**W1D2**

1 = 1

1 + 3 = 4

1 + 3 + 5 = 9

1. What is 1 + 3 + 5 + … + (2n -1)?
2. Explain based on the above pattern. This is NOT  a proof. Just a guess.
3. Explain based on some formula you know.
4. Prove using algebra.
5. Prove using geometry.
6. Prove by induction.

**W1D3**

What is the sum? 7 + 12 + 17 + ...+ 1087.

What is the sum? 1 + 1/3 + 1/9 + 1/27 + ...

What is the sum? 1 + 1/2 + 1/4 + 1/8 + …

What is the sum? 1 + 2/3 + 1/3 + 4/27 + ...

What is the sum? a + ax + ax2 + …

What is the sum? a + ax + ax2 + … + axm

**W1D4**

How many ways you can compute (i.e. there is an algorithm) the nth Fibonacci term?

List those ways along with their time complexities.

Based on your listing above, which algorithm is the fastest?

**W1D5**

1. Write down the **recurrence formulas** to compute the Time Complexity for both algorithms given below.
2. Determine the time complexity of both algorithms. You are free to use known formulas, theorems or your own new results supported by algebra.
3. Based on (b), which is the better algorithm?

**Algorithm FindMaxOne(A, low, high)**

if (low ==high) return A[low]

  else

                    mid = ( low + high )/2              //We always do integer division

leftMax = FindMaxOne(A, low, mid)

rightMax = FindMaxOne(A, mid + 1, high)

return Math.max(leftMax, rightMax)

**Algorithm FindMaxTwo(A, low, high)**

if (low ==high) return A[low]

  else

rightMax = FindMaxTwo(A, low + 1, high)

return Math.max( A[low] , rightMax)

**amortCostWorkSheet**

**c** function: 1 to add.

                    3k to resize (if k >0.  Note: k is the size of the “completely filled array”)

**ĉ** function: 7 to add. (customer is willing to pay for add)

                    0 to resize (customer do not want to pay for resizing. It is not his/her concern)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Item #** | **Operation** | **Cost for us** | **Customer paid** | **Profit** | **Balance** |
| 1 | Add | We assume we start with 1 slot. We add 1 item at the cost of 1. | 7 | 6 | 6 |
| 2 | Add | 3 to resize  (We have two slots)  1 to add | 7 | 6 | 3    9 |
| 3 | Add | 6 to resize  (We have 4 slots)  1 to add | 7 | 6 | 3    9 |
| 4 | Add | 1 to add | 7 | 6 | 15 |
| 5 | Add | 12 to resize  (We have 8 slots)  1 to add | 7 | 6 | 3    9 |
| 6 | Add | 1 to add | 7 | 6 | 15 |

**W1D6**

1. Prove  1 + 1/2  + 1/3 + …+ 1/n is W(log n).
2. Prove 1 + 1/2  + 1/3 + …+ 1/n is O(log n).
3. Prove 1 + 1/2  + 1/3 + …+ 1/n is Q(log n).

**W2D1**

**ArrayList : Size quadruple strategy**

Then c(resize) = \_\_\_\_\_\_\_\_\_ and ĉ(add) = \_\_\_\_\_\_\_\_\_\_\_\_\_\_.

Assume one resize Just happened. For example, assume you are left with say 16 size array.

c(resize) 5k

Note that 4 "cells" are already occupied. Thus, you can add 12 more items without another resizing. At the time of next resizing, the real cost is 5\*16 = 80.

The total real cost of all adds + next resizing is = 12 (to add) + (80 to resize) = 12 + 80 = 92..

We need to spread the total cost 92 among all "12 adds" before the resize. This is 92/12 = 8. Hence

ĉ(add) = 8.

ĉ(resize) = 0.

**W2D2**

Prove  by induction 1k + 2k + … + nk = O(nk+1), for k = 1, 2, …

**W2D3**

What is the lower bound of the following problems?

1. Find an item in an unsorted array?
2. Find an item in a sorted array?
3. Given n, print an n x n square?

               Example:

               n = 5

XXXXX

       XXXXX

XXXXX

XXXXX

XXXXX

1. Given n, print all permutations of 1, 2, …, n.
2. Given n integers, sort them.
3. Given n integers, sort them using a comparison based algorithm
4. Given n integers, sort them using inversion bound algorithm.