CS 344 Problem Set 2: Divide and Conquer Algorithms

Due : (July 20, 2023 11:55 pm)

The objective of this assignment is to get you to work on developing divide and conquer algorithms to solve problems. You must attempt all problems, even if you are unable to completely solve them.
You may discuss these problems with your classmates - and you are strongly encouraged to discuss them with me.

1. Let $T(n) = T(an) + T(bn) + n$ for some constants $0 < a \leq b < 1$. Show that :
   (a) $T(n) = \Theta(n)$ if $a + b < 1$
   (b) $T(n) = \Theta(n \log n)$ if $a + b = 1$
2. (Merging sorted lists) Suppose you are given k sorted arrays, each having n elements (integers). The objective is to combine all these sorted arrays to form one large sorted array having kn elements. Let's call the sorted arrays $A_1, A_2, ..., A_k$ for convenience.
   (a) Iteratively use the merge procedure to first merge A_1 and A_2 (let's call the result B_1). Then continue by merging B_1 and A_3 to form B_2. Repeat this process until all arrays have been merged to get a kn sized sorted array. What is the time complexity of this algorithm?
   (b) Using divide and conquer, provide a more efficient algorithm to merge the k arrays. What is the time complexity of your algorithm?
3. (Median from sorted halves) You are given as input two *sorted* arrays A and B having $\frac{n}{2}$ elements (integers) each. Consider the task of finding the $\frac{n}{2}$-th smallest element in the union of A and B. For example, if the input arrays are [1 3 5 7 9 11] and [2 26 48 82 100 164] then the answer must be 9.
   (a) Give an $O(n)$ time algorithm to solve this problem. Justify correctness and the time taken by your algorithm.
   (b) Using divide and conquer, show that this problem can actually be solved in $O(\log n)$ time .
4. (Majority element) An array A of n elements is said to have a majority element if strictly more than half of its entries are the same. For example, the array [13 23 13 47 31 13 13] has a majority element - 13 whereas the array [20 40 20 13 7 40] has no majority element. On input array A provide algorithms according to the following requirements that either output the majority element (if one exists) or returns that none exists.
   (a) Show that the frequencies of all elements of the array can be computed in total time $O(n^2)$. Use this to provide an $O(n^2)$ time algorithm to solve this problem.
   (b) Next, show that you can actually compute the frequencies of all elements in time $O(n \log n)$. Use this to provide an $O(n \log n)$ time algorithm to solve this problem.
5. (Range queries) Consider an array A having n integers. We define middle half of A to be the elements of A having ranks in the range $[\frac{n}{4}, \frac{3n}{4}]$ (both inclusive). For example, the middle half of the array [20 13 50 2 38 21 -3 59] is the set $\{20, 13, 2, 38, 21\}$.
   (a) Provide an $O(n \log n)$ algorithm that returns the middle half of a given input array A of n integers.
   (b) Show that there is a more efficient algorithm that returns the middle half in time $O(n)$. (It may be helpful to note that the middle half need not necessarily be returned in a sorted order)

6. (**Extra Credit**) (Out of order pairs) You are given an array A of n integers. We call a pair of elements (a_i, a_j) to be out of order if $i < j$ and $a_i \geq a_j$. For example, the array [20 13 50 2 38 21] has 7 out of order pairs - (20,13), (20,2), (13,2), (50,2), (50,38), (50,21) and (38,21).

   (a) Provide an $O(n^2)$ algorithm that counts the number of out of order pairs in A.

   (b) Using Divide and Conquer technique give a more efficient algorithm that counts the number of out of order pairs. Try to achieve the following recurrence for the running time of your algorithm :

$$T(n) = 2T(\frac{n}{2}) + O(n \log n)$$