MultiMerge

Benjamin Katz

March 2022

1 Iterative Algorithm

The Iterative Algorithm is an algorithm that takes an array of sorted array and iterates through the array of arrays and merges the next sub array with the large array that all previous arrays have already merged into. At every iteration n(the size of the sub array) times i(the iteration that we are up to) amount of work has to be done in order to march through both the large array(of size $n*(i-1)$) and the array we are up to (of size n) one time in order to merge them into one larger sorted array. The total amount of work done is given as the work done at the last iteration plus the sum of all work done in the previous iterations. We can model this as a recurrence relationship.

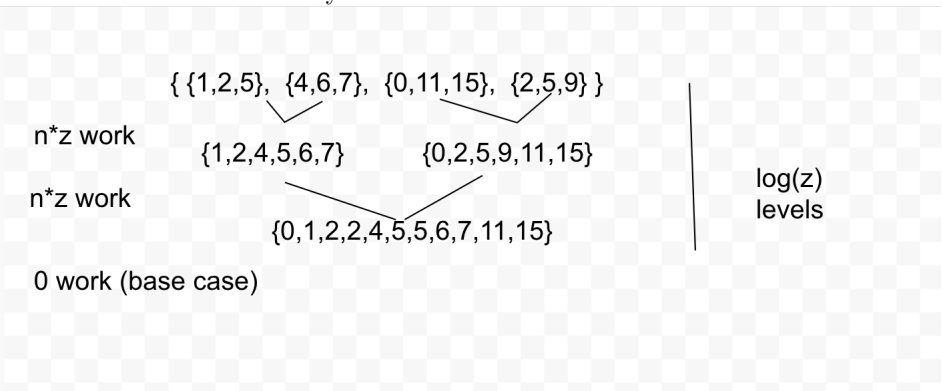
- $T_1 = 0$ (Base case does no work as it just returns the single sorted array)
- $T_i = T_{i-1} + (i*n)$(Recursive case of all previous work plus the cost of this merge)

To solve this recurrence into closed form and solve for the total work done given an array size of z arrays each of size n we use the plug and chug method.

- $T_i = T_{i-1} + (i*n)$
- $T_{i-1} = T_{i-2} + ((i-1)*n)$
- $T_{i-2} = T_{i-3} + ((i-2)*n)$
- ...
- $T_i = T_{i-3} + n*(i-2) + n*(i-1) + n*i$
- Through inspection we can see that this recurrence can be stated in closed form
- $T_i = (i-1)*(i*n) - n*\sum_{k=1}^{i-2} i$
- Simplifying further and subbing in z for the last iteration to find the total work of the problem:
- $T_z = (z-1)*(z*n) - \frac{n*(z-2)*(z-1)}{2} = O(n*z^2)$

2 Divide-and-Conquer Algorithm

In my code I took the large 2d array of length z with sub arrays size n and recursively combined it into a 2d array of size $z/2$ with sub arrays of size $2 * n$ by combining every two arrays into one one larger sorted array until reaching the base case where there is just one single sorted array containing all the elements. When combining two sub arrays $2 * n$ work is done. To combine all pairs of arrays on a given level, $z * n$ amount of work must be done since every element must be looked at once and only once.



Because we are dividing the problem in half every time and doing $n * z$ work at every level, we can write a recurrence relationship that looks like this:

- $T_1 = 0$ (Base case does no work)
- $T_i = T_{(i/2)} + (n * z)$(Every level is split in half but still does the same amount of work)
- Plug and chug to get closed form
- $T_i = T_{(i/4)} + (n * z) + (n * z)$
- $T_i = T_{(i/8)} + (n * z) + (n * z) + (n * z)$
- $T_i = T_{(i/16)} + (n * z) + (n * z) + (n * z) + (n * z)$
- $T_i = T(i/2^k) + k * (n * z)$
- let $k = \log_2 i$
- Find the work of the entire problem let $i = z$
- $T_z = T_{\frac{z}{2^{log_2 z}}} + log_2 z * (n * z)$
- $T_z = 0 + log_2 z * (n * z)$

The total work of this algorithm is $O(nz) \log z$