1. Fibonacci numbers and complexity

Fibonacci numbers are defined recursively as follows:
F(0) = 1, F(1) = 1
F(n) = F(n-1) + F(n-2) for n > 1

Write the following methods to compute F(n):
   a) A \( O(2^n) \) method based on the recursive definition
   b) A \( O(n) \) method that uses a loop
   c) A \( O(1) \) method that uses the closed form solution (feel free to look online for the formula)

2. Sorting lists and complexity

Implementing two algorithms (described below) to sort lists of integers implemented as a reference-based lists using the iNode class. Your program must prompt the user to select the length of the list, whether the elements will be entered manually or generated randomly, and the choice of algorithm to use. As the elements are entered they must be stored in a reference-based list (you may consider storing them in the inverse order in which they are entered). After that, the selected method must be called to sort the list, and finally you should display the resulting sorted list and the time it took to sort the list.

Algorithm A starts with the original list of elements and an initially empty sorted list, and repeatedly finds the node that contains the element with the maximum value in the original list and moves it to the sorted list, until the original list is empty

```java
SortListA(L)
    SortedList = empty list
    while L is not empty
        x = node with maximum element in L
        remove x from L
        add x at the beginning of SortedList;
    return SortedList
```

Algorithm B, takes the first element of the list as pivot, and splits the list into three lists; the first list contains the elements that are smaller than the pivot, the second contains the pivot itself, and the third contains the elements that are greater or equal to the pivot. Then the algorithm recursively sorts the first and the third lists, and finally it concatenates the first list, now sorted, the list that contains the pivot, and the third list, also sorted.
SortListB(L)
  If length(L) >1
    let f be item in the first node in L
    remove f from L
    split L into three lists as follows:
      L1 that contains all the elements that smaller than f
      L2 that contains (only) f
      L3 that contains the elements of L that are greater or equal to f
    L1 = SortListB(L1);
    L3 = SortListB(L3)
    L = concatenate(L1,L2,L3);
  return L

What are the running times of algorithms A and B with respect to the length of the lists?

Write a report describing your work. We are particularly interested in your observations about the behavior of each algorithm as the size of the input data increases and also in the comparison of different algorithms for each input size. Use graphs or plots to illustrate this.