**Homework 6: implementation and complexity analysis of recursive algorithms.**

**The main goal** of this homework is to get more familiar with recursive algorithms by (1) implementing three recursive algorithms that were discussed in class and (2) analyzing the complexity of two recursive algorithms.

**Submission:**  Your final submission will be one compressed file. This file contains three program files (one header, two .cpp) and one PDF that contains your solution to the two complexity analysis problem.   **(40 points)**

**1. Coding question**: Create an ADT that contains a fixed-size array that holds 20 integers (i.e., i*nt array[20];* ) and the following member functions:  (**30 points**)

* A default constructor that initializes all the elements in this array to 20 random numbers (you can call the [rand() function](http://www.cplusplus.com/reference/cstdlib/rand/) to generate a pseudo-random number).
* A member function that recursively searches the largest number in the array. This function will return the value of the largest number.
* A member function that recursively finds the value of the k-th smallest number in the array, where k is provided by the end-user. **You are required to use the partition-based, recursive algorithm as discussed in class.**
* A member function that implements the recursive QuickSort algorithm to sort the array in increasing order.

You are required to use separate compilation. Specifically, your project will contain one header file, one .cpp file that implements all the member functions, and another .cpp file that contains the main() function. In the main() function, you will include test cases to call and test all your recursive functions.   
  
**2.1 Complexity analysis:** Represent the time complexity of the following recursive algorithm, T(n), as a recurrence equation: **(5 points)**

    int pow\_2( int n ){  
      if ( n==1)   
               return 2;  
     if ( n > 1)   
               return ( 2 \* pow\_2( n-1 )  );  
    }    
  
**2.2  Complexity analysis:** analyze the time complexity of the Top-Down implementation of the MergeSort algorithm on the following wikipedia webpage:  
            <http://en.wikipedia.org/wiki/Merge_sort>  
  
For

TopDownMerge(A[], iBegin, iMiddle, iEnd, B[])

assume (iEnd - iBegin + 1) is n  (that's the size of A), this algorithm will take c1.n + c0 to finish. Please represent the time complexity of TopDownSplitMerge(A[], iBegin, iEnd, B[]) as a recurrence equation.  You don't need to solve this equation.   **(5 points)**