Algorithms HW11

worked w/
Emily Wagner
Carbin Rosset

Jose Niño

Problem 1.

a) let A & B be two sorted arrays of length n elements each.

1. Find median m_a of array A and Median m_b of array B. call every element in A less than m_a A_1 and everything greater than m_a A_2. Call everything in B less than m_b B_1 and everything greater than m_b B_2.
2. If $m_a = m_b$ then return m_a
3. Else if size of A = size of B = 2.
   Sort the for elements (which is constant time) add throw A_1 + throw B_1 and add throw A_2 + throw B_2.
   throw away two elements either from the left end, right end or one from each end such that the addition of $A_1 + B_1$ = Addition of $A_2 + B_2$
   There are only 2 items remaining, return the smaller of the two.
3. Else if $m_a < m_b$
   Then the median cant be in A_1 or B_2 so recurse on A' = from m_a to the end of A. And B' = from the beggining of B to m_b. Add the size of A_1 to throw A_1 and size of B_2 to throw B_2
4. Else if $m_b < m_a$
   Then the median cant be in A_2 or B_1 so recurse on A' = from the beggining of A up to m_a. And B' = from m_b to the end of B. Add the size of A_2 to throw A_2 and sized B_1 to throw B_1.

On every recursion of the algorithm, on each of the two arrays we are throwing away half of the array which means that we can represent this by the recurrence relation

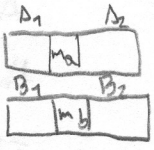
$$T(n) = 2T\left(\frac{n}{2}\right) + 2O(1)$$
(finding M_a and m_b)

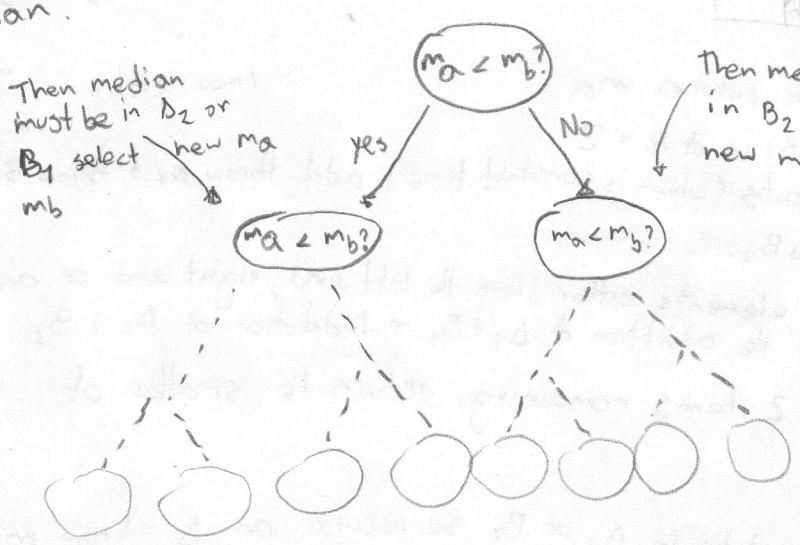
that is $T(n) = O(\log(n))$

b) This problem can be represented by the idea of we are trying to find a specific element in 2n possible elements.

So lets suppose our algorithm gives us the median of array A and of array B and we compare them. This comparison enables us to reduce the number of possible elements and then ask the question again.

At the end of the decision tree we will have found a single element. Moreover, based on the problem this tree has 2n leaves, that is all the possibilities for our median.





The leaves are numbers, that could be the medians and by following one path of the tree we get to one answer.

There are 2n leaves because each original array has n elements

This is a binary tree so depth of tree is $\geq \log(\#\text{leaves})$

Therefore at the very least we have to make $\log(2n)$ comparisons to get to any given leaf.

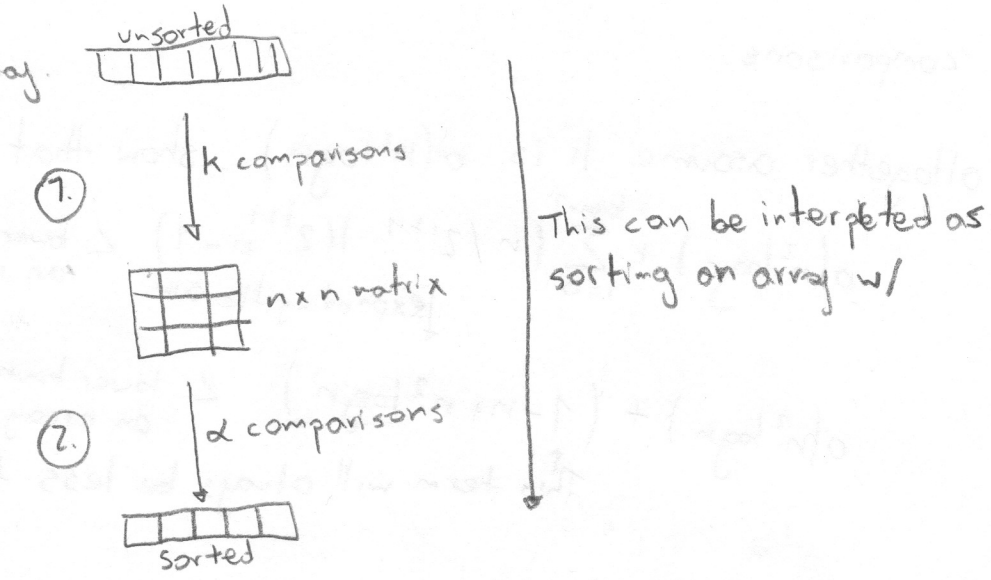
Therefore Number of steps for finding the median of two arrays $\geq \log(2n) = \Omega(\log(n))$

Problem 2.

