Samuel Sicklick

Iterative Algorithm

1. $T_{(z)} = \left\{ \begin{array}{ll} 0 & z = 1 \\ T_{(z-1)} + z\,n & Otherwise \end{array} \right\}$
2. The base case of $T_{(1)}$ is 0 since the algorithm there does no work, as the single inputted array is already sorted. For all further cases the algorithm must merge the first and second arrays at a time of $2n$ (due to its comparing of n values in each of those two arrays) and then subsequently merge the next array of size n with that previously merged array. Thus, every additional z array will take as much time as it took to get the merged array of all those which came before z, $T_{(z-1)}$, plus the sum of that array and the last z array, $(z - 1 + n)n$. This can be simplified to $T_{(z-1)} + zn$.
3. Closed Form Solution:
   a. $T_{(z)} = T_{(z-1)} + zn$
   b. $T_{(z-1)} = T_{(z-2)} + (z - 1)n + zn$
   c. $T_{(z-2)} = T_{(z-3)} + (z - 2)n + (z - 1)n + zn$

   …

   d. $T_{(z-k)} = T_{(z-(k+1))} + \sum_{i=2}^{z-k} ni = 0 + \left(\sum_{i=0}^{z-k} ni - \sum_{i=0}^{1} ni\right) = 0 + \frac{(z-k)(z-k+1)}{2}n - n$
   e. $T_{(2)} = T_{(1)} + \sum_{i=2}^{2} ni = 0 + \left(\sum_{i=0}^{2} ni - \sum_{i=0}^{1} ni\right) = 0 + \frac{(2+1)2}{2}n - n = 2n$
   f. $T_{(z)} = 0 + \frac{(z+1)z}{2}n - n = \frac{(z+1)z}{2}n - n$

Divide - and - Conquer Algorithm

1. Recursively divide the input size, z, of the n pre sorted arrays, until there are only two of such arrays being focused upon by the algorithm. These two "base case" arrays are then merged into a single 1d array according to their natural order. Then, the method returns upward and recursively does the same to another two array section of the original z arrays.[1] The two returned arrays are then merged and returned upward until all of the z arrays have been "multimerged" into a single organized 1d array.
2. $T_{(z)} = \left\{ \begin{array}{ll} 0 & z = 1 \\ 2T_{(z/2)} + z\,n + 3 & Otherwise \end{array} \right\}$
   a. The base case of $T_{(1)}$ is 0 since the algorithm there does no work, as the single inputted array is already sorted. For all other cases, the algorithm divides the input size z into two halves, via 3 arithmetic operations, and recursivley "sorts" them down into the "base case." The two returned 1d arrays from the original z arrays are merged at a time of zn since each of the n values in the two $\frac{z}{2}$ arrays must be compared as they merge upward

[1] In the event that z is an odd number, it is possible at this point for a single remaining array to get merged with the other two arrays from the recursive call instead.

with previously merged arrays from lover levels of the recurssion. Thus, the recursive step is $2T_{(z/2)} + zn + 3$.

3. Closed Form Solution:
   a. $T_{(z)} = 2T_{(z/2)} + zn + 3$
   b. $T_{(z/2)} = 2\left(2T_{(z/4)} + \frac{z}{2}n + 3\right) + zn + 3$
   $$= 4T_{(z/4)} + zn + 3(2) + zn + 3(1) = 4T_{(z/4)} + 2zn + 3\sum_{i=0}^{2} 2^{i}$$
   c. $T_{(z/4)} = 4\left(2T_{(z/8)} + \frac{z}{4}n + 3\right) + zn + 3(2) + zn + 3(1)$
   $$= 8T_{(z/8)} + zn + 3(4) + zn + 3(2) + zn + 3(1) = 8T_{(z/8)} + 3zn + 3\sum_{i=0}^{3} 2^{i}$$
   …
   d. $T_{(z)} = 2^{k}T_{(z/2^{k})} + kzn + 3\sum_{i=0}^{k} 2^{i}$
   $$= 2^{k}T_{(z/2^{k})} + kzn + 3\left(\frac{2^{k}-1}{2-1}\right) = 2^{k}T_{(z/2^{k})} + kzn + 3\left(\frac{2^{k}-1}{1}\right) = 2^{k}T_{(z/2^{k})} + kzn + 3\left(2^{k} - 1\right)$$
   e. $T_{(z)} = 2^{k}T_{(z/2^{k})} + kzn + 3\left(2^{k} - 1\right)$
   $$= 2^{k}\left(2T_{(z/2^{k+1})} + \frac{z}{2^{k}}n + 3\right) + kzn + 3\left(2^{k} - 1\right) = 2^{k+1}T_{(z/2^{k+1})} + (k + 1)zn + 3\left(2^{k+1} - 1\right)$$
   f. Let $k = log_2 z$:
      i. $T_{(z)} = 2^{log_2 z}T_{(z/2^{log_2 z})} + z(log_2 z)n + 3\left(2^{log_2 z} - 1\right) = zT_{(1)} + z(log_2 z)n + 3(z - 1)$
      $$= z(0) + z(log_2 z)n + 3(z - 1) = z(log_2 z)n + 3(z - 1)$$

Conclusion

The closed formula for the iterative approach is $T_{(z)} = \frac{(z+1)z}{2}n - n$, running in $O_{(zn)}$. This takes noticeably longer than the $O_{(z(log_2 z)n)}$ divide and conquer approach, whose closed formula is $T_{(z)} = z(log_2 z)n + 3(z - 1)$. The improvement gets more discernable as the number of arrays, z increases. For example, using the iterative approach it can be determined that $T_{(4)} = 9n$, while the divide and conquer approach results in $T_{(4)} = 8n + 9$. Yet, a more dramatic example is seen from the fact that $T_{(8)} = 35n$ when using the iterative approach, while the application of divide and conquer yields $T_{(8)} = 16n + 27$.