Practice Problem Set: Divide & Conquer

Problem 1. Given a sorted array of distinct integers $A[1...n]$, you want to find out whether there is an index i for which $A[i] = i$. Give a divide and conquer algorithm that runs in $O(\log n)$ time to find out an index i if it exists.

Problem 2. Suppose you are choosing between the following 3 algorithms:

- Algorithm A solves the problem of size n by dividing it into 5 subproblems of size $n/2$, recursively solving each subproblem, and then combining the solutions in linear time.
- Algorithm B solves the problem of size n by recursively solving two subproblems of size $n-1$ and then combining the solutions in constant time
- Algorithm C solves the problem of size n by dividing it into nine subproblems of size $n/3$, recursively solving each subproblem, and then combining the solutions in $O(n^2)$ time.

What are the running times of each algorithm and which would you choose and why?

Problem 3. Assume that $n = 2^k$ for some positive integer k. Using induction prove that if $T(n)$ is given as follows, then $T(n) = n \log n$.

$$T(n) = \begin{cases} 2 & \text{if } n = 2 \\ 2T\left(\frac{n}{2}\right) + n & \text{if } n > 2 \end{cases}$$

Problem 4. Consider the following recurrence equation, defining a function $T(n)$:

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n-1) + n & \text{if } n > 1 \end{cases}$$

Show, by induction, that $T(n) = \frac{n(n+1)}{2}$

Problem 5. Assume you have an array $A[1..n]$ of n elements. A majority element of A is any element occurring in more than $n/2$ positions (so if $n = 6$ or $n = 7$, any majority element will occur in at least 4 positions). Assume that elements cannot be ordered or sorted, but can be compared for equality. (You might think of the elements as chips, and there is a tester that can be used to determine whether or not two chips are identical.) Design an efficient divide and conquer algorithm to find a majority element in A (or determine that no majority element exists). Aim for an algorithm that does $O(n \log n)$ equality comparisons between the elements. A more difficult $O(n)$ algorithm is possible, but may be difficult to find.

Problem 6. Suppose you have an unsorted array A of all integers in the range 0 to n except for one integer, denoted the *missing number*. Assume $n = 2^k - 1$. Design a $O(n)$ Divide and Conquer algorithm to find the missing number. Argue (informally) that your algorithm is correct and analyze its running time and space requirements.

Problem 7. Given a binary string S of type$\{1^m 0^n\}$, devise an algorithm that finds the number of zeroes in $O(\log k)$ time. Let $m + n = k$

Problem 8. The Tower of Hanoi is a classic puzzle invented by the French Mathematician Edouard Lucas in 1883. In this puzzle there are 8 disks, initially stacked in decreasing order of size one of three pegs. We have to move all of the disks to another peg while following these two rules

- Only one disk can be moved at a time
- A larger disk can't be put on top of a smaller disk

Design an algorithm to solve this problem and analyze the running time of your algorithm. *Hint :* Think how you can solve the problem if there were only 2 disks instead of 8 and see if you can build from that.

Problem 9. [**K-way Merge**] Suppose you have k sorted arrays $A_1, A_2, ...A_k$ each with n elements. You want to combine them into a single sorted array of size kn. One way to do this would be to use the merge operation we discussed in class. First merge arrays A_1, A_2 then merge the result with A_3 and so on

- Figure out how many steps this algorithm would take.
- Design a better algorithm for this problem.

Problem 10. [**Fibonacci Numbers**]
Given below is a recursive algorithm for finding the n_{th} Fibonacci number

Algorithm 1 :

```
function FIB(n)
    if n == 0 or n == 1 then
        return 1
    else
        return FIB(n-1) + FIB(n-2)
```

- Analyze the running time of this algorithm
- A lot of computation seems to be repeated over and over in this algorithm. For example $fib(4)$ would compute $fib(3)$ and $fib(2)$ and then the called $fib(3)$ would also compute $fib(2)$. Is there a way we could avoid computing the same thing over and over again while still using a recursive approach?

Problem 11. [**GCD**]
We studied the following recursive algorithm for finding the gcd of two numbers p and q in discrete mathematics.

Algorithm 2 :

```
function GCD(p,q)
    if q == 0 then
        return p
    else
        return GCD(q, p mod q)
```

It turns out that the worst case input for this algorithm is $p = fib(n)$, $q = fib(n-1)$. Analyze the running time of the algorithm for this input.

Hint : Recall the identity $fib(n) = fib(n-1) + fib(n-2)$. Also note that $fib(n) \leq 2fib(n-1)$

Problem 12. Given the following recurrence relations, find the running time (in big-O notation) using the recursion tree method, substitution method and the Master Theorem.

