Homework 2

Due by 4:30pm on September 23, 2020

Viewings and Readings

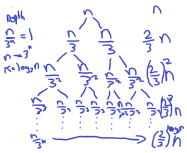
[Links to the slides, recordings, and quiz are on the course web site.]
Review the week 2 lecture slides.
View the week 2 lecture recordings and the discussion session recording.
Complete the week 2 quiz.
Read chapter 1 of the textbook (section 1.8 is optional).

Problems

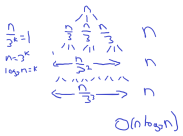
Please write precise and concise answers. Your algorithm descriptions should use either clear, concise, and precise plain English or clear, concise, and precise pseudo-code that uses a style similar to the pseudo-code in your textbook. Submit your solutions to problems **1-4(a)** via D2L as a Word or PDF file or as scans/photos of legible handwritten notes. Submit your solutions to problems **4(b)-5** via Kattis.

1. Use recursion trees to solve the following recurrences. You may assume that T(1) = O(1).

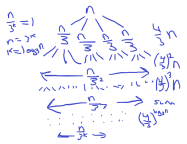
(a) T(n) = 2T(n/3) + n



(b) T(n) = 3T(n/3) + n



(c) T(n) = 4T(n/3) + n



2. Let A[1..n] and B[1..n] be two sorted arrays. We can easily output the k-th smallest element in A in Theta(1) time by just outputing A[k]. Similarly, we can find the k-th smallest element in B. Give a O(log k) time divide-and-conquer algorithm to find the k-th smallest element overall, i.e. the k-th smallest in the union of A and B. Explain why your algorithm works. You can assume that all elements in A and B are different and that k is no greater than n. (Hint: get inspiration from binary search.)

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

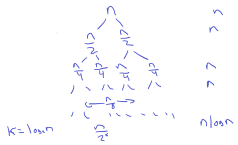
1	2	3	4	5	6	7	8	9	10
31	-41	59	26	-53	58	97	-93	-23	84

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 problemand 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.)

```
Recursively
  base case is only one element in the array, just return that element
  Divide the array in half
  Compare the left sum, the right sum and a combination of the left sum + right sum (So after the base case the next comparison would be the base cas
  The left sum is the resursive return on the left side of the array, the right sum is the recursive return of the right side
  The max of all the recursive calls will be the absolute max of the array

Since each recursive call is n/2 of the previous we know that this is O(log(n)) by drawing out a recursive tree
```



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/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.]

```
Basically do a mergesort but keep track of the number of times the sort needs to rearrange indexes
Divide array in half and recursively run on left half and right half
Keep halfing until the base case of one element in the array
Merge the left half and the right half
While merging check if left index is greater than right, if it is increment a counter
Return the counter from the merge step and add it together with all other recurcive merge steps that were called

This is just a merge sort with an extra linear step of incrementing a counter so we know this is O(nlogn)
```

5. Week 2 problem *batmanacci* on Kattis.

(a) Describe the solution for inputs N and K using a recursive formula or pseudo-code and analyze the running time of your algorithm

```
Since [n] is [n-1] + [n-2] we know if the length of k is greater than the length of [n-2] then k is the [k-length[n-2]] element in [n-1]
We can recursively loop through this logic of shifting the string number and character index until we get the the base case where k = length of strir

def batmanacci(string_number, character_index):
  if string_number == 1:
    return "N"
  elif string_number == 2:
      return "A"

  else:
      while string_number > 2:
          if character_index > fibNumbers[string_number - 2]:
              character_index -= fibNumbers[string_number - 2]
              string_number -= 1
          else:
              string_number -= 2
          return batmanacci(string_number, character_index)
```

(b) Implement your solution using your prefered language and submit your implementation via Kattis.

6. **[Optional]** Week 2 problem *closestpairs2* on Kattis. Submit your solution via Kattis. NOTE: Unfortunately, the CPU time limit for this problem is very tight and penalizes slower executing languages... My Python solution got a "Time Limit Exceeded" error whereas a C++ solution passed even though they both used the same algorithm. So, either use a "fast language" like C++, C, Rust, ... or be willing to accept that the judge will reject your correct solution...