Warm-up

Problem 1. The product of two $n \times n$ matrices X and Y is a third $n \times n$ matrix $Z = XY$, where the (i, j) entry of Z is $Z_{ij} = \sum_{k=1}^{n} X_{ik} Y_{kj}$. Suppose that X and Y are divided into four $n/2 \times n/2$ blocks each:

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \text{ and } Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix}.$$

Using this block notation we can express the product of X and Y as follows

$$XY = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}.$$

In this way, one multiplication of $n \times n$ matrices can be expressed in terms of 8 multiplications and 4 additions that involve $n/2 \times n/2$ matrices. Let $T(n)$ be the time complexity of multiplying two $n \times n$ matrices using this recursive algorithm.

a) Derive the recurrence for $T(n)$. (Assume adding two $k \times k$ matrices takes $O(k^2)$ time.)

b) Solve the recurrence by unrolling it.

Problem 2. Similar to the integer multiplication algorithm, there are algebraic identities that allow us to express the product of two $n \times n$ matrices in terms of 7 multiplications and $O(1)$ additions involving $n/2 \times n/2$ matrices. Let $T(n)$ be the time complexity of multiplying two $n \times n$ matrices using this recursive algorithm.

a) Derive the recurrence for $T(n)$. (Assume adding two $k \times k$ matrices takes $O(k^2)$ time.)

b) Solve the recurrence by unrolling it.

Problem solving

Problem 3. Your friend Alex is very excited because they have discovered a novel algorithm for sorting an array of n numbers. The algorithm makes three recursive calls on arrays of size $\frac{2n}{3}$ and spends only $O(1)$ time per call.

```
function NEW-SORT(A)
    if |A| < 3 then
        Sort A directly
    else
        NEW-SORT(A[0 : 2n/3])
        NEW-SORT(A[n/3 : n])
        NEW-SORT(A[0 : 2n/3])
```

Alex thinks their breakthrough sorting algorithm is very fast but has no idea how to analyze its complexity or prove its correctness. Your task is to help Alex:

a) Find the time complexity of NEW-SORT.

b) Prove that the algorithm actually sorts the input array.

Problem 4. Suppose we are given an array A with n distinct numbers. We say an index i is locally optimal if $A[i] < A[i-1]$ and $A[i] < A[i+1]$ for $0 < i < n-1$, or $A[i] < A[i+1]$ for if $i = 0$, or $A[i] < A[i-1]$ for $i = n-1$.

Design an algorithm for finding a locally optimal index using divide and conquer. Your algorithm should run in $O(\log n)$ time.

Problem 5. Given two sorted lists of size m and n. Give an $O(\log(m+n))$ time algorithm for finding the kth smallest element in the union of the two lists.

Problem 6. Solve the following recurrences using the Master Theorem or unrolling (the solutions will use the Master Theorem to allow you to practice that). All are $O(1)$ for $n = 1$.

a) $T(n) = 4T(n/2) + O(n^2)$
b) $T(n) = T(n/2) + O(2^n)$
c) $T(n) = 16T(n/4) + O(n)$
d) $T(n) = 2T(n/2) + O(n \log n)$
e) $T(n) = \sqrt{2}T(n/2) + O(\log n)$
f) $T(n) = 3T(n/2) + O(n)$
g) $T(n) = 3T(n/3) + O(\sqrt{n})$

Problem 1. The product of two $n \times n$ matrices X and Y is a third $n \times n$ matrix $Z = XY$, where the (i,j) entry of Z is $Z_{ij} = \sum_{k=1}^{n} X_{ik} Y_{kj}$. Suppose that X and Y are divided into four $n/2 \times n/2$ blocks each:

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \text{ and } Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix}.$$

Using this block notation we can express the product of X and Y as follows

$$XY = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}.$$

In this way, one multiplication of $n \times n$ matrices can be expressed in terms of 8 multiplications and 4 additions that involve $n/2 \times n/2$ matrices. Let $T(n)$ be the time complexity of multiplying two $n \times n$ matrices using this recursive algorithm.

a) Derive the recurrence for $T(n)$. (Assume adding two $k \times k$ matrices takes $O(k^2)$ time.)

b) Solve the recurrence by unrolling it.

(a) $T(n) = 8T\left(\frac{n}{2}\right) + 4O(k^2)$ i.e. $k = n$

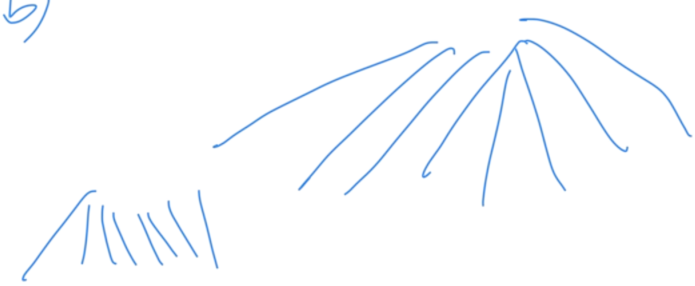
$a = 8$ $b = 2$ $\log_b a = 3$

$f(n) = 4O(k^2) = n^2$

So Case 1

We have $O(n^3) = T(n)$

(b)



cn^2

$2cn^2$

$4cn^2$

Problem 2. Similar to the integer multiplication algorithm, there are algebraic identities that allow us to express the product of two $n \times n$ matrices in terms of 7 multiplications and $O(1)$ additions involving $n/2 \times n/2$ matrices. Let $T(n)$ be the time complexity of multiplying two $n \times n$ matrices using this recursive algorithm.