- $$T(n) = \begin{cases} 8T\left(\frac{n}{2}\right) + \Theta(n^2) & \text{if } n \geq 2 \\ 1 & \text{else} \end{cases}$$
- $$T(n) = \begin{cases} 3T\left(\frac{n}{4}\right) + n\log n & \text{if } n \geq 2 \\ 1 & \text{else} \end{cases}$$
- $$T(n) = \begin{cases} 2T\left(\frac{n}{4}\right) + \sqrt{n} & \text{if } n \geq 2 \\ 1 & \text{else} \end{cases}$$

Problem 13. Flavius Josephus was a famous historian of the first century. During the Jewish-Roman war, he was captured along with 40 other jewish rebels by the Romans. The rebels preferred suicide to capture and so decided that they would form a circle and kill every third person in the circle until no one was left. Josephus and one of his friend didn't want suicide so Josephus quickly calculated where the two should stand in the circle so that they would be the last one's left.

- Calculate where Josephus and his friend would have stood.
- Come up with a more general solution where instead of 41, there are n people in the circle.

Problem 14. Suppose that you somehow know that the number of inversions in an array of size N is 10 where $10 \ll N$. Which of the following sorting algorithms should you use to sort this array completely: Merge Sort, Selection Sort, Insertion Sort. Give an appropriate reason for your choice.

Problem 15. Suppose you are given an array X of m integers, devise an efficient algorithm to find the sum of the total number of inversions in all sub-arrays of X of length l. *Hint:* you should find the number of inversions in each of the $(m - l + 1)$ sub-arrays of length l and simply add them.

Problem 16. Given a series of line segment, $l_i = (x_i, y_i)$ where x_i is the start point and y_i is the end point and i ranges from $1, 2, 3, \cdots, n$, find the number of pairs i, j such that $i > j$ and l_j completely contains l_i. For example, $j = 1, i = 3, l_1 = (2, 10)$ and $l_3 = (3, 5)$. Here, $i > j$ and l_j completely contains l_i. Devise an efficient algorithm for this problem, better than $\mathrm{O}(n^2)$.

Problem 17. A bank confiscates n bank cards suspecting them to be involved in fraud. Each card corresponds to a unique account but an account can have many cards corresponding to it. Therefore, two bank cards are equivalent if they belong to the same account. The bank has a testing machine for finding out if two cards are equivalent.
They want to know that among the collection of n cards, are there more than $n/2$ cards that are equivalent. The only operation that the machine can do is to select two cards and tests them for equivalence. You are required to come up with a solution to this using only $\mathrm{O}(n \log n)$ invocations of the testing machine.

Problem 18. Prove the correctness of the following algorithm for incrementing natural numbers. Analyze the algorithm to find out how many times is line 3 executed in the worst case.

Algorithm 3 : Increment Natural Numbers

```
function INCREMENT(x)                                   ▷ Returns x + 1
    if x mod 2 == 1 then
        return 2 × INCREMENT(⌊x/2⌋)
    else
        return x + 1
```

Problem 19. Prove the correctness of the following recursive algorithm for the multiplication of two natural numbers is correct, $\forall$ integer constants ≥ 2. Analyze this algorithm to find out how many times line 5 executed in the worst case.

Algorithm 4 : Multiply Two Numbers

```
function MULTIPLY(x,y)                                  ▷ Returns product xy
    if y == 0 then
        return 0
    else
        return MULTIPLY(cx, ⌊y/c⌋) + x(y mod c)
```

Problem 20. Consider an $n - node$ complete binary tree T, where $n = 2^d - 1$ for some d. Each node v of T is labeled with a real number x_v. You may assume that the real numbers labeling the nodes are all distinct. A node v of T is a local minimum if the label x_v is less than the label x_w for all nodes w that are joined to v by an edge.
You are given such a complete binary tree T, but the labeling is only specified in the following implicit way: for each node v,you can determine the value x_v by probing the node v. Show how to find a local minimum of T using only $O(\log n)$ probes to the nodes of T.

Problem 21. An investment company has stock prices for n consecutive days. That is for each day, they have a specific stock price. They want to know on which day they

should buy a stock and sell it on some other day so that they make the maximum profit out of it.

Hint: Think of it as an array $A[0..n]$ of distinct integers, you need to find two indices i and j such that $(j > i)$ and $A[j] - A[i]$ is maximum. Design a $O(n \log n)$ algorithm for this problem.

Problem 22. Suppose you are given an array A with n entries, with each entry holding a distinct number. You are told that the sequence of values $A[1], A[2], ..., A[n]$ is unimodal. For some index p between 1 and n, the values in the array entries increase up to position p in A and then decrease the remainder of the way until position n. (So if you were to draw a plot with the array position j on the x-axis and the value of the entry $A[j]$ on the y-axis, the plotted points would rise until x-value p, where they'd achieve their maximum, and then fall from there on.) You'd like to find the "peak entry" p without having to read the entire array in fact, by reading as few entries of A as possible. Show how to find the entry p by reading at most $O(\log n)$ entries of A.