# CMPSCI 611 - Advanced Algorithms
## Homework 1

Jennie Steshenko (Collaboration: Emma Strubell)

## Question 1

### Part A

The order of growth of functions is as follows
$g_7 = (n^{1/log_2 n})^3 = 8$,
$g_{10} = (logn)^2$,
$g_1 = 2^{\sqrt{log_2 n}}$,
$g_3 = n(log_2 n)^3$,
$g_9 = n^{1+1/loglogn} \approx n^1.2$,
$g_4 = n^{4/3}$,
$g_{11} = (logn)^{logn}$,
$g_5 = n^{log_2 n}$,
$g_6 = 2^n$,
$g_{12} = \pi^n = 3.14^n$,
$g_8 = n!$,
$g_2 = 2^{2^n}$

### Part B

1. $logf(n)$ *is* $O(logg(n))$ - **True**
   $O(log(g(n))) = log(cf(n)) = logc + logf(n)$
   $\lim_{x\to\infty} \frac{logc+logf(n)}{logf(n)} = \lim_{x\to\infty}(\frac{logc}{logf(n)} + \frac{logf(n)}{logf(n)}) = 0 + 1 = 1$

2. $2^{f(n)}$ *is* $O(2^{g(n)})$ - **False**
   Let us take $f(n) = x$ and $g(n) = 2x$ then:
   $\lim_{x\to\infty} \frac{2^{2x}}{2^x} = \lim_{x\to\infty} \frac{2^x * 2^x}{2^x} = \lim_{x\to\infty} 2^x = \infty$

3. $(f(n))^2$ *is* $O((g(n))^2)$ - **True**
   $O((g(n))^2) = (cf(n))^2 = c^2(f(n))^2$
   $\lim_{x\to\infty} \frac{c^2(f(n))^2}{(f(n))^2} = c^2$

## Question 2

### Proposed Algorithm

The assumptions and notations used in this algorithm:

- Arrays: A, B
- Array sizes: sizeA, sizeB (e.g. the index of the last cell in array A that should be checked)

The Algorithm: The general idea is to look for the $K^{th}$ smallest item by scanning the arrays intermintently, in a manner similar to a binary search.

## Question 3

The algorithm to solve this problem is very similar to the MergeSort algorithm, with one additional variable to count the number of inverted pairs, and one additional line in the algorithm to track the value of the variable. *The suggested algorithm is a modification to an array based algorithm of the MergeSort algorithm on Wikipedia.org*

```
function CountInvertedPairs(intArray intArr, int start, int end)
// if list size is 1  consider it sorted and return it
if sizeOf(intArr) <= 1
   return intArr
// else array size is > 1, so split the array into two subarrays
// The counter of inverted pairs, as global
var integer invertedCount = 0
// The start and end of the left of the subarray
var int leftStart, leftEnd
// The start and end of the right of the subarray
var int rightStart, rightEnd
var integer middle = floor((end - start) / 2)
// recursively call CountInvertedPairs() to further split each
// sublist until subarray size is 1
left = CountInvertedPairs(intArr, start, middle)
right = CountInvertedPairs(intArr, middle + 1, end)
// merge the sublists returned from prior calls to
// CountInvertedPairs() and return the resulting merged sublist
return merge(left right  invertedCount)

function merge(intArray left, intArray right, invertedCount)
var intArray result
// Indexes to each of the arrays - left, right and result
var int leftIndex = 0, rightIndex = 0, resIndex = 0
while sizeOf(left) > 0 or sizeOf(right) > 0
   if sizeOf(left) > 0 and sizeOf(right) > 0
     if left[leftIndex] <= right[rightIndex]
       result[resIndex] = left[leftIndex]
       leftIndex++
       resIndex++
     else
       result[resIndex] = right[rightindex]
       rightIndex++
       resIndex++
       invertedCount = rightIndex - resIndex
   else if sizeOf(left) > 0
     result[resIndex] = left[leftIndex]
     leftIndex++
```

```
    resIndex++
  else if sizeOf(right) > 0
    result[resIndex] = right[rightindex]
    rightIndex++
    resIndex++
    invertedCount = rightIndex - resIndex
end while
return result
```

## Question 4

Assumption: The set S is stored in an array
The Algorithm: At each iteration, a number will be chosen to which other numbers will added in an attempt to achive the sum of x

```
CheckElementsSum(sIndex, eIndex, element, x)

 SumXOf2Elements(S,x)
  MergeSort(S)
```

## Question 5