MultiMerge

Asher Kirshtein

February, 27 2022

1 Iterative Recurrence

1.1 Function

- Equation: $T_z = (z-1)n + (z-2)n + ... + 2n + n$
- Base Case: $T(1) = 0$

  The base case T(1) is 0 since when you have only one array you do 0 merges and simply just return

1.2 Justification

The equation has z arrays and each array contains n integers. For each array we need to do $z-1$ merges while our array grows each time. The last merge gives us an array of size $z*n$ before that merge we have 1 array with size n to merge with an array that is size $z*n-n$ and it progressively lower and lower the closer we get to the start of the merging until we have z arrays each of size n.

1.3 Solving the Recurrence

Solving the Recurrence: $T_z = (z-1)n + (z-2)n + ... + 2n + n$

$T_z = n((z-1) + (z-2) + ... + 2 + 1)$
$T_z = n(z(z-1)/2)$
$T_z = n(z^2 - z)/2$
$T_z = nz^2$

$$\mathrm{T}_z = \begin{cases} 0 & z = 1 \\ n * z^2 & z > 1 \end{cases}$$

$O(n) = nz^2$

2 Divide and Conquer Recurrence

2.1 Code Explanation

In my divide and conquer algorithm I take our 2 dimensional array I make a new array with half of the arrays but each array is double the length. I then merge the first 2 arrays from our original into our new array and then the next 2 into the next until all of the arrays have been merged into our new array. I recursively repeat this process until I have only one array in our 2D array and then return that as our final product.

It is better than the iteration method because when we merge we only merge values that have already been merged every time we go up a level in the tree as opposed to every single merge. Therefore since we are traversing a tree we will have linearithmic time as opposed to quadratic time

2.2 Recurrence

Equation: $T_z = 2n * z/2$

2.3 Solving the Recurrence

$T(2) = 4z/2 + 2n * z/2$
$T(4) = 8z/2 + 4z/2 + 2n * z/2$
$T(8) = 16z/2 + 8z/2 + 4z/2 + 2n * z/2$

$T(2) = T_1 + 2z$
$T(4) = T_2 + 4z$
$T(8) = T_4 + 8z$

Pattern we see that our equation is increasing by a power of 2 every time

We can do $T_z = 2^z * T_{n/z^k} + (n - 2^z - 1) + (n - 2^z - 1) + ... + (n - 2^0)$
$T_z = 2^z * T_{n/z^z} + zn - 2^z - 1 - ... - 2^0$
$T_z = 2^z * T_{n/z^z} + zn - 2^z + 1$

Let $z = log(n)$

$T_z = 2^l og(n) * T_{n/(log(n))^k} + zn - 2^z + 1$
$n * T_1 + nlog(n) - n + 1$
$nlog(n) - n + 1$

We can see from the fact that when we used our plug and chug method our values doubled every time we are dealing with a logarithmic increase instead of a quadratic which shows that it is faster than the iterative version