# E:\Prj\Pulse Big Screen App\BigScreenApp\BigScreenApp.View\Images\resonatelogo.png1. Technical Exercise

Consider a method that takes as input a rectangular array double[,] and efficiently calculates the aggregated result of each column for a number of aggregation functions.

You can find a dotnet fiddle <https://dotnetfiddle.net/RelwNb> with a small amount of code in it. In this you are expected to provide the implementation of a single method:

public static Dictionary<string, double>[] DoAggregations(double[,] input)

{

throw new NotImplementedException();

}

The fiddle will validate the output against for the below scenario, as well as assess performance against a larger dataset.

Example input & output:

|  |  |  |  |
| --- | --- | --- | --- |
|  | 1 | 4 | 3 |
|  | 2 | 3 | 4 |
|  | 1 | 3 | 7 |
|  | 5 | 4 | 2 |
| SUM | 9 | 14 | 16 |
| AVERAGE | 2.25 | 3.5 | 4 |
| COUNT DISTINCT | 3 | 2 | 4 |

A: Implement the DoAggregations method as performant as possible for 3 supported aggregations: SUM, AVERAGE and COUNT DISTINCT. What is the space & time complexity of the implementation? Please fork/copy the fiddle with a solution and provide this link back.

B: Are there any trade-offs if we allow a generic aggregations method to be used? E.g. other developers use your library and provide their own aggregation methods (such as Func<>)

# 2. Code Review

|  |
| --- |
| public class Foo<T>  {  static protected T StartValue { get; private set; }  public Foo(T startValue)  {  StartValue = startValue;  }  public static int DoCalc(object arg1, T arg2)  {  return (int)arg1 + Convert.ToInt32(arg2) / Convert.ToInt32(StartValue);  }  public static int higherOrderCalculation(Func<int, int, int> Func, int one, int two)  {  return Func(one, two);  }  public void LogToConsole(string message)  {  Console.WriteLine(string.Format(message));  }  public async void LogtoFile(string msg)  {  await File.AppendAllTextAsync($"logfile{DateTime.Now.ToString("yyyy-mm-dd")}.log", "msg");  }  } |

The above class Foo is a working piece of code, up for code review by you. The code is fully functional, as the author of the code walks you through it, they demonstrate this with the below code.

var foo = new Foo<double>(1.5d);

foo.LogToConsole("Hello world");

foo.LogtoFile("Hello world");

var result1 = Foo<double>.DoCalc(1, 2);

var result2 = Foo<double>.higherOrderCalculation((x, y) => x + y, 3, 4);

**Please provide:**

- A bullet-pointed list of all the items that are poor/wrong in this class. From blatant mistakes, to best practices.

- The best, most awesome version of the class Foo possible. Code truly worthy to go into production.