Aula 02 – Exercícios

Parte 1 – Escrita (entregue via Formulário no Google Classroom)

Nos slides da Aula 02, foram deixados 4 exercícios para serem executados no papel com os algoritmos estudados. Em seu caderno, simule a execução dos exercícios correspondentes. Ao final, você deve escanear ou fotografar as páginas com a câmera do celular, e submetê-las no formulário que acompanha esta atividade. As páginas devem ser reunidas em um único arquivo PDF e devem estar legíveis. Atenção para enviar seu arquivo no Formulário disponível na atividade, e não na seção de upload do Classroom. Lembre-se de colocar nome nas suas folhas.

Parte 2 – Codificação (entregue via Run Codes)

Exercício 1: escreva, em linguagem C pura (sem nenhum comando próprio de C++), um programa que leia um valor N. Após criar um vetor de N posições, repasse-o, juntamente com o valor de N a uma função que leia e armazene valores no vetor. Por fim, execute o algoritmo Bubble Sort para ordenar o vetor e mostre, como resposta, uma única linha contendo dois valores numéricos, separados por um único espaço. O primeiro valor é a quantidade de comparações executadas pelo algoritmo de ordenação. O segundo valor é a quantidade de trocas realizadas.

Seu programa deve ser dividido em funções. É obrigatório que você escreva, pelo menos, as seguintes funções, seguindo os protótipos dados:

```
// esta função lê n elementos e os armazena no vetor v. Recebe como parâmetros o vetor v e a quantidade n.
void leVetor(int *v, int n);

// esta função ordena o vetor. Recebe como parâmetros o vetor v e o tamanho n do vetor.
void bubbleSort(int *v, int n);
```

Exercício 2: repetir o Exercício 1, porém, para o algoritmo Bubble Sort Inteligente, com uma pequena variação. O algoritmo deve ordenar o vetor em ordem decrescente. Ao final do processo, o maior elemento estará na posição 0 do vetor, e o menor em sua última posição. Seu programa deve imprimir, como resposta, duas linhas. A primeira linha contém os elementos do vetor ordenado, separados por um único espaço. A segunda linha contém dois valores numéricos, separados por um único espaço: o primeiro valor é a quantidade de comparações executadas pelo algoritmo de ordenação, enquanto que o segundo valor é a quantidade de trocas.

Seu programa deve ser dividido em funções. É obrigatório que você escreva, pelo menos, as seguintes funções, seguindo os protótipos dados:

```
// esta função lê n elementos e os armazena no vetor v. Recebe como parâmetros o vetor v e a quantidade n.
void leVetor(int *v, int n);
```

```
// esta função ordena o vetor. Recebe como parâmetros o vetor v e o tamanho n do vetor.
void smartBubbleSort(int *v, int n);
```

Exercício 3: escreva, em linguagem C ou C++, um programa que, inicialmente, leia o nome de um arquivo. A seguir, seu programa deve abrir esse arquivo e armazenar os valores nele encontrados em um vetor. Então, ordene o vetor em ordem crescente usando o algoritmo Comb Sort, porém, com uma pequena variação. Originalmente, a Fase 1 do Comb Sort analisa os elementos a serem trocados iniciando da última posição em direção a primeira. Em seu programa, esta análise deverá ser realizada de modo contrário: inicie pela posição 0 do vetor em direção à sua última posição. Seu programa deve imprimir várias linhas como resposta. Para cada troca realizada pelo Comb Sort, imprima uma linha contendo dois inteiros separados por um espaço: estes inteiros são os índices das posições trocadas. Ao final do processo de ordenação, imprima mais uma linha contendo dois inteiros separados por um espaço: a quantidade de comparações e a quantidade de trocas.

Observação 1: esta lista está dividida em uma parte escrita e uma parte prática de codificação. A entrega só será válida se as duas partes forem entregues.
Observação 2: todos os códigos foram testados no Run Codes e todos os casos de testes estão corretos.
Observação 3: a nota da última submissão realizada no Run Codes é a que conta.

Exercício 4 – Desafio

*Não é necessário entregar o Exercício 4, porém, se estiver **totalmente correto**, conta como nota extra. "**Totalmente correto**" significa "**passar em todos os casos de teste**".*

Uma das operações mais frequentes em computação é ordenar uma sequência de objetos. Portanto, não é surpreendente que essa operação seja também uma das mais estudadas.

