Documentação acerca do exercício "Soma Máxima":

1. As bibliotecas utilizadas foram apenas a "stdio.h" e a "stdlib.h";
2. Foram adicionadas ao programa algumas frases para guiar o usuário durante a execução;
3. Adicionei um *while* que itera enquanto o usuário não inserir um valor válido para *n* (entre 3 e 20), após o usuário inserir um número válido, o laço de repetição encontra um *break* e passa para a instrução de baixo;
4. Em seguida aloquei memória dinamicamente para o vetor, utilizando *malloc*, com base no valor de n inserido pelo usuário e fiz um *for* para permitir que o usuário entrasse com os valores de cada elemento do vetor;
5. A lógica na qual eu pensei para solucionar o problema consiste em percorrer todos os *n* elementos do vetor *n* vezes e, a cada iteração, eu somo o elemento atual do vetor à soma atual (inicializada como zero) e verifico se a soma atual é maior que a soma máxima (também inicializada como zero), e se for, eu atualizo essa variável. Também são analisadas as somas contíguas que não se iniciam na posição zero do vetor. Essa lógica foi implementada da seguinte forma:
6. Primeiramente eu inicializei 3 variáveis como zero: *soma_maxima*, *primeiro_indice* e *ultimo_indice*. Em seguida, construí um *for*, no qual a variável *i* é inicializada como 0 e vai iterando enquanto seu valor for menor que *n* (a cada iteração essa variável *i* é acrescida), e dentro desse *for* implementei outro *for*, no qual a variável *j* é inicializada igual ao *j* atual, e que itera enquanto seu valor for menor que *n* (a cada iteração essa variável *j* é acrescida);
7. O *for* de dentro serve para percorrer os elementos do vetor, inicialmente da primeira posição até a última, e após percorrer, o *for* externo itera, e dessa vez o *for* de dentro percorre o vetor partindo da segunda posição até a última (e assim, sucessivamente, sempre que o *for* externo iterar, o de dentro começa partindo da posição seguinte do vetor, em relação à que ele começou na iteração anterior do *for* externo);
8. Dentro do *for* interno, é somado o valor do vetor na posição atual à variável *soma_atual*, e em seguida, um *if* teste se *soma_atual* é maior que *soma_maxima*, e se for, a soma máxima passa a ser a soma atual e as duas variáveis *primeiro_indice* e *ultimo_indice* salvam, respectivamente, o valor dos índices inicial e final desse intervalo de subvetores contíguos;
9. Entre os dois *for*, a variável *soma_atual* é sempre inicializada como zero, para reiniciar o seu valor para as próximas iterações do *for* interno.
10. No final da execução, o programa exibi na tela qual é a maior soma de subvetores contíguos e seu respectivo intervalo de índices;
11. Se o usuário insere um vetor composto por apenas números positivos, a soma máxima retornada é a soma de todos os elementos do vetor;
12. Se o usuário insere um vetor composto apenas por números negativos, a soma máxima retornada é sempre zero, pois como no programa temos um *if* que testa se a variável *soma_atual* é maior que a *soma_maxima*, se essa *soma_atual* for negativa, ela nunca será maior que zero, e assim, a *soma_maxima* continuará sendo zero.

13. No que diz respeito à análise de complexidade de pior caso, o custo assintótico de execução desse algoritmo é $O(n^2)$, visto que os *n* elementos do vetor são percorridos *n* vezes.

Documentação acerca do exercício "Quadrado Mágico":

1. As bibliotecas utilizadas foram apenas a "stdio.h", a "stdlib.h" e a "math.h;
2. Foram adicionadas ao programa algumas frases para guiar o usuário durante a execução;
3. Primeiramente eu criei as seguintes funções: *int get_numMagico()*, que recebe como parâmetro um inteiro *n* (lado do quadrado mágico) e retorna o número mágico (soma das linhas, colunas e diagonais) de acordo com a fórmula $n^* (n^2 + 1) / 2$, *void exibir_num_magico()*, que recebe como parâmetro um inteiro *n* (lado do quadrado mágico) e exibe na tela o lado e a soma, *void exibir_matriz()*, que recebe como parâmetro o inteiro *n* e uma matriz *n* x *n*, e exibe a matriz na tela, e *void liberar_matriz()*, que recebe como parâmetro o inteiro *n* e a matriz *n* x *n*, e libera a memória alocada dinamicamente para a matriz.
4. Agora dentro do *main*, inicializei um ponteiro de matriz e um inteiro *n*;
5. Adicionei um *while* que itera enquanto o usuário não inserir um valor válido para *n* (entre 3 e 6), após o usuário inserir um número válido, o laço de repetição encontra um *break* e passa para a instrução de baixo;
6. Em seguida, aloquei memória dinamicamente para uma matriz de inteiros, utilizando a função *malloc*, alocando memória para as linhas, e em seguida, para as colunas.
7. Decidi utilizar um *switch* com 4 casos para que o programa avançasse para a instrução de acordo com o valor de n inserido pelo usuário;
8. A lógica para se criar um quadrado mágico de lado 3 consiste em enumerar uma matriz 3x3 com os números de 1 a 9 da seguinte forma: coloca-se o 1 na posição superior do meio, e em seguida vai numerando de acordo com a regra "cima, direita", e se nessa posição já houver outro número, você volta e coloca o número na posição abaixo da última posição enumerada;
9. Em *case 3*, ou seja, no caso em que o usuário deseja um quadrado mágico de lado 3, eu utilizei dois *for*, um dentro do outro, com *i e j* variando de 0 a 2, para criar uma matriz 3x3 de zeros;
10. Após isso, reinicializei 2 variáveis, *i* e *j*, com *i* = 0 e *j* = 1, para significarem a posição superior do meio de uma matriz 3x3;
11. Em seguida, fiz um *for*, com a variável *num* variando de 0 a 9 (n^2), e dentro desse *for* eu implementei a lógica do quadrado de lado 3, que será detalhada nos próximos tópicos;
12. Primeiramente, temos um *if*, que analisa se o elemento que ocupa a posição *matriz[i][j]* (inicialmente, *matriz[i][j] = matriz [0][1]*) é igual a zero, se for verdadeiro, *matriz[i][j]* recebe *num + 1* (que no começo é igual a 1), em seguida diminuímos 1 na variável *i* e acrescentamos 1 em *j* (para que o "cursor" (*matriz[i][j]*) , mova-se no sentido "cima, direita"), em seguida temos um *if* que analisa se após essa mudança o elemento está numa posição válida, e se *i* for menor que 0, é acrescido 3 a ele, se *j* for maior ou igual a 3, é subtraído 3 dele, fazendo com que *matriz[i][j]* fique na posição esperada;

