor nota já estar armazenada em `nota1`, e nesse caso não necessitamos fazer

nada, ou ela está armazenada em `nota2` ou `nota3`. Basta nesse caso verificar se a nota está armazenada em `nota2` ou `nota3` e mover a menor nota para `nota1`.

Para verificar se a menor nota está em `nota2` ou `nota3` basta comparar se o valor armazenado em `nota1` é maior que `nota2` ou maior que `nota3`.

```
(nota2 < nota2) or (nota3 < nota1)
```

Se uma dessas cláusulas for verdadeira, temos que verificar qual variável (`nota2` ou `nota3`) armazena a menor nota e mover (trocar) o menor valor para `nota1` (o valor que estiver armazenado em `nota1` será movido para essa variável).

```
# Passo 2.1. Calcule a menor nota e armazene em nota1
if nota2 < nota1: # nota2 é a menor nota
   aux = nota1
   nota1 = nota2   # nota2 para nota1
   nota2 = aux     # nota1 para nota2
if nota3 < nota1: # nota3 é a menor nota
   aux = nota1     # armazena nota1
   nota1 = nota3   # nota3 para nota1
   nota3 = aux     # nota1 para nota3
```

Após o Passo 2.1, o valor da menor nota está armazenada na variável `nota1`. Agora temos que armazenar na variável `nota2` o valor da segunda menor nota. Como nessa fase temos apenas duas variáveis (`nota2` e `nota3`) a solução é mais simples, pois basta comparar as duas variáveis para computar a segunda menor nota, e consequentemente obter a terceira menor nota.

```
# Passo 2.2. Calcule a segunda menor nota e armazene em  nota2
if nota3 < nota2:
   aux = nota3
   nota3 = nota2
   nota2 = aux
```

Após o Passo 2.2., as variáveis `nota1`, `nota2` e `nota3` armazenam os valores das notas em ordem não decrescente.

A descrição completa do programa em Python fica:

```
1  # -*- coding: utf-8 -*-
2  # Programa: ordenanotas00.py
3  # Programador:
4  # Data: 22/09/2016
5  # Este algoritmo lê uma lista com três notas entre 0.0 e 10.0, ordena a
6  # lista de notas em ordem não decrescente e imprime o resultado.
7  # início do módulo principal
8  # descrição das variáveis utilizadas
9  # float nota1, nota2, nota3, aux
10
11 # pré: para i = {0, 1, 2}:nota[i] | 0.0 =< nota[i] <= 10.0
12
13 # Passo 1. Leia as notas
```

```
nota1, nota2, nota3 = map(float, input().split())
# Passo 2. Ordene as notas nota1, nota2, nota3
# Passo 2.1. Calcule a menor nota e armazene em nota1
if nota2 < nota1: # nota2 é a menor nota
   aux = nota1
   nota1 = nota2   # nota2 para nota1
   nota2 = aux     # nota1 para nota2
if nota3 < nota1: # nota3 é a menor nota
   aux = nota1     # armazena nota1
   nota1 = nota3   # nota3 para nota1
   nota3 = aux     # nota1 para nota3
# Passo 2.2. Calcule a segunda menor nota e armazene em  nota2
if nota3 < nota2:
   aux = nota3
   nota3 = nota2
   nota2 = aux
# Passo 3. Imprima as notas ordenadas
print(nota1, nota2, nota3)

# pós: para i = {1,2}:nota[i] | nota[1] <= nota[i+1]
# fim do módulo principal
```

1.3 Ordenando quatro números

Vamos agora tentar ordenar uma lista com quatro números. A ideia é adaptar o algoritmo anterior para ordenar uma lista com 4 números. A estratégia básica do algoritmo do algoritmo anterior é a de obter o menor elemento da lista e colocá-lo na primeira posição da lista. Vamos utilizar essa mesma abordagem para projetar um algoritmo para ordenar uma lista com quatro números inteiros usando comparações. Como no caso anterior, o nosso projeto de um algoritmo para ordenar objetos utiliza comparações e a estrutura de controle de seleção. O algoritmo deve ler uma lista com quatro números inteiros, ordenar a lista de números em ordem não decrescente e imprimir a lista.

