WEEK 2, LECTURE 4 ON 28 AUGUST 2021 CS1.301.M21 ALGORITHM ANALYSIS AND DESIGN

MASTER THEOREM

For a recurrence relation $T(n) = aT(\lceil \frac{n}{b} \rceil) + O(n^d)$, then:

> $O(n^d)$ being the complexity of other functions that join the subparts into a final part.

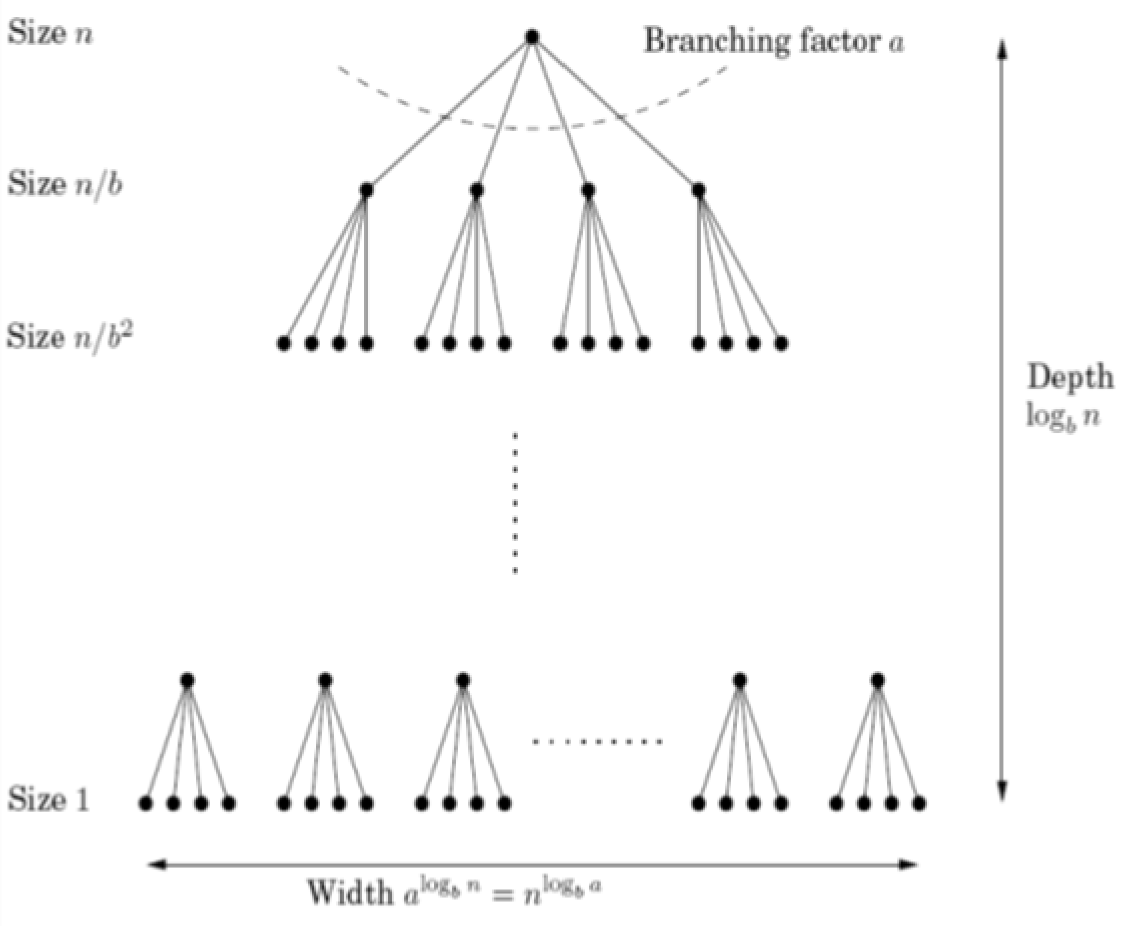
$$T(n) = \begin{cases} O(n^d) & d > \log_b a \\ O(n^d \log_b n) & d = \log_b a \\ O(n^{\log_b a}) & d < \log_b a \end{cases}$$

Proof

In a recurrence relation, the work done per depth is:

$$a^k \times O((\frac{n}{b^k})^d) = O(n^d) \times (\frac{a}{b^d})^k$$

And the summation of the geometric series at all these k depths gives the total work.



The common ratio for the series is $\frac{a}{b^d}$ and the first term is n^d.