Jose Nino
HW II

let us have an array A of size n^2 that is unsorted
want to build an $n \times n$ matrix in which all rows and columns are sorted.

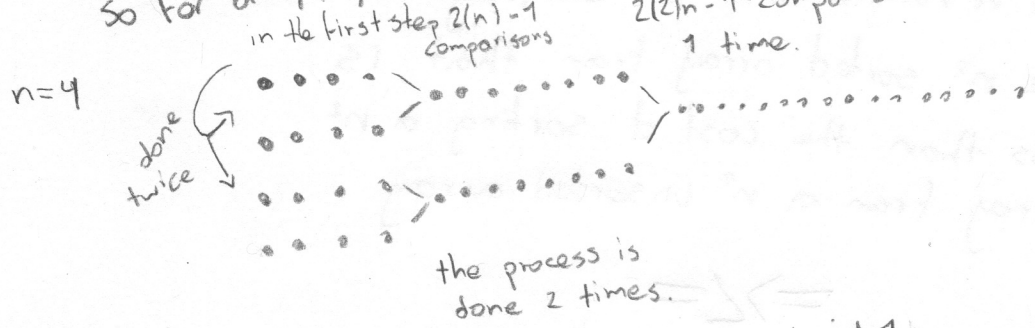
if we have an unsorted array



1. we want to show that getting from the unsorted array to the matrix takes $\Omega(n^2 \log n)$, it is given by the problem that this process is $O(n^2 \log n)$ time. That is, to show Ω we could show that the process CANT be $o(n^2 \log n)$ that is showing that $o(n^2 \log n) + \alpha$ would imply that we can sort the array in less that the lower bound for sorting making it clear that assuming $k = o(n^2 \log n)$ led us to a contradiction.

2. Thus we need to calculate the number of comparisons needed in α we can merge two arrays of size n using $2n-1$ comparisons.

so for a 4×4 matrix
in the first step $2(n)-1$ comparisons
$2(2)n-1$ comparisons 1 time.



so at the ith level we are merging $n/2^{i+1}$ times. and we are performing $2(2^i)n-1 = 2^{i+1}n-1$ comparisons.

that is, to merge an $n \times n$ matrix into a sorted n^2 array it takes

$$\sum_{i=0}^{\log n - 1} (n/2^{i+1})(2^{i+1} n - 1)$$

comparisons.

let $m = n^2$
$m! > (\frac{m}{e})^n \Rightarrow \log(m!) > m \log(\frac{m}{e})$
$> m \log(m) = n^2 \log n^2 = 2n^2 \log n$
$> n^2 \log n$.

altogether assume k is $o(n^2 \log n)$ show that

$$o(n^2 \log n) + \sum_{i=0}^{\log n - 1} (n/2^{i+1})(2^{i+1} n - 1) < \text{lower bound of sorting an } n^2 \text{ array} = cn^2 \log n$$

↓ expanding the sum

$$o(n^2 \log n) + (1 - n + n^2 \log n) < \text{lower bound of sorting an array.}$$

↑ This term will always be less than $cn^2 \log n$ for $n > n_0$

So we can state

$$o(n^2 \log n) + o(n^2 \log n) < \text{lower bound of sorting an } n^2 \text{ array} = cn^2 \log n.$$

and from the identities of little o.

$$o(f) + o(f) = o(f)$$

So,

It must be that the cost of creating the $n \times n$ matrix and then creating the n^2 sorted array from that is less than the cost of sorting a n^2 array from a n^2 unsorted array

$$\Rightarrow \not< =$$

Because if $k + \alpha <$ the lower bound of sorting then it must be that we can sort an array of size n^2 in less than the lower bound for the

Comparison based model.

