

Homework 3: Due February 14 (11:55 a.m.)

Instructions

- Answer each question on a separate page.
- Honors questions are optional. They will not count towards your grade in the course. However you are encouraged to submit your solutions on these problems to receive feedback on your attempts. Our estimation of the difficulty level of these problems is expressed through an indicative number of stars ('*' = easiest) to ('*****' = hardest).
- You must enter the names of your collaborators or other sources as a response to Question 0. Do NOT leave this blank; if you worked on the homework entirely on your own, please write "None" here. Even though collaborations in groups of up to 3 people are encouraged, you are required to write your own solution.

Question 0: List all your collaborators and sources: ($-\infty$ points if left blank)

Question 1: (20 points)

This problem involves coding. Please make sure to:

1. Sign the consent form (click here) to use HackerRank.
2. Create an account at HackerRank and join the contest (click here).

For credit on the coding part, please list your HackerRank username here: _____.

Question 2: (10 points)

Naively, in order to compute 2022^n given input n , we need to perform $n - 1$ many multiplications. Give an algorithm that takes as input n and computes 2022^n using only $O(\log_2 n)$ many multiplications. Prove the correctness and complexity (number of multiplications used) bound of your algorithm. (Hint: consider the case when n is a power of 2, i.e. $n = 2^k$. Then, extend it to the general case).

Question 3: (10 points)

A sorting algorithm is *stable* if objects with equal keys appear in the same order in the sorted output as in the unsorted input. Formally, if $A[]$ is the unsorted array, for any two indices i and j , $i < j$, such that $A[i].\text{key} = A[j].\text{key}$, their final positions i' and j' in the output of Merge Sort satisfy $i' < j'$. For example consider a list of Student Objects, with two fields, Student.name and Student.age. Now given a list of Student objects, a stable sort with respect to age would order the list in increasing order (say) of the student ages but if two students have the same age, they will appear in the same order as in the unsorted list. Prove that Merge Sort (as seen in class) is stable.

Question 4: (2+3+5+5+5=20 points)

Suppose you are given a list A of n distinct numbers. You are guaranteed that this list is ‘close to sorted’ in the following sense: if A_{sorted} denotes the list fully sorted in increasing order, then A_{sorted} differs from A only at $\log_2 n$ many positions. Intuitively, this says that at most $\log_2 n$ elements are out of place in the original list. For instance, in the list

$$(2, 5, 9, 11, 20, 14, 15, 12, 25, 30)$$

only two elements are out of place (20 and 12), since after sorting, we get

$$(2, 5, 9, 11, 12, 14, 15, 20, 25, 30) .$$

The following exercises will ultimately lead to an algorithm which sorts A in time $O(n)$ ¹. Answer each exercise. **Don’t forget to read the hints!**

1. (Warm up) Prove that the leftmost (first) out of place element must be too big (not too small) for its place.
2. Suppose we construct a stack S as follows: We go through A from left to right, pushing elements one by one into S as long as the next element from A is larger than the top element s of the stack so far. If at any step i the element $A[i]$ which we are currently considering in A is smaller than s , we instead pop s from S and continue. This idea is described in the pseudo code below:

Algorithm 1 Construct increasing subsequence S

Input: A and an empty stack S .

Output: S representing an increasing subsequence of A

```

 $n \leftarrow \text{len}(A)$ 
for  $i = 1$  to  $n$  do
  if  $S$  is empty or  $S.\text{topElement}() \leq A[i]$  then
     $S.\text{push}(A[i])$ 
  else
     $S.\text{pop}()$ 
  end if
end for

return  $S$ 

```

If A is initially $(2, 5, 9, 11, 20, 14, 15, 12, 25, 30)$, what will S be after Algorithm 1 has run?

3. Prove that S is an increasing subsequence of A (the elements of the output stack are increasing from bottom to top).
4. Prove that S contains all but at most $2 \log_2 n$ of the elements of A . (Hint: show that every time we leave out a pair of elements (the “else” case) at least one of them must have been an out of place element.)

¹Note that sorting algorithms take $O(n \log n)$ time without any assumptions on the inputs. Here, we are able to get the faster $O(n)$ run-time by *assuming* that the input array A is already close to being sorted.

