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 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).

3 Matrix multiplication: a special case

This problem is about matrix operation. 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 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 $O(n^c)$.

4 Subset of lists

Given an unordered list $L$ of $n$ numbers and a target value $C$, we want to find the smallest set (i.e. the set with the smallest number of values in the list) from the list whose sum is at least $C$. For example, suppose we have the following numbers: $L = \{1, 9, 8, 4, 6, 5\}$ and $C = 20$, then we need to select at least three numbers (say 4, 8, 9) whose sum is 21 which is over 20, and thus is a good choice for $C = 20$.

Now design an algorithm to find the smallest subset for the given list of numbers $L$ and $C$. For simplicity, we may assume there is no duplicate numbers in $L$ and all numbers in $L$ are positive, and also $C > 0$. Your algorithm should run in $O(n)$ time. Note: you cannot use hashing: hashing often works great in practice but they don’t have worst case run time as desired here. Also you cannot use linear time sorting algorithms such as counting sort: values of these numbers and $C$ can be large. Hint: use the linear-time selection algorithm; and apply divide and conquer.