Multi-Merge

Jonathan Wenger

Iterative Algorithm

The recurrence relation for the iterative algorithm is:

T_z={0 when z=1
{T_{z-1}+nz when z>1

- **The Base Case:** You only need to merge one of the subarrays into nothing else. Because this subarray is an already sorted array, that takes zero time.
- **The Recursive Work** :T_{z-1}, as it is the amount of work it takes to get the current final sorted array.
- **The Non-Recursive Work:** Every time you merge another subarray into the sorted final array, the worst case scenario is iterating through the entire final array (which at that point is of size n(z-1) elements, as it's the size of all of the subarrays already went through), and iterating through the entire current subarray (which is of size n). So that would be iterating through n(z-1)+n=nz elements.

I got the closed form of **T_z**$=n(\frac{1}{2}z^2+\frac{1}{2}z-1)$ (using the **guess and check** method):

- I started by using the recurrence relation to find T_1, T_2, T_3, and T_4, and started to notice a pattern:
    - T_1=0. T_2=0+2n, T_3=0+2n+3n, T_4=0+2n+3n+4n
- I then assumed the pattern was T_z=0+2n+3n+4n+5n+6n…+zn=n(2+3+4+5+6…+z)
- The sum of consecutive integers from 2 to z = $\frac{z-1}{2}(z+2)$ (using the sum of consecutive numbers formula[1])
- Therefore, T_z would = n($\frac{z-1}{2}(z+2)$)$=n(\frac{1}{2}z^2+\frac{1}{2}z-1)$ (which is what I got above).
- Because I used guess and check, I have to prove by induction this closed formula works.
    - **Base Case (z=1):** $T_0=n(\frac{1}{2}(1)^2+\frac{1}{2}(1)-1)$=0, so the base case is correct.
    - **Inductive Step:** I'm going to check my closed formula. If my closed formula is correct, the closed formula for T_{z+1} should equal the closed formula for T_z+the non-recursive work (which is n(z+1)).
        - The closed formula for T_z+the non-recursive work: $=n(\frac{1}{2}z^2+\frac{1}{2}z-1)$+n(z+1) = $n(\frac{1}{2}z^2+\frac{3}{2}z)$.
        - The closed formula for T_{z+1}: $T_{z+1}=n(\frac{1}{2}(z+1)^2+\frac{1}{2}(z+1)-1)$, which after simplifying = $n(\frac{1}{2}z^2+\frac{3}{2}z)$.
    - Because the closed formula for T_z+the non-recursive work=the closed formula for T_{z+1} this closed form of T_z is correct. End of proof.

[1] This formula says, if I am trying to add n consecutive integers from the lowest integer a to the highest integer b, the sum = (n / 2)(a + b).

Divide-And-Conquer Algorithm

The way I created this algorithm is that I viewed this problem as a basic integer MergeSort problem, but instead of each item in the array being an integer, each item in the array was an array of sorted integers. I first broke up the 2D array (by recursively halving the 2D array) into z 1D sorted arrays (each of length n), and then built it back up by merging two 1D sorted arrays into a larger 1D sorted array using the merge method that is found in MergeSort. Because this algorithm divided the arrays recursively, it also built the arrays back up recursively, which works in faster time.

The recurrence relation for this algorithm is:

T_z={0 when z=1
{$2(T_{z/2})$+zn when z>1

- **The Base Case:** At this point in time, there is only one subarray (of length n). Because this is a sorted array, and you're not merging this array into anything else, this takes zero time.
- **The Recursive Work** : $2(T_{z/2})$, as every time you're splitting up the 2D array into two 2D arrays, and that is the amount of work it takes to get the final sorted array.
- **The Non-Recursive Work:** At every "level", you are merging two sorted subarrays into a larger array. That means, in the worst case scenario, at a given level you are merging each subarray with another subarray. Because there are z arrays of length n, there are a total of zn elements in the 2D array, so at every level you (worst-case) are iterating through zn elements to merge.

I got the closed form of **T_z**$=zn(log(z))$ (using the **Master Theorem**): This proof will be assuming z is a power of b, but works even if z is not a power of b. Additionally, replace n with z in the diagram ↠.

Suppose $T(n) = a \cdot T\left(\frac{n}{b}\right) + O(n^d)$. Then

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

  - In this case, **a**=**2**, as that is the constant next to T.
  - **b**=**2**, as that is what z is being divided by every time.
  - **d**=**1**, as we are going through zn elements.
    - **DISCLAIMER**: Although we are going through zn elements, and not z elements, one can claim that the Master Theorem cannot apply in this case. However, I would like to argue that it can. Although n and z are constantly changing, **d will always equal 1**. This is because n and z are directly proportional to each other. By any scale n decreases, z increases, and vice versa. That means regardless of what values n and z are, the value of n multiplied by z will never change. Because of this, **d will always equal 1 and does not change** regardless of what level of the recurrence. **The only difference is in the solved closed formula, by the n^d, it just replaces to $(zn)^d$.**
- Because $a=b^d$ ($2=2^1$), the first case of the Master Theorem applies. So the closed formula of this recurrence is **T_z**$=zn(log(z))$**.**

Because the Divide-And-Conquer algorithm runs in O(zn(log(z)) time, and the Iterative algorithm runs is O(nz^2) time, the Divide-And-Conquer algorithm is better than the Iterative algorithm.