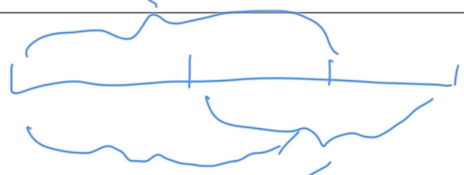
a) Derive the recurrence for $T(n)$. (Assume adding two $k \times k$ matrices takes $O(k^2)$ time.)

b) Solve the recurrence by unrolling it.

(a) $T(n) = \begin{cases} 7T\left(\frac{n}{2}\right) + O(n^2) \\ O(1) \end{cases}$

$a = 7 \quad b = 2$

Problem 3. Your friend Alex is very excited because they have discovered a novel algorithm for sorting an array of n numbers. The algorithm makes three recursive calls on arrays of size $\frac{2n}{3}$ and spends only $O(1)$ time per call.



```
function NEW-SORT(A)
    if |A| < 3 then
        Sort A directly
    else
        NEW-SORT(A[0 : 2n/3])
        NEW-SORT(A[n/3 : n])
        NEW-SORT(A[0 : 2n/3])
```

Alex thinks their breakthrough sorting algorithm is very fast but has no idea how to analyze its complexity or prove its correctness. Your task is to help Alex:

a) Find the time complexity of NEW-SORT.

b) Prove that the algorithm actually sorts the input array.

(a) $T(n) = \begin{cases} 3\,T(\frac{2}{3}n) + \underbrace{O(1)} \\ O(1) \end{cases}$

why?

Because we just manipulate original array

Problem 4. Suppose we are given an array A with n distinct numbers. We say an index i is locally optimal if $A[i] < A[i-1]$ and $A[i] < A[i+1]$ for $0 < i < n-1$, or $A[i] < A[i+1]$ for if $i = 0$, or $A[i] < A[i-1]$ for $i = n-1$.

Design an algorithm for finding a locally optimal index using divide and conquer. Your algorithm should run in $O(\log n)$ time.

找到一个就行, unsorted array

Base Case: Array with 3 elements

1

3

2

Problem 6. Solve the following recurrences using the Master Theorem or unrolling (the solutions will use the Master Theorem to allow you to practice that). All are $O(1)$ for $n = 1$.

$f(n) < n^{\log_b a - \varepsilon}$

a) $T(n) = 4T(n/2) + O(n^2)$
b) $T(n) = T(n/2) + O(2^n)$
c) $T(n) = 16T(n/4) + O(n)$
d) $T(n) = 2T(n/2) + O(n \log n)$
e) $T(n) = \sqrt{2}T(n/2) + O(\log n)$
f) $T(n) = 3T(n/2) + O(n)$
g) $T(n) = 3T(n/3) + O(\sqrt{n})$

Depending on a, b and f(n) the recurrence solves to:

1. if $f(n) = O(n^{\log_b a - \varepsilon})$ for $\varepsilon > 0$ then $T(n) = \Theta(n^{\log_b a})$, Upper
2. if $f(n) = \Theta(n^{\log_b a} \log^k n)$ for $k \geq 0$ then $T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$,
3. if $f(n) = \Omega(n^{\log_b a + \varepsilon})$ and $a\, f(n/b) \leq \delta\, f(n)$ for $\varepsilon > 0$ and $\delta < 1$ then $T(n) = \Theta(f(n))$, Lower

$(\log n)^k$?

(a) $a = 4 \quad b = 2 \qquad \log_b a = \boxed{2}$

$f(n) = n^2 \quad \Rightarrow \quad n^2 = n^{\boxed{2}} (\log k)^0$ i.e. $k = 0$

so case 2

$O(n^2 \log n)$

(b) $a = 1 \quad b = 2 \qquad \log_2 1 = 0 \quad \Rightarrow \quad 2^n$

$f(n) = 2^n$

$T(n) = O(2^n) \qquad$ Case 3

(c) $a = 16 \quad b = 4 \qquad \log_4 16 = 2 > 1 \qquad$ Case 1

$f(n) = n \qquad T_n = O(n^2)$

(d) $a=2$ $b=2$ $\log_2 2=1$

$f(n)=n\log n$

Case 2

$T(n)=O(n\log^2 n)$

(e) $a=\sqrt{2}$ $b=2$ $\log_2 \sqrt{2}=\frac{1}{2}$

$f(n)=\log n$