Um algoritmo bem simples para ordenação é chamado Bubble sort. Ele consiste de vários turnos. A cada turno o algoritmo simplesmente itera sobre a sequência trocando de posição dois elementos consecutivos se eles estiverem fora de ordem. O algoritmo termina quando nenhum elemento trocou de posição em um turno.

O nome Bubble sort (ordenação das bolhas) deriva do fato de que elementos menores ("mais leves") movem-se na direção de suas posições finais na sequência ordenada (movem-se na direção do início da sequência) durante os turnos, como bolhas na água. Abaixo tem-se uma implementação do algoritmo em pseudocódigo:

```
para i variando de 1 até N faça
    para j variando de N - 1 a i faça
        se vet[j - 1]> vet[j] então
            intercambie os elementos vet[j - 1] e vet[j]
        fim-se
    fim-para
    se nenhum elemento trocou de lugar então
        final do algoritmo
    fim-se
fim-para
```

Por exemplo, ao ordenar a sequência [5, 4, 3, 2, 1] usando o algoritmo acima, quatro turnos são necessários. No primeiro turno ocorrem quatro intercâmbios: 1 x 2, 1 x 3, 1 x 4 e 1 x 5; no segundo turno ocorrem três intercâmbios: 2 x 3, 2 x 4 e 2 x 5; no terceiro turno ocorrem dois intercâmbios: 3 x 4 e 3 x 5; no quarto turno ocorre um intercâmbio: 4 x 5; no quinto turno nenhum intercâmbio ocorre e o algoritmo termina.

Embora simples de entender, provar correto e implementar, o algoritmo Bubble sort é muito ineficiente: o número de comparações entre elementos durante sua execução é, em média, diretamente proporcional a N^2, onde N é o número de elementos na sequência. Você foi requisitado para fazer uma "engenharia reversa" no Bubble sort, ou seja, dados o comprimento da sequência, o número de turnos necessários para a ordenação e o número de intercâmbios ocorridos em cada turno, seu programa deve descobrir uma possível sequência que, quando ordenada, produza exatamente o mesmo número de intercâmbios nos turnos.

Entrada

A entrada contém vários casos de teste. A primeira linha de um caso de teste contém dois inteiros N e M que indicam respectivamente o número de elementos ($1 \leq N \leq 100000$) na sequência que está sendo ordenada, e o número de turnos ($0 \leq M \leq 100000$) necessários para ordenar a sequência usando Bubble sort. A segunda linha de um caso de teste contém M inteiros X_i, indicando o número de intercâmbios em cada turno i ($1 \leq X_i \leq N - 1$, para $1 \leq i \leq M$). O final da entrada é indicado por $N = M = 0$.

Saída

Para cada caso de teste da entrada seu programa deve produzir uma linha na saída, contendo uma permutação dos números {1, 2, ..., N}, que quando ordenada usando Bubble sort produz o mesmo número de intercâmbios no mesmo número de turnos especificados na entrada. Ao imprimir a permutação, deixe um espaço em branco entre dois elementos consecutivos. Se mais de uma permutação existir, imprima a maior na ordem lexicográfica padrão para sequências de números (a ordem lexicográfica da permutação $a_1, a_2, ..., a_N$ é maior do que a da permutação $b_1, b_2, ..., b_N$ se para algum $1 \leq i \leq N$ temos $a_i > b_i$ e o prefixo $a_1, a_2, ..., a_{i-1}$ é igual ao prefixo $b_1, b_2, ..., b_{i-1}$).

Em outras palavras, caso exista mais de uma solução, imprima aquela onde o primeiro elemento da permutação é o maior possível. Caso exista mais de uma solução satisfazendo essa restrição, imprima, dentre estas, aquela onde o segundo elemento é o maior possível. Caso exista mais de uma solução satisfazendo as duas restrições anteriores, imprima, dentre estas, a solução onde o terceiro elemento é o maior possível, e assim sucessivamente.

Para toda entrada haverá pelo menos uma permutação solução.

Implemente em C ou C++.