Exemplo

```
# formato da entrada
17 70 -15 23

# formato de saída
-15 17 23 70
```

Como no exemplo anterior, a descrição detalhada para esse problema é praticamente a própria descrição do problema.

```
# Este algoritmo lê uma lista de quatro números inteiros, ordena os
# números em ordem não decrescente e imprime a lista ordenada.
```

As especificações de entrada e saída ficam:

Entrada	**Saída**
Uma lista com 4 números inteiros $num1$, $num2$, $num3$ e $num4$	lista com 4 números inteiros $num1' \leq num2' \leq num3' \leq num4'$

Nesse exemplo as variáveis são compostas de tipos básicos. Necessitamos do tipo `int` para armazenar os valores de `num1`, `num2`, `num3`, `num4` e `aux`.

```
# descrição das variáveis utilizadas
# int num1, num2, num3, num4, aux
```

sendo que as pré e pós-condições ficam:

```
# pré: para i = {0, 1, 2, 3}: num[i] | num[i] é um número inteiro

# pós: para i = {0, 1, 2}, num[i] | num[i] <= num[i+1]
```

Os passos do Algoritmos são os mesmos do exemplo anterior:

```
Algoritmo: OrdenaLista
# Passo 1. Leia a lista de 4 números inteiros
# Passo 2. Ordene a lista de 4 números inteiros
# Passo 3. Imprima a lista de inteiros ordenada
# fim do algoritmo
```

Como no exemplo anterior, o Passo 2 necessita um melhor detalhamento. Necessitamos descrever como faremos a ordenação, agora com quatro elementos. Já vimos anteriormente que a solução ingênua para esse problema utilizaria um número exponencial de comparações.

Vamos ampliar a ordenação por seleção para uma lista com 4 elementos, para isso vamos analisara ordenação com sublistas. Essa ideia poderá ser utilizada para o ordenação de n elementos. Vamos considerar a nossa entrada como sendo uma lista `L = (num1, num2, num3, num4)`. Para ordenar a lista, primeiro computamos o menor elemento da lista L e movemos esse elemento para a primeira posição da lista. Isso é feito com a comparação dois a dois e troca, quando for o caso, dos elementos da lista `(num1 > num2)`, `(num1 > num3)` e `(num1 > num4)`. Com isso o primeiro elemento da lista estará na posição correta. Depois computamos a menor elemento da sublista com 3 elementos `(num2, num3, num4)` e o movemos para a primeira posição dessa sublista (segunda posição da lista original). Esse procedimento é repetido para a sublista `(num3, num4)`. No caso de uma sublista com apenas um elemento, ela já está com o menor elemento na primeira posição.

Como temos quatro elementos para ordenar, para computar o menor elemento da lista e colocá-lo na primeira posição da lista, temos que fazer três comparações. Uma vez colocado o menor elemento da lista na primeira posição, temos agora que computar o segundo menor e colocá-lo na segunda posição. Como o primeiro já está na posição correta, temos que analisar os elementos, dois, três e quatro da lista e mover o segundo menor para a segunda posição. Podemos fazer isso com duas comparações. Depois basta colocar o terceiro menor elemento na terceira posição, o quarto estará automaticamente na posição correta.

```
# Passo 2.1. Calcule a menor número e armazene em num1
if num2 < num1:
   aux = lista1
```

```
    num1 = num2    # num2 para num1
    num2 = aux     # num1 para num2
if num3 < num1:
    aux = num1
    num1 = num3    # num3 para num1
    num3 = aux     # num1 para num3
if num4 < num1:
    aux = num1
    num1 = num4    # num4 para num1
    num4 = aux     # num1 para num4
```

Após o Passo 2.1, o valor do menor número está armazenado na variável `num1`. Agora temos que armazenar na variável `num2` o valor do segundo menor número. Como nessa fase temos apenas três variáveis (`num2`, `num3` e `num4`) a solução é mais simples, pois bastam duas comparações para computar o segunda menor número.

```
# Passo 2.2. Calcule o segundo menor número e armazene em num2
if num3 < num2:
    aux = num2
    num2 = num3    # num3 para num2
    num3 = aux     # num2 para num3
if num4 < num2:
    aux = num2
    num2 = num4    # num4 para num2
    num4 = aux     # num2 para num4
```

