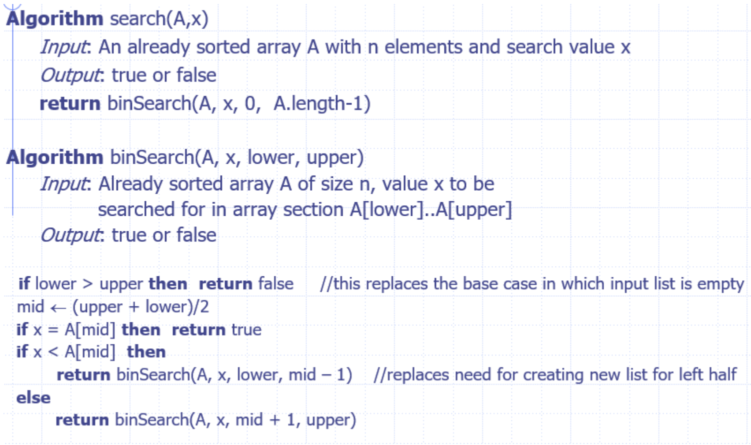
Lab 3B

1. Perform the MergeSort algorithm by hand on the list [7, 6, 5, 4, 3, 2, 1] using a MergeSort recursion tree, as was done in the lecture.

2. The fast BinarySearch algorithm is shown below. We have already shown that the algorithm's worst-case running time is $\Theta(\log n)$ using the Master Formula. For this problem, use the method of counting self-calls to establish this running time.

   Hints. Remember the result about length of descending sequences, discussed in the slides. Remember the worst case for BinarySearch.



```
Algorithm search(A,x)
    Input: An already sorted array A with n elements and search value x
    Output: true or false
    return binSearch(A, x, 0,  A.length-1)

Algorithm binSearch(A, x, lower, upper)
    Input: Already sorted array A of size n, value x to be
           searched for in array section A[lower]..A[upper]
    Output: true or false

 if lower > upper then  return false    //this replaces the base case in which input list is empty
 mid ← (upper + lower)/2
 if x = A[mid] then  return true
 if x < A[mid]  then
     return binSearch(A, x, lower, mid – 1)   //replaces need for creating new list for left half
 else
     return binSearch(A, x, mid + 1, upper)
```

3. Solve the following problem with a recursive algorithm: Given a list with n elements, put the elements of the list in reverse order. Compute the running time of your algorithm (hint: count self-calls).

4. In Lesson 3 slides it was shown that the recursive algorithm fib(n), for computing the nth Fibonacci number, runs in exponential time. Develop two fast alternatives to the fib(n) algorithm:
   - Devise an iterative algorithm that runs in O(n) time
   - Devise a O(n) recursive algorithm, similar to fib(n), that stores results of computations and re-uses them.

   Implement both in code.

5. We showed that the SecondSmallest problem can be solved with an O(n) iterative algorithm. Does the same technique solve the corresponding ThirdSmallest problem in O(n) time? Explain. What if "second" and "third" are replaced by "*k*th", where *k* represents any number from 2 to n-1? For instance, is there a fast algorithm for finding the n/2-smallest element in a list of n elements? Is the sorting approach the fastest way in this case? Explain