# CSE 222 HOMEWORK 2

1. Specify true or not, prove.
2. Order functions
   1. therefore, growth of is less than growth of
   2. therefore, growth of is less than growth of
   3. therefore, growth of is less than growth of
   4. therefore, growth of is less than growth of
   5. therefore same
   6. therefore, growth of is less than growth of
   7. therefore, growth of is less than growth of   
        
      with these results the ordering from less to great is
3. Write time complexity of following functions, printing is assumed as constant time
   1. The variable assignment and increment in for loop parenthesis are θ(1), the modulo (assumed) and equality operations in first if condition are also θ(1), increment of count is θ(1), for else block and since loop goes on until i reaches to n the loop has time complexity, multiplying all these results that function has time complexity.
   2. Assignments before the loop are constant time, assignment and increment inside for loop parenthesis are constant time, there are only getting elements with index which is constant time and checks and assignments in the loop block and all of them are constant time, loop is iterated size of the array times and if that were to be n than it would be iterated n times. The for loop in total has linear time complexity and with sum of assignments before it the complexity doesn’t change. There are no different worst and best case scenarios therefore it can be denoted by theta.
   3. Assuming multiplication is constant time and knowing that getting elements from arrays with index is also constant time, and return statement also being constant time this function has constant time complexity. There are no different worst and best case scenarios therefore it can be denoted by theta.
   4. Assignment of variable sum is constant time, assignment and increment in loop parenthesis are constant time, getting elements and multiplication (assumed) are constant time, increment of sum is also constant time. Loop terminates when i reaches to n and i is incremented with a constant value therefore loop iteration has linear complexity, with multiplying complexities inside it the result is linear complexity. Return statement is also constant time and with sum of all these operations the function’s complexity is linear. There are no different worst and best case scenarios therefore it can be denoted by theta.
   5. The outer loop has linear complexity for i increases at a constant rate, the inner loop has logarithmic complexity since j increases as by being multiplied by 2 every iteration. Getting elements and multiplication are constant time. Multiplying these we have nlogn complexity as function’s complexity. There are no different worst and best case scenarios therefore it can be denoted by theta.
   6. The if condition is linear time since p\_4 is linear time, the if block may or may not be executed, if it’s not executed then it doesn’t affect but if it’s executed than we must sum n and nlogn complexities which results in nlogn complexity. Fort the last line p\_3 is constant time so doesn’t affect whatsoever and p\_4 is again linear time. So, it’s seen that in best case scenario (where if block is not executed) the time complexity is and for the worst case scenario it is . Therefore, function’s time complexity is O(nlogn).
   7. The while loop is logarithmic time since i is halved each iteration. For loop is linear since it is iterated n times and n doesn’t change. Function’s time complexity is multiplication of these two which is nlogn. There are no different worst and best case scenarios therefore it can be denoted by theta.
   8. The while loop itself is logarithmic since n is halved each iteration. For loop depends on n and as mentioned it is halved each while loop iteration so the total number of iterations of this for loop for the whole while loop is where f(n) is number of total iterations (or time complexity) of for loop throughout the while loop life. Therefore function time complexity is linear and can be denoted by theta since there’s no different worst and best case scenarios.
   9. This is a recursive function and depends on n, and n changes at a constant rate. If first condition block is executed it is constant time but if else block is executed then it has to be repeated n-1 more many times therefore linear complexity. Best case is and worst case is . Therefore time complexity of the function is O(n).
   10. If first condition block is matched, then function returns, and this is constant time. If not function calls itself by decrementing n by one each time making this part linear. For the while loop its best case scenario is A[j] being greater or equal to A[j-1] making it constant, and worst case scenario is the inequality satisfies for every value of j and making it linear time. Swapping is assumed as constant time. Therefore, function can be constant time, linear time, or quadratic time (linear by recursion and linear by while loop in every recursion call). Best case is worst case is . Making function O(n^2) complexity.
4. 1. “At least O(n^2)” cannot be said since O(n^2) implies that function may have less time complexity, such as linear.
   2. Prove
      1. contradiction therefore
      2. f(n) = O(n^2) = Ω(1), g(n) = θ(n^2) = Ω(n^2)  
         f(n) \* g(n) = θ(n^4) implies f(n) \* g(n) = O(n^4) = Ω(n^4)   
         but f(n) \* g(n) = Ω(1) \* Ω(n^2) = Ω(n^2) is possible therefore it is false
5. 1. Couldn’t solve it.
   2. Find characteristic roots of T(n):
6. metin içeren bir resim

   Açıklama otomatik olarak oluşturuldu  
   Outer loop has n iteration and inner loop has n iteration at first and one less after each iteration of outer loop. So total iteration (time complexity) of the function is

Time for an array of 1,000 elements (no valid pair is found thus no print):  


Time for an array of 10,000 elements (no valid pair is found thus no print):

  
Since theoretical time complexity is quadratic, for 100,000 elements the execution time should be , and the result is:  
  
Our prediction was in 5.4% error margin, it is because this stopwatch also measures compilation time and I didn’t test many times to find the average execution time, instead I just did the stopwatch measurement several times and used the most frequent outcome.

1. metin içeren bir resim

   Açıklama otomatik olarak oluşturuldu  
   And default i and j values are 0 and 1 in that order. The first if statement enables function to traverse all until the n for the i value. The second if statement enables j to traverse all until the n while i remaining the same. And for the last condition it just checks if elements sum up to given value. I couldn’t measure execution time due to stackoverflow exception caused from too many recursions but i gets to be set to all integer values between 0 to n (array length) and for each i value j gets to be all integer values between i+1 to n so time complexity is the same as it was in the previously shown iterative function.  
     
   For the part “write a recurrence relation and solve it”, I didn’t get it. But if a random recurrence relation is wanted then: