**TASK A**

**Question 1**

The QuestionOne() function is used to initialize an inventory (in x and y dimensions), and load it with objects.

Code Analysis:

- The following operations have fixed overhead and can’t be optimized further:

a). Console.WriteLine("Inventory initialisation"); cost=1

Console.WriteLine("Define X dimension size: "); cost=1

Console.WriteLine("Define Y dimension size: "); cost=1

Item[,] inventory = new Item[x, y]; cost=1

Console.WriteLine("Now loading inventory"); cost=1

Console.WriteLine("Inventory finished loading."); cost=1

Console.WriteLine("Happy travels!"); cost=1

--------------------------

total=4

--------------------------

b).

i). The loop:

for (int i = 0; i < x; i++){

runs x times

ii) within the loop:

- Console.WriteLine("Starting on row " + (i + 1));

has same cost as for one operation, for each of the outer loop iteration

iii). for the inner loop:

for (int j = 0; j < y; j++){

- the loop runs x\*y times (as it has iteration condition y, and it is nested within the outer loop with iteration condition of x).

- within this loop:

- Console.WriteLine("Current Coords. " + i + " , " + j); has the cost of one single operation

- Item test = new Item((i + " , " + j), 1, 1); creates a new item and has the cost of one operation

- inventory[i, j] = test; insertion of object into array has the cost of one operation

- Console.WriteLine("Added " + test.Name + " at Coords."); has the cost of a single operation.

Total cost of **each** operation within this loop = 4

Total cost of **all** operations within this loop = 4\*x\*y

Two additional fixed operations from the inner loop:

The 2 Console.WriteLine(); have the cost of 2\*x operations

The Console.ReadLine(); has the cost of 1 operation

Totoal cost of operations for QuestionOne():

7 + x 4\*x\*y + 3 = 10+x+4\*x\*y

The Big O notation:

O(x2)

or

O(y2)

or

O(xy)

**Question 2**

This code creates an array, given the dimension specified by the user, then analyzes the array by printing the values of the members:

Code Analysis:

a). The following lines will have fixed overhead and can’t be optimized further:

Console.WriteLine("How many items to analyse?: "); cost=1

Console.WriteLine("Items analysed"); cost=1

Console.ReadLine(); cost=1

----------------------------

total=4

----------------------------

b). for the for loop:

for (int i = 0; i < (n - 1); i++){

i). thereare n-1 iterations, and so for n-1 time:

Item current = items[i]; cost=1

Console.WriteLine("Displaying value of: " + current.Name); cost=1

Console.Write(current.Value); cost=1

-------------------------

total=3

-------------------------

For the for-loop, total cost = (n-1)\*3 = 3\*(n-1)

c). after the for loop:

Console.WriteLine("Items analysed"); cost=1

Console.ReadLine(); cost=1

-------------------------

cost=2

------------------------

Total cost for a), b) and c) above = 4 + 3\*(n-1) + 2

= 6 + 3\*(n-1)

= 6 + 3\*n – 3

= 3 + 3\*n

Therefore the Big O notation for this function is O(n)

**TASK B**

static public void GenericInsertSort<T>(T[] a) where T : Icomparable{

//for a null or empty array

if (a == null || a.Length == 0){

throw new ArgumentException("You must add items to the array");

}

//to be removed when the needed compareTo() function is added

//throw new NotImplementedException("Required implementation in development phase");

for (int i = 1; i < a.Length; i++){

T value = a[i];

int j = i;

for (; j > 0 && value.CompareTo(a[j - 1]) < 0; j--){

a[j] = a[j - 1];

}

a[j] = value;

}

}

The CompareTo() is preferred as this new implementation will not only be able to sort Integers, but other types also, including Strings, as they implement Icomparable. All other implementation remain unchanged.

**TASK C**