# University of Colombo School of Computing
# SCS 1308 - Foundations of Algorithms
## Take-home 02

Instructions

- Try the following questions and upload your answer script as a zip file to the given link in the UGVLE on/before 8th December at 6 pm.
- Note: Rename your zip file with your index number and name. (i.e: indexNo_Name.zip).

**01. Write a non-recursive algorithm as a pseudo code to find the sum of the first n odd numbers. Prove the correctness of the algorithm by using the loop Invariant property method and mathematical Induction method**

**02. Two UCSC 1st year students are working on recursive search algorithms and have been studying a variant of binary search called *trinary search*. One has created the following pseudocode for this algorithm:**

```
TSearch (A [a...b], t)
        If a > b
                return -1
        Let p1 = a + Floor((b - a)/3)
        If A[p1] = t
                return p1
        If A [p1] > t
                return TSearch (A[a...p1-1],t)
        Let p2 = a + Ceiling (2(b - a)/3)
        If A [p2] = t
                return p2
        If A [p2] > t
                return TSearch (A [p1+1...p2-1],t)
                Return TSearch (A[p2+1...b],t)
EndTSearch
```

I. State a recurrence relation that expresses the number of operations carried out by this recursive algorithm when called on an input array of size n.

II. One has heard that trinary search is no more efficient than binary search when considering asymptotic growth. Help prove him correct by using induction to show that your recurrence relation is in O(log n) as well.

**A.** Split the tight bound into and upper (big-O) and lower (big-Ω).

**B.** For each bound select a function from O(log n) to use in your proof, like a $\log_2 n$ or a $\log_2 n$ - b. Remember there are typically multiple ways to prove the theorem using different choices of functions.

**C.** Use induction to prove your bound. Include all parts of the proof including base case, inductive hypothesis, and inductive case. Be as precise as possible with your language and your math. Remember it's possible to get stuck at this point if you have selected the wrong function in the last step.

**03. TRI_PHASE_SORT is a sorting algorithm that works by breaking the array into two overlapping parts, sorting each part, and then sorting part of it again to ensure the whole array is sorted.**

I. Prove that the following recursive algorithm actually sorts its input.

```
TRI_PHASE_SORT ( A [0], A[n-1]):
        if n=2 and A [0] > A [1] swap A[0] with A[1]
        else (n>2)
m = ceiling(2n/3) 3
        TRI_PHASE_SORT ( A [0], A[m-1])
        TRI_PHASE_SORT (A[n-m],..., A[n-1])
        TRI_PHASE_SORT (A[0],..., A [m-1])
```

II. If we replace **ceiling** with **floor**, is the correctness proof from part (a) still valid? If so, explain why each statement in the proof remains correct. If not, identify the first incorrect statement and explain why it is false.

III. Formulate a recurrence relation, including the base case(s), for the number of times TRI_PHASE_SORT compares two elements of A. Explain how each part of your recurrence links to the algorithm.

**04. Write a complete answer consisting of three parts: a description of the algorithm, a proof of correctness, and a running time analysis**

We have an n-story building and a series of magical vases that work as follows: there is some unknown level L in the building that if we throw these vases down from any of the levels L, L+1,..., n, they will definitely break; however, no matter how many times we throw the vases down from any level below L nothing will happen them. Our goal in this question is to determine this level L by throwing the vases from different levels of the building.

For each of the scenarios below, design an algorithm that uses asymptotically the smallest number of times we throw a vase (so the measure of efficiency for us is the number of vase throws).

A. **When we have only one vase.** Once we break the vase, there is nothing else we can do.
B. **When we have four vases**. Once we break all four vases, there is nothing else we can do.
C. **When we have an unlimited number of vases**.