Após o Passo 2.2., as variáveis `num1` e `num2` armazenam os valores dos elementos em ordem não decrescente. Falta ordenar a sublista com os elementos `num3` e `num4`.

```
# Passo 2.3. Calcule o terceiro menor número e armazene em num3
if num4 < num3:
    aux = num3
    num3 = num4    # num4 para num3
    num4 = aux     # num3 para num4
```

Os passos 2.1, 2.2. e 2.3 ordenam um lista de 4 números. A descrição completa do programa em Python fica:

```
1  # -*- coding: utf-8 -*-
2  # Programa: ordenanotas01.py
3  # Programador:
4  # Data: 22/09/2014
5  # Este algoritmo lê uma lista de quatro números inteiros e ordena os
6  # números em ordem não decrescente e imprime a lista ordenada.
7  # início do módulo principal
8  # descrição das variáveis utilizadas
9  # int num1, num2, num3, num4, aux
10
11 # pré: num1 num2 num3 num4
12
```

```
# Passo 1. Leia a lista de 4 inteiros
num1, num2, num3, num4 = map(int, input().split())
# Passo 2. Ordene a lista de 4 inteiros
# Passo 2.1. Calcule a menor número e armazene em num1
if num2 < num1:
   aux = lista1
   num1 = num2   # num2 para num1
   num2 = aux    # num1 para num2
if num3 < num1:
   aux = num1
   num1 = num3   # num3 para num1
   num3 = aux    # num1 para num3
if num4 < num1:
   aux = num1
   num1 = num4   # num4 para num1
   lista4 = aux     # num1 para num4
# Passo 2.2. Calcule o segundo menor número e armazene em num2
if num3 < num2:
   aux = num2
   num2 = num3   # num3 para num2
   num3 = aux      # num2 para num3
if num4 < num2:
   aux = num2
   num2 = num4   # num4 para num2
   num4 = aux    # num2 para num4
# Passo 2.3. Calcule o terceiro menor número e armazene em num3
if num4 < num3:
   aux = num3
   num3 = num4   # num4 para num3
   num4 = aux    # num3 para num4
# Passo 3. Imprima a lista de inteiros ordenada
print(num1, num2, num3, num4)

# pós: para i = {0,1, 2, 3}: num[i] | num[i] <= lista[i+1]
# fim do módulo principal
```

2 Exercícios

2.1 Poluição

O nível de poluição do ar na cidade de Cachorro Sentado é medido por um índice de poluição. As leituras são feitas às 12:00 hs em três locais: Olaria da Sucuri, no Bolicho do Jaburu, e num ponto aleatório na vila das Emas. A média dessas três leituras é usada como o índice de poluição da cidade, valores desse índice do ar abaixo de 35 indicam **condição agradável**, valores maiores ou iguais a 35 e menores ou iguais a 60 indicam **condição desagradável**, enquanto valores maiores que 60 indicam **condição perigosa**. Visto que o cálculo deve ser feito diariamente, a Agência de Meio Ambiente de Cachorro Sentado necessita um programa que calcule o índice de poluição e então determine a condição apropriada, agradável, desagradável ou perigosa.

Exemplo 1

```
# formato de entrada
65 35 31

# formato da saída
Condição Desagradável
```

Exemplo 2

```
# formato de entrada
38 22 21

# formato da saída
Condição Agradável
```

Como vimos anteriormente, em Python a atribuição de uma string a uma variável é feita diretamente.

```
msg = 'Condição Agradável'
```

2.2 Ordena Notas

Descrevemos acima soluções para os problemas de ordenar uma lista com três notas e o de ordenar uma lista com 4 números inteiros. Usando essas ideias, projete e implemente um programa na linguagem Python (`ordnotas.py`) que leia quatro notas de 0.0 a 10.0 e ordene em ordem não decrescente as notas e imprima a lista das notas ordenadas.

Exemplo 1

```
# formato de entrada
8.0
5.0
7.0
9.0

# formato da saída
5.0 7.0 8.0 9.0
```

Exemplo 2

```
# formato de entrada
6.0
9.0
8.0
7.0

# formato da saída
6.0 7.0 8.0 9.0
```

Usando as mesmas ideias, projete e implemente um programa em Python que leia uma lista com 5 notas, ordene a lista em ordem não decrescente e imprima o resultado.