
1: Divide and Conquer

Solve the Following Problems:

1. Use the Master Theorem to solve the following recurrences (see Lecture). If you cannot get a Θ bound via the Master Theorem directly, give a reasonable upper bound in big- O notation. You do not have to convert logarithms to decimals, though you should understand logarithms well enough that you can compare them to small integers like 1, 2, or 3 to figure out which is larger.

(a) $T(n) = 4T(\frac{n}{4}) + 5n$

(b) $T(n) = 4T(\frac{n}{5}) + 5n$

(c) $T(n) = 4T(\frac{n}{2}) + n^2$

(d) $T(n) = 9T(\frac{n}{2}) + n^2 \log n$

2. For the following divide-and-conquer problems, write out the recurrence for runtime and solve using the Master Theorem.

(a) Algorithm A solves a problem of size n by dividing it into 4 subproblems, each of size $n/3$, solving each subproblem, and then combining the solutions in linear time.

(b) Algorithm B solves a problem of size n by dividing it into nine subproblems, each of size $n/2$, solving each subproblem, and then combining the solutions in quadratic time.

3. With sorting, the primitive operation you have is " $<$ ", which tells you which of two items is smaller. This only works when you have an ordering.

Let's consider a scenario where the primitive operation is equivalence " $\equiv$ ", which has the transitive property that if $A \equiv B$ and $B \equiv C$, then $A \equiv C$. For example, maybe we say that all months are equivalent to each other, and all colors are equivalent to each other. Then $\text{NOVEMBER} \equiv \text{JANUARY}$, and $\text{RED} \equiv \text{BLUE}$, but $\text{RED} \not\equiv \text{NOVEMBER}$.

We are given n objects and an equivalence relationship on those objects. Construct an algorithm to determine whether more than $n/2$ of those objects are equivalent; i.e. is there a majority element?

(a) What is an $O(n^2)$ solution to the problem? (You may assume that each equivalence invocation takes $O(1)$ time.)

(b) Here is a first attempt at an $O(n \log n)$ solution (that doesn't work). Use divide and conquer as such: split the list in half, then try to find the majority element by continuing to split the list in half until there are only two items left in each list, whereupon you can find the majority element by just testing if the two items are equivalent. Merge together by checking if the majority element in each sublist is the same. Why would this not work?

(c) How can we use divide-and-conquer to develop an $O(n \log n)$ solution?

2: Solutions

1. Asymptotics:

- (a) $T(n) = \Theta(n \log n)$
- (b) $T(n) = \Theta(n)$
- (c) $T(n) = \Theta(n^2 \log n)$
- (d) $T(n) = O(n^{\log_2 9})$

2. Recurrence relation and runtime:

- (a) $T(n) = 4T(\frac{n}{3}) + 5n \implies T(n) = \Theta(n^{\log_3 4})$
- (b) $T(n) = 9T(\frac{n}{2}) + n^2 \implies T(n) = \Theta(n^{\log_2 9})$

3. Majority elements

(a) What is an $O(n^2)$ solution to this problem?

- Keep a list U of unique objects, along with a counter of how many times that objects appears. This list U starts as the empty list.
For each of our n objects, check it against every object in U . If it is equivalent to an object, increment that counter, and stop. Otherwise, if it is not equivalent to anything in U , add the object to U and set that counter equal to 1.
At the end of the algorithm, we simply scan U for the largest counter value, and it is larger than $n/2$, we have a majority element.

(b) Incorrect $n \log n$ algorithm.

- This will only work if every object is equivalent. At the bottom level with 2 objects, they have to both be the same, but every time we compare the majority elements of a sublist, they have to be equal to move on, so everything has to be the same.

(c) How can we use divide-and-conquer to develop an $O(n \log n)$ solution?

- What we need is to keep track of the number of times the majority element within a sublist was seen, which requires more careful merging.
We will still use divide and conquer, and split the list L in half each time. Notice that the majority element $M(L)$ of a list must be the majority element of at least one of the two half-lists, L_{left} and L_{right} . When merging, instead of just checking that the two majority elements are the same, we will instead compare the majority element $M(L_{left})$ against all items in L , and then compare $M(L_{right})$ against all items in L , keeping track of how many times each appear. At most one of the two items can be the majority element of L , so that's the item passed up to the next level.
Note that sometimes a sublist will have no majority element, but that's fine, and we can just emit no majority element for that sublist. At the next level up, when merging, all we have to do is check whether the majority element of the other sublist is still a majority element of the full list. The merging operation will take $2n$ equivalence checks at each level. Furthermore, we are dividing up the problem into two problems half the size. So the recurrence is

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

which leads to an $O(n \log n)$ algorithm.