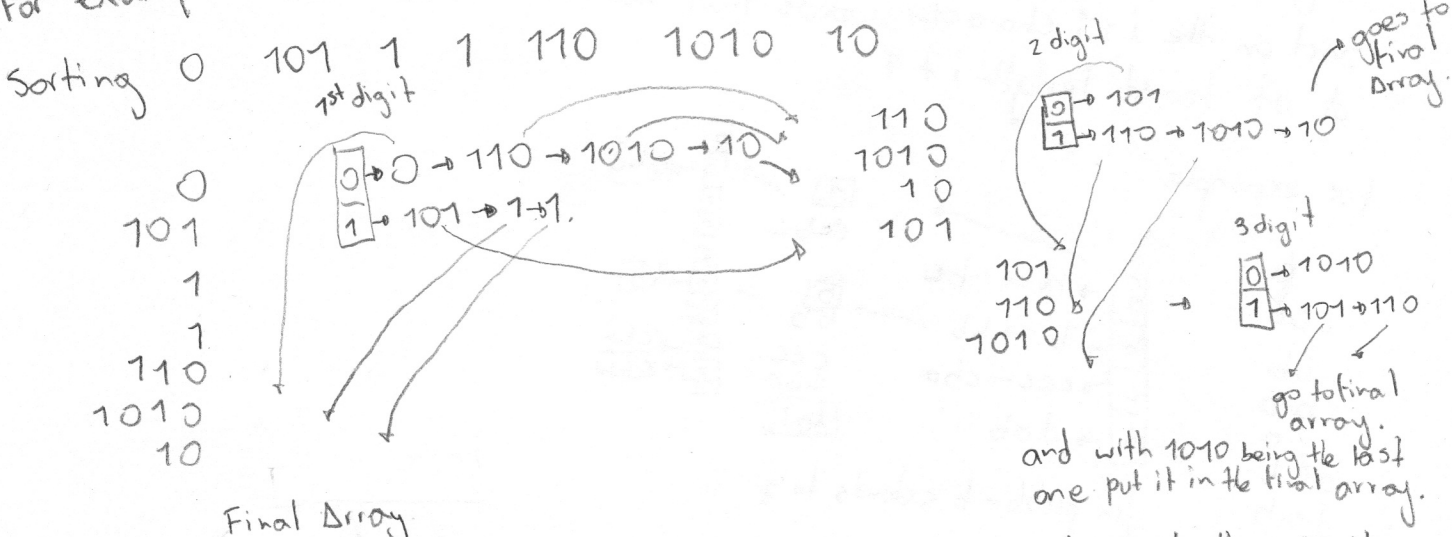
Thus it must be that k cant be done in $o(n^2 \log n)$ time so it must be that $k = \Omega(n^2 \log n)$

Problem 3

a) Here we can use a modified version of Least Significan radix sort to sort this array of integers in $O(n)$ time where n is the total number of bits over all integers.

0. Starting at $i=1$
1. Perform a stable sort on the ith bit of all integers with at least i bits.
2. Increase i and perform the same sort.

So for example



Final Array

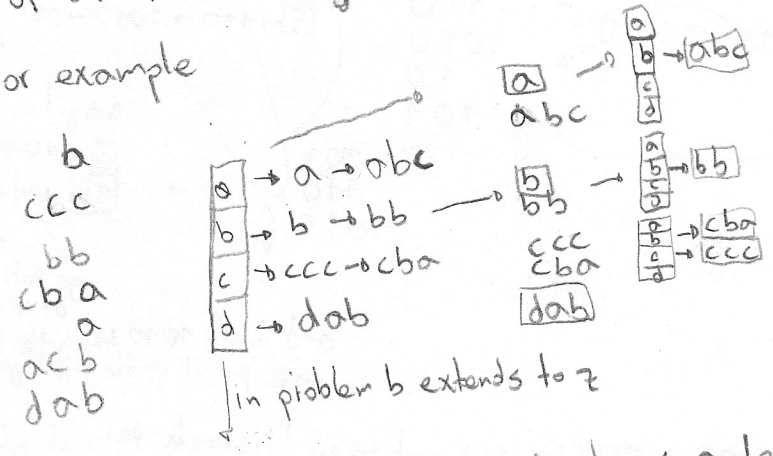
$0 \to 1 \to 1 \to 10 \to 101 \to 110 \to 1010$ did not need to check this digit.

So it is the case with this algorithm that the worst case is where all of the integers have the same number of digits in which case we would need to perform max number of digits stable sorts. Thus we would need to inspect- max number of digits $\cdot$ number of entries = total number of bits that is in the worst case this algorithm makes $O(n)$ inspections to sort the array.

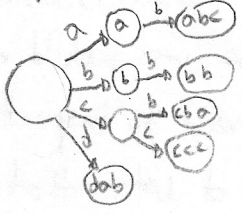
b) At first glance this problem seems fairly similar to the problem in part a). However the root of the difference is that while with numbers any given i-bit integer is < any given j-bit integer if $i<j$, this is not the same for strings in lexicographic order because $a < ab < b$

Therefore it is reasonable to attack this problem from the most significant character.
start at $i = 1$
For any given set of words order them by a stable sort them by their i^{th} most significant character if the word is of at least length i
Sort on the $i+1$ character words that have the same i character and are of at least length $i+1$

for example



so this process could be imagined as a lexicographic trie



so inspired by the trie representation it is not hard to see that the maximum number of connections in the trie is the amount of all the characters over all strings in the original array, because if there were more we would end up with a word that was not in the original array. So in the worst case we need to inspect all the characters in all the strings of the array, making this algorithm run in $O(n)$ time.

n