For the **first case**, the series is always decreasing and is dominated by $O(n^d)$ which will become the complexity of the total work as well

For the **second case**, the ratio is exactly 1 and this means that all the $O(\log n)$ terms all have work equal to $O(n^d)$ and hence totally the work will be $O(n^d \log n)$

For the **third case**, The ratio is greater than 1 and the total will be dominated by the second term:

$$n^d \left(\frac{a}{b^d}\right)^{\log_b n} = n^d \left(\frac{a^{\log_b n}}{(b^{\log_b n})^d}\right) = a^{\log_b n} = a^{(\log_a n)(\log_b a)} = n^{\log_b a}$$

MERGE SORT

An algorithm to sort a list of n numbers by

- Splitting the list into two equal halves
- Recursively sorting each half
- Merging the two sub lists into a sorted list

Merging step takes a linear amount of time ($O(n)$) so the recurrence relation can be written as:

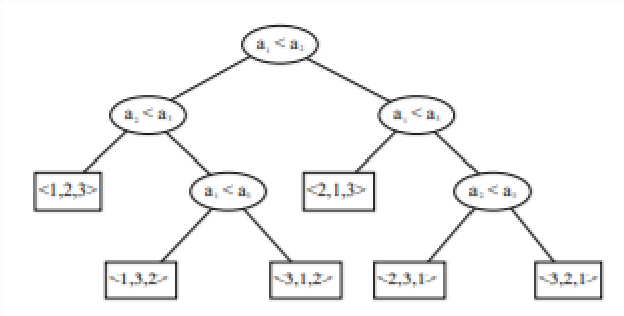
$$T(n) = 2T(n/2) + O(n)$$

> T(n/2) for each sub list and linear time for the merge

$\Omega(n \log n)$ is the most optimal complexity for sorting. Why?

For any n length array, there are $n!$ different permutations

When we make a comparison, we are deciding between two different permutations of the array. Hence a decision tree might be formed like so:



To decide on an answer, i.e. to get to a leaf in the decision tree we must go through at least $\log n!$ comparisons. which is $\Omega(n \log n)$ (shown here).

Hence we say that comparison based sorting is optimal when it is $O(n \log n)$

MATRIX MULTIPLICATION

The naive method

Doing calculations according to the definition of matrix multiplication as shown:

$$XY = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE+BG & AF+BH \\ CE+DG & CF+DH \end{bmatrix}$$

This will yield a complexity of $O(n^3)$

Strassen's Algorithm

This algorithm obtains a correct result with seven unit multiplications:

$$XY = \begin{bmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{bmatrix}$$

$$\begin{aligned} P_1 &= A(F-H) \\ P_2 &= (A+B)H \\ P_3 &= (C+D)E \\ P_4 &= D(G-E) \end{aligned} \quad \begin{aligned} P_5 &= (A+D)(E+H) \\ P_6 &= (B-D)(G+H) \\ P_7 &= (A-C)(E+F) \end{aligned} \quad \begin{aligned} T(n) &= 7T(n/2) + O(n^2) \\ O(n^{\log_2 7}) &\approx O(n^{2.81}) \end{aligned}$$

MEDIAN COMPUTATION

Problem: Find the median of n numbers

We can merge sort and then output the exact middle element in $O(n \log n)$

Or can even be generalized as: Find the kth smallest element

Can we do better?

Consider an operation:

We create three arrays for an integer k in which the elements are

- Smaller than k (S_L)
- Equal to k (S_V)
- Bigger than k(S_R)

We can then recursively continue on the array as follows:

$$\text{selection}(S,k) = \begin{cases} \text{selection}(S_L,k) & \text{if } k \le |S_L| \\ v & \text{if } |S_L| < k \le |S_L| + |S_v| \\ \text{selection}(S_R, k - |S_L| - |S_v|) & \text{if } k > |S_L| + |S_v|. \end{cases}$$

Since the kth smallest element will be the kth smallest element in S_L as well. And will be the $k - |S_L| - |S_V|$.

But for an even split, it turns out the **best choice is the median itself**.

So how do we choose v?

We could choose randomly (which is postponed to another lecture)

We choose v **deterministically**:

- Divide the elements into groups of 5 each.
- Find the median of each of the n/5 groups by sorting a length 5 array and picking the third element (which is a linear time operation)
- Then find the median of these n/5 medians.

After this we can conduct the above shown recursion with the median of the n/5 medians to solve the problem.

But why are we doing all this? Why is it better than any arbitrary v?

Consider the median of medians to be x. This value is greater than at least half of the considered $n/5$ medians (i.e. $n/10$ medians). And these themselves are medians of the groups of 5 and are greater than at least half of those groups of 5 (i.e. 3)

Therefore x is greater than at least $\frac{3n}{10}$ elements are greater (and in fact lesser) than x.

The following recurrence relation is obtained for the entire operation.

$$\boxed{T(n) \le \begin{cases} O(1) & \text{if } n < 140 \\ T(\lceil n/5 \rceil) + T(7n/10 + 6) + O(n) & \text{if } n \ge 140 \end{cases}}$$

$T(\lceil n/5 \rceil)$ is the work done to find the median of the medians of the $n/5$ groups/

$T(7n/10 + 6)$ is coming from the recursion relation. Since there are at least $\frac{3n}{10}$ elements that are greater than or less than x then at most $\frac{7n}{10}$ will be the maximum size of S_L or S_R and

$O(n)$ to create S_L ,S_R and S_V

The recurrence relation can be solved by substitution (and not Master Theorem since it is not of that form) to give a total time complexity of $O(n)$.