1 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.

2 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: \((x_1 + x_0) \cdot (y_1 + y_0) = x_1y_1 + x_1y_0 + x_0y_1 + x_0y_0\).

3 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: \( A[i] > A[j] \)? 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: \( A[i] = A[j] \)? 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).

4 Matrix multiplication: a special case

This problem is about matrix operation. If you are unfamiliar with matrices, review Section D.1 (p. 1217) of the textbook. In class, I presented the Strassen’s algorithm for multiplying two square matrices. Now, we consider a special case of multiplying two square matrices: given an \( n \times n \)
matrix $A$, compute the product $A \times A$. That is, we want to compute the product of $A$ and $A$ itself. Note that $A \times B$ is often written as $AB$.

a First show that you only need five multiplications for computing $A \times A$ when $n = 2$ (so let $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$).

b After knowing how to do the previous part, Tom thought he can use the following approach to obtain a faster algorithm for computing $A \times A$: “Just like the Strassen’s algorithm, except that using 7 subproblems of size $n/2$, I can now only use 5 subproblems of size $n/2$ based on my observation in step (a). Then I get an algorithm runs in time $O(n^{\log_2 5})$.” Now, tell me what is wrong with Tom’s approach.

c Now let me convince you that computing $A \times A$ is no easier than than the general square matrix multiplication in terms of algorithm efficiency. For this purpose, let us suppose that you can compute $A \times A$ for a square matrix $A$ with $S(n) = O(n^c)$ time (for some constant $c \geq 2$). Then I claim that any two $n$ by $n$ matrices can be multiplied in time $O(n^c)$. Your task is to fill in the missing parts of the following argument.

(i) Given two $n$ by $n$ matrices $A$ and $B$, show me that you can compute the matrix $AB + BA$ in time $3S(n) + O(n^2)$.

(ii) Given two $n$ by $n$ matrices $X$ and $Y$, we consider the $2n$ by $2n$ matrices $A$ and $B$, where $A = \begin{bmatrix} X & 0 \\ 0 & 0 \end{bmatrix}$, and $B = \begin{bmatrix} 0 & Y \\ 0 & 0 \end{bmatrix}$. Tell me what is $AB + BA$ in terms of $X$ and $Y$.

(iii) Using (i) and (ii), argue that $XY$ can be computed in $3S(2n) + O(n^2)$ time. Then conclude that matrix multiplication takes time $O(n^c)$.

5 Balls and Bins

Suppose we throw $n$ balls (note we will throw exactly $n$ balls, and each throwing is independent of other throwing) into $n$ bins with the probability of a ball landing in each of the $n$ bins being equal. We assume each throwing is independent of other throwing. You can assume $n$ is large. You may need the following mathematical fact: when $n \to \infty$, $(1 - \frac{1}{n})^n \to e^{-1}$, where $e$ is the well-known mathematical constant.

1. What is the probability of a particular box (say the first box) end up being empty after the $n$ throwing?

2. What is the expected number of empty bins?