5. Use the previous statements to design and prove the correctness and run-time of an algorithm to sort the nearly sorted array in time $O(n)$. (Hint: Suppose the out of place elements could be separated from the rest of A which is sorted (This is what parts 3 and 4 show). Then since there are only a few of them they can be sorted quickly and the rest of the elements are in S which is already increasing. Now think of the “Merge” subroutine in Merge Sort.)

Question 5: (2+2+4+2=10 points)

In the lecture (and previous homework) we looked at using Recursion Trees to solve recurrence relations. However, there is a simplified result called Master Theorem, for a limited class of recurrence relations. An expansive discussion on this can be found in CLRS, Section 4.5.

Theorem 1 (Master Theorem). *Let $a \geq 1$ and $b \geq 1$ be constants, let $f(n)$ be a function, and let $T(n)$ a function defined on non-negative integers by the recurrence.²*

$$T(n) = aT(n/b) + f(n) .$$

Then, $T(n)$ has the following asymptotic bounds:

- (a) If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
- (b) If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$.
- (c) If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, **and** $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n , then $T(n) = \Theta(f(n))$.

The proof of this theorem is immediate by applying recursion tree approach to solving the recurrence relation but we omit it for sake of simplicity.

We now illustrate a simple example of how you can use the Master theorem. We want to solve the recurrence

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

- Looking at the theorem statement: $a = 2, b = 2$ and $f(n) = n$.
- We observe that $\log_b a = \log_2 2 = 1$. So $n^{\log_b a} = n$.
- Since $f(n) = n = \Theta(n)$, we are in Case (b).
- So $T(n) = \Theta(n^{\log_b a} \log n) = \Theta(n \log n)$.

The fancy language with the ϵ is a mathematical way of formalizing the following idea: Take the function $f(n)$, and take the function $g(n) = n^{\log_b a}$. We can only apply Case (a) or Case (c) if functions f and g are “polynomially” larger than each other. If f is polynomially smaller than g , apply Case (a). If f is polynomially larger than g apply Case (c). For example, if $f(n) = n^{1.0001}$ and $g(n) = n^{\log_b a} = n$, then they can be compared and it would satisfy Case (c), but $f(n) = n \log n$ and $n^{\log_b a} = n$ they would not satisfy this formulation of the Master Theorem, and you cannot apply it.

Read the above preamble before you proceed. We promise it will be useful!

For each of the following recurrence relations, solve by Master Theorem. State and justify which case of the Master Theorem is applied. If Master Theorem cannot be applied, justify why it cannot be applied.

²For simplicity, we ignore issues of divisibility, and assume that n/b is an integer.

1. $T(n) = 4T(\frac{n}{2}) + n \log n$
2. $T(n) = 4T(\frac{n}{2}) + n^2$
3. $T(n) = 4T(\frac{n}{2}) + n^2 \log n$
4. $T(n) = 4T(\frac{n}{2}) + n^3$

Honors Questions

Question 6: Honors

(*) We have just seen how to sort in linear time when there are only $O(\log n)$ elements out of place in the original list. But can it be done even if more than $O(\log n)$ elements are out of place? What is the largest number of out of place elements you can handle while still maintaining linear running time? (Hint: Does the method we developed in the homework *require* that only $O(\log n)$ elements are out of place? What happens if there are more elements out of place? What is the run-time if there are k out of place elements?)

Question 7: Honors

Recall the closest pair algorithm. In class you have seen the algorithm to find the closest pair of points in a plane.

1. (***) Consider the 3-dimensional version of this problem: given n points in \mathbb{Z}^3 we want to find the closest pair of points. Try to use the ideas seen in class to develop an algorithm to solve this problem as efficiently as possible.
2. (***) Can you find an $O(n \log^2 n)$ time algorithm?
3. (****) What about an $O(n \log n)$ time algorithm?
4. (*****) In fact there is an algorithm which solves this problem for any constant size dimension d in time $O(n \log n)$!

You can look at these notes for ideas (preferably after trying the problem): <https://sites.cs.ucsb.edu/~suri/cs235/ClosestPair.pdf>