13. Para o caso de o elemento em *matriz[i][j]* ser diferente de zero, ou seja, a posição já ter sido ocupada por algum *num + 1*, temos um *else*;
14. Dentro desse *else*, nós somamos 2 à variável i e subtraímos 1 da variável j para fazer o "cursor" voltar para a posição abaixo da última "enumerada", e em seguida atribuímos *num + 1* à *matriz[i][j];*
15. Ainda nesse *else*, subtraímos 1 da variável *i* e somamos 1 à variável *j*, para o "cursor" ir para a posição "cima, direita", e temos também aqueles *if*'s que consertam a posição de *matriz[i][j]* em caso de o cursor ir para uma posição inválida (no caso em que *i* é menor que zero ou *j* é maior ou igual a 3);
16. Seguindo essa lógica, um quadrado mágico de lado 3 é criado, e em seguida chamamos as funções que exibem o número mágico, a matriz, e no final, a que libera a memória alocada dinamicamente para a matriz;
17. A lógica para se formar um quadrado mágico de lado 4 consiste em criar uma matriz 4x4 e ir enumerando as posições com os números de 1 a 16, em ordem crescente (primeiro enumera a linha e em seguida passa para a linha de baixo), e depois inverter os elementos das diagonais internas e externas (4 trocas no total).
18. Em *case 4,* ou seja, no caso em que o usuário deseja um quadrado mágico de lado 4, primeiro eu inicializei uma variável "*elemento"* igual a 1, e utilizei dois *for*, um dentro do outro, com *i* e *j* variando de 0 a 3, para criar uma matriz 4x4 e enumerá-la de 1 a 16, em ordem crescente;
19. Feito isso, inverti os elementos das diagonais externas e internas, respectivamente, utilizando uma variável auxiliar para segurar um valor enquanto eu fazia a troca;
20. Após fazer as 4 trocas, finalizei a criação do quadrado mágico, em seguida, chamei as funções para exibi-lo na tela junto do número mágico, e no final liberei a memória alocada dinamicamente para a matriz;
21. A lógica do quadrado mágico de lado 5 é a mesma do quadrado mágico de lado 3;
22. Em *case 5,* ou seja, no caso em que o usuário deseja um quadrado mágico de lado 5, utilizei o mesmo algoritmo do *case 3*, porém onde era "3" agora é "5" e o *for* itera de 0 a 25;
23. A lógica do quadrado mágico de lado 6 consiste em dividir uma matriz 6x6 em 4 quadrantes 3x3, e enumerá-los de acordo com a lógica do quadrado mágico de lado 3. Para o primeiro quadrante (da esquerda para direita e de cima para baixo), usamos os números de 0 a 9, para o segundo, de 19 a 27, para o terceiro, de 10 a 18 e, para o último, de 28 a 36. Depois, devemos trocar os elementos das posições [0][0], [1][1] e [2][0] do primeiro quadrante, com o quadrante de baixo (mesmas posições, respectivamente);
24. Em *case 6,* ou seja, no caso em que o usuário deseja um quadrado mágico de lado 6, utilizei o algoritmo do *case 3* para formar um quadrado mágico 3x3, que é o primeiro quadrante, e utilizei um *for* dentro de outro *for* para completar simultaneamente os outros 3 quadrantes, de acordo com as posições já completadas do primeiro quadrante, ou seja, para completar o 2º quadrante, eu fui assinalando o valor de *matriz[i][j] + 18* à *matriz[i][j+3]*, para completar o 3º quadrante, assinalei o valor de *matriz[i][j] + 27* à *matriz[i+3][j]* e, para completar o 4º quadrante, assinalei o valor de *matriz[i][j] + 9* à *matriz[i+3][j+3]*;
25. Em seguida, fiz as trocas necessárias, utilizando variável auxiliar para segurar valores enquanto fazia a troca;

26. Feito isso, chamei as funções para exibir a matriz na tela junto ao número mágico e, por fim, liberei a memória alocada dinamicamente para a matriz;
27. No que diz respeito à análise de complexidade assintótica, o custo assintótico de execução desse algoritmo é $\Theta(n^2)$, visto que para todos os casos, as matrizes são formadas por n linhas e n colunas, ou seja, n^2 elementos, e todos os elementos são percorridos, logo, esse algoritmo é $\Omega(n^2)$. A soma das linhas e das colunas devem ser iguais, como temos n linhas, n colunas e 2 diagonais, com n entradas cada, o resultado da complexidade é $O(n^2)$. Como o algoritmo é $\Omega(n^2)$ e $O(n^2)$, então ele é $\Theta(n^2)$.