1. Create a data-structure for efficient (log n) retrieval and update of inventory at any time bucket. This data structure should expose an interface to

- AddSupply(int bucket, float delta)
- AddDemand(int bucket, float delta)
- float GetInventory(int bucket)

All in log(n) time.

Inventory in any bucket is given by

$$i_n = i_{n-1} + s_n - d_n$$

where

i_n is Inventory in n^{th} bucket
s_n is Supply in n^{th} bucket
d_n is Demand in n^{th} bucket

The following example shows the successive states after each operation.

AddSupply(2, 50);

Bucket		0	1	2	3	4	5	6	7	8	9
Supply				50							
Demand											
Inventory		0	0	50	50	50	50	50	50	50	50

AddDemand(3, 25);

Bucket		0	1	2	3	4	5	6	7	8	9
Supply				50							
Demand					25						
Inventory		0	0	50	25	25	25	25	25	25	25

AddDemand(2, 30);

Bucket		0	1	2	3	4	5	6	7	8	9
Supply				50							
Demand				30	25						
Inventory		0	0	20	-5	-5	-5	-5	-5	-5	-5

2. Following code implements a semaphore of a given capacity using C# class System.Threading.Monitor.
Learn the basics of the System.Threading.Monitor class methods such as Wait()/Pulse()/Enter()/Exit()
Identify weaknesses/bugs, if any, in the following code (before the interview)
We will discuss during the interview.   (10 min)

```
public class Semaphore
{
     private object _mutex = new object;
     private int _currAvail;

     public Semaphore(int capacity)
     {
          _currAvail = capacity;
     }

     public void Wait()
     {
          lock(_mutex)
          {
               if (_currAvail == 0) Monitor.Wait(_mutex);
               _currAvail--;
          }
     }

     public void Signal()
     {
          lock(_mutex)
          {
               _currAvail++;
               Monitor.Pulse(_mutex);
          }
     }
}
```