1 Search in $O(\log n)$ time

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 an algorithm that runs in time $O(\log n)$. Note: you should explain why the algorithm is correct and then analyze the running time.

2 Merge sort

Suppose we modify the merge sort algorithm as follows. Instead of dividing the list into two non-overlapping sub-lists of roughly equal size, we divide the list into three non-overlapping sub-lists of roughly equal size. Then we recursively sort the three sub-lists and then merge the three sub-lists. First, give an $O(n)$ time algorithm for merging three sorted lists into one sorted list. Then, analyze the running time of the modified algorithm: how does it compare with the original algorithm?

3 Recurrence

Solve the following recurrence. Show the steps.

1. $T(n) = \begin{cases} 12T(n/4) + n^{1.5}, & \text{if } x \geq 4 \\ 1, & \text{otherwise} \end{cases}$

2. $T(n) = \begin{cases} T(\sqrt{n}) + \log n, & \text{if } x \geq 4 \\ 1, & \text{otherwise} \end{cases}$

3. $T(n) = T(n/2) + \Theta(1)$, which is the running time of the binary search.

4 Integer multiplication

The divide and conquer approach for multiplying two integers $x$ and $y$ discussed in class does not seem to work too well. Recall that we treat $x$ and $y$ as binary encoded. We divide $x$ (and $y$) into half: the high order $n/2$ bits and the low order $n/2$ bits. That is, we write $x = x_1 \cdot 2^{n/2} + x_0$ and $y = y_1 \cdot 2^{n/2} + y_0$. Then we recursively solve four sub-problems and then combine into the product of $x$ and $y$.

Now we want to improve the algorithm to make it run faster. The idea is similar to the matrix multiplication: we want to reduce the number of sub-problems to call recursively. Give an improved divide and conquer algorithm by improving upon the previous algorithm. Analyze the running time of your algorithm. Hint: this observation should help: $(x_1 + x_0) \cdot (y_1 + y_0) = x_1 y_1 + x_1 y_0 + x_0 y_1 + x_0 y_0$.

5 Divide and conquer

An array $A[1,n]$ is said to have a majority element if more than half of its entries are the same. Given an array, the task is to design an algorithm to tell whether the array has a majority element, and, if so, to find that element. The elements of the array are not necessarily from some ordered domain like the integers, and so there can be no comparisons of the form like: is $A[i] > A[j]$? This occurs often: for example, there is no natural way of saying which of two images are larger, but we can tell whether they are the same bitmap image. On the positive side, you can get answer for questions of the form: is $A[i] = A[j]$? in constant time. Let me repeat: you can only examine whether two array elements are the same or not, but you will not be able to know which one is smaller and which one is larger.

Now, show how to solve this problem in $O(n \log n)$ time using a divide and conquer approach. You may use a natural dividing way: divide $A$ into two arrays of half size. Argue why your algorithm is correct, and analyze its running time. Again, you can only rely on queries like $A[i] = A[j]$? but not $A[i] \leq A[j]$? (which means you can not sort the list by say merge sort).