**Problems**

Please write precise and concise answers. Your algorithm descriptions should use either clear, concise, and precise plain English or clear, concise, and precise pseudo-code that uses a style similar to the pseudo-code in your textbook. Submit your solutions to problems **1-5(a)** via [D2L](https://d2l.depaul.edu/) as a Word or PDF file or as scans/photos of legible handwritten notes. Submit your solutions to problems **5(b)** via [Kattis](https://depaul.kattis.com/courses/CSC421/Winter2021).  
  
  
**1.**    Use recursion trees to solve the following recurrences. You may assume that T(1) = O(1).

1. T(n) = T(n/2) + n

Using tree: t(n) -------n

Height of t(n/2) -----n/2

Tree = log2N. t(n/4)--------n/4

T(n)= (n+(n/2)+(n/4)+( [ yellow highlighted text +1 ]) (log 2^n +1) terms

T(n)= [1(2^log2^n+1 -1)]/2-1. = 2n-1. = 0(n)

1. T(n) = 4T(n/4) + n

Using a Recussion tree…

Height of tree = log 4^n T(n) ---------🡪 n

| | | |

T(n/4) T(n/4) T(n/4) T(n/4)🡪 4\*n(4) =N

| | | |

T(n)=(n+n+…+n)log4^n times.

T(n) = nlog4^n = o(nlogn)

1. T(n) = 3T(n/2) + n

Height = log 2^2

T(n) ---n

| | | 🡪 3n/2

T(n/2) T(n/2) T(n/2) 🡪 (3n/2)

| | | | | | | | |

T(N/4) repeated 9 times. Or (9n/4)

T(n)= n[1+(3/2)+(9/4)…]log2^n times.

T(n)= n[1(3/2)^log2n -1) =. 0 (n^ log 2^3)

(3/2) -1

**2.   [Optional]**    Let A[1..n] and B[1..n] be two sorted arrays. We can easily find the middle element (median) in A in Theta(1) time: it's simply A[n//2 + 1] (assuming // is integer division). Similarly, we can find the median in B. Give a O(log n) time divide-and-conquer algorithm to find the median overall. Explain why your algorithm works. (Hint: get inspiration from binary search.)

NA

**3.**    Given an array X of N real numbers we would like to find the maximum sum of entries found in any subarray of X. For instance, if N = 10 and X[1..10]  is

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 31 | -41 | 59 | 26 | -53 | 58 | 97 | -93 | -23 | 84 |

then the answer is 187, which is the sum of entries 59, 26, -53, 58, 97 in the "maximum subarray" X[3..7]. The problem is easy when all the entries are positive -- the maximum subarray is the entire input vector. The rub comes when some of the numbers are negative: should we include a negative number in hopes that the positive numbers to its sides will compensate for its negative contribution? Of course, if the entries are all negative then the maximum subarray is the empty subarray and zero should be returned. **Design an O(n log n) divide and conquer algorithm for this problem and describe it using (the textbook style) pseudo-code**. Explain why your algorithm works and justify your running time analysis.  
  
(Hint: Divide the problem in half and solve the left and right subproblem recursively. This will find the maximum subarrays contained in the left and right subproblems, but it will overlook subarrays that start in the left subproblem and end in the right subproblem. How do you find the maximum such subarray? **You must do this in O(n) time in order to achive the O(n log n) running time for the whole algorithm.)**

two ways to do this:

Var arrayOne = [31, -41, 59, 26, -53, 58, 97, -93, -23, 84 ];

Var arrayTwo = [31, -41, 59, 26, -53];

Var arrayThree = [58, 97, -93, -23, 84];

Function 1: Traverse over each element of a given array.

Or

Function 2: Using the two variables created we can return (use recursive calls on vars 2/3)the max var2, var 3, and the mid point, (divide and conquer).

**4.**    Problem 13 page 51 in your textbook.

An inversion in an array A[1.. n] is a pair of indices (i, j) such that i < j and A[i] > A[j].

The number of inversions in an n-element array is between 0 (if the array is sorted) and n (n/2) (if the array is sorted backward).

Describe and analyze an algorithm to count the number of inversions in an n-element array in O(n log n) time. [Hint: Modify mergesort.]

// Count inversions in an array.

//using java

Class Inversion { //main class

Static int array[] = [1, 2, 3, 4, 5 }; //create an array for example could be 0, 05, //.25, etc. .075,

Static int findInversions (int n) { //function to find the count

Int inversions = 0; //the actual count a variable

For (int i = 0; i < n -1; i ++) //loop to get the count

For (int j = I +1; j <n; J++)// a recursive call

If (arr[i] >arr [j]) //condition

Inv\_count++;

Return inv\_count; //returns result.

}

//tests code… displays to console.

Public static void main(string[] args) {

}

System.out.println (“there are a total of “ + findInversionssions (arr.length));

} }

//when run the total time complexity is 0(log n).

//end of program

**5.**Week 2 problem *batmanacci* on [Kattis](https://depaul.kattis.com/courses/CSC421/Winter2021).

1. Describe the solution for inputs N and K using a recursive formula and analyze the running time of your algorithm

Followed the solution. Used Java and standard OOP practices. Using OOP makes more logical sense. Use 1 main class and 1 method for testing. In the main class, have as many vars as needed then use loops, recursions, and functions (actions) in order to solve the problem. Test in the second part.

    (b)  Implement your solution using your prefered language and submit your implementation via [Kattis](https://depaul.kattis.com/courses/CSC421/Winter2021).