LAB 5 (08 QUESTIONS)

Ex1: implement function **int countDiv3(int arr[])** of which inputs are an array and its size.

This function will return the number of elements which is divisible by 3.

Initialize your array in **main**() (scanner is not required) and call **countDiv3**() to get your result

*Note: Do NOT use sout() in function.*

*Use sout() only in main()*

Sample run

arr[] = {1, 3, 5, 7, 9}

there are 2 elements divisible by 3

Ex2: implement function **boolean isAscending(int arr[])** to check if elements in an array are in ascending order.

This function returns true if elements are in ascending order; otherwise returns false.

Initialize your array in **main**() (scanner is not required) and call **isAscending**() to get your result

*Note: Do NOT use sout() in function.*

*Use sout() only in main()*

Sample run 1:

arr[] = {1, 3, 7, 0, 9}

Elements are NOT in ascending order

Sample run 2:

arr[] = {1, 3, 7, 8, 9}

Elements are in ascending order

Ex3\*: implement function **int greatestSum(int[] arr)** to find the greatest sum of 2 elements in an array

This function returns the greatest sum of 2 elements in an array

Initialize your array in **main**() (scanner is not required) and call **greatestSum**() to get your result

*Note: Do NOT use sout() in function.*

*Use sout() only in main()*

Sample run 1:

arr[] = {1, 3}

the greatest sum is 4

Sample run 2:

arr[] = {1, 3, 7, 0, 9}

the greatest sum is 16

Sample run 3:

arr[] = {37, 13, 37, 10, 9}

the greatest sum is 74

Ex4: implement function **int sumLast3(int arr[])** to compute the sum of the last 3 elements in an array

Initialize your array in **main**() (scanner is not required) and call **sumLast3**() to get your result

*Note: Do NOT use sout() in function.*

*Use sout() only in main()*

Sample run

|  |  |
| --- | --- |
| **arr[]** | **sumLast3(arr, size)** |
| { } | 0 |
| {5} | 5 |
| {12, -3} | 9 |
| {20, 12, 25, 8, 36, 9} | 53 |
| {-1, 2, -3, 4, -5, 6, -7, 8, 9, 10} | 27 |

Ex5: Initialize your 2D array in **main**() (scanner is not required) and find the minimum elements of the array

Sample run 1:

arr[] = { {1, 2}, {-2, 3} }

min = -2

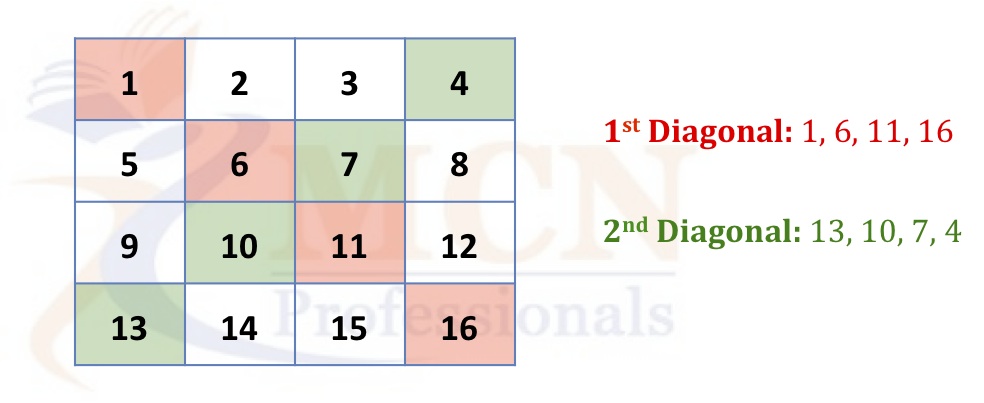
Sample run 2:

arr[] = { {1, 2, 4, -2}, {-2, -3, -1, 0} }

min = -3

Ex6: A square array is an array that has the number of rows equals the number of columns e.g. arr[2][2], arr[3][3]…

In a square array, there are 2 diagonals (see the picture)



Initialize a square array in **main**() and find 2 products of elements in 2 diagonal lines

Sample run

arr[4][4] = {

{1, 2, 3, 4},

{5, 6, 7, 8},

{9, 10, 11, 12},

{13, 14, 15, 16}

}

First product is 1056, second product is 3640

Ex7:

1. Implement function **boolean isPartOf(int x, int arr[])** to check whether arr contains x or not.

This function returns true if arr contains x, otherwise returns false.

Initialize your array in **main**() (scanner is not required) and call **isPartOf**() to get your result

*Note: Do NOT use sout() in function.*

*Use sout() only in main()*

Sample run 1:

arr[] = {37, 13, 37, 10, 9}

x = 10

10 is a part of arr

Sample run 2:

arr[] = {37, 13, 37, 10, 9}

x = 11

11 is NOT a part of arr

1. Implement function **boolean isSubset(int arrA[], int arrB[])** to check whether arrA is a subset of arrB.

U is a subset of V if and only if all elements in U belongs to V (i.e. V contains all elements of U)

The function returns true if arrA is a subset of arrB, otherwise returns false.

Initialize your array in **main**() (scanner is not required) and call **isSubset**() to get your result

*Hint: use* ***isPartOf****() in* ***isSubset****()*

*Note: Do NOT use sout() in function.*

*Use sout() only in main()*

Sample run 1:

arrA[] = {37, 13, 37, 10, 9}

arrB[] = {37, 0, 13, 37, 3, 10, 9}

arrA is a subset of arrB

Sample run 2:

arrA[] = {37, 13, 37, 10, 9, 11}

arrB[] = {37, 0, 13, 37, 3, 10, 9}

arrA is NOT a subset of arrB

Ex8:

1. Initialize your array **arr**[] and number **pivot** in **main**(), print out all elements in arr that are less than or equal pivot and all elements that are greater than pivot

Sample run

arr[] = {37, 13, 37, 10, 9}

pivot = 11

Less than or equal 11: 10, 9

Greater than 11: 37, 13, 37

1. Implement function **void partition(int arr[], int pivot)** which does the following
   1. Move all elements less than or equal pivot to the left of array
   2. Move all elements greater than pivot to the right of array

Initialize your array and pivot in **main**() (scanner is not required) and call **partition**() to *modify your array* according to the above requirements.

Sample run

arr[] = {37, 13, 37, 10, 9}

pivot = 11

arr[] = {10, 9, 37, 13, 37} //arr is modified after calling partition()