CS 577: Introduction to Algorithms — Homework 1 Solutions

Solutions

1(a) During each pairwise merge, one of the arrays is size n and the other is size $O(2^s n)$. So each pairwise merge takes $O(2^s n)$ comparisons. We complete $O(2^s)$ pairwise merges for a total of $O((2^s)^2 n) = O(4^s n)$ total comparisons.

1(b) We can apply pairwise merging in a tree fashion using the MERGE step from MERGESORT which takes two sorted arrays of size m and n and returns a combined sorted array in time $O(m+n)$. First, we pairwise merge arrays of length n until we have $2^s/2 = 2^{s-1}$ arrays of length $2n$. Then we pairwise merge these arrays until we have 2^{s-2} arrays of length $4n$. We continue this process until we have 1 array of length $2^s n$. We return this array. We can define this recursively as follows:

Algorithm 1: COMBINE($A_1, A_2, \dots A_{2^s}$): returns a single sorted array that combines the 2^s input arrays, each of size n

```
1 if $2^s = 1$ then
2     return $A_1$.
3 else
4     for $i = 1 \dots 2^{s-1}$ do
5         $B_i$ =MERGE($A_i, A_{2^{s-1}+i}$)
6     return Combine($B_1, B_2, \dots B_{2^{s-1}}$)
```

1(c) We will use induction to prove that COMBINE correctly combines 2^s arrays of size n into one sorted array.

Proof. We will induct over s.

- Base Case: $s = 0$

  In this case there is only one array that is already sorted. COMBINE would simply return this sorted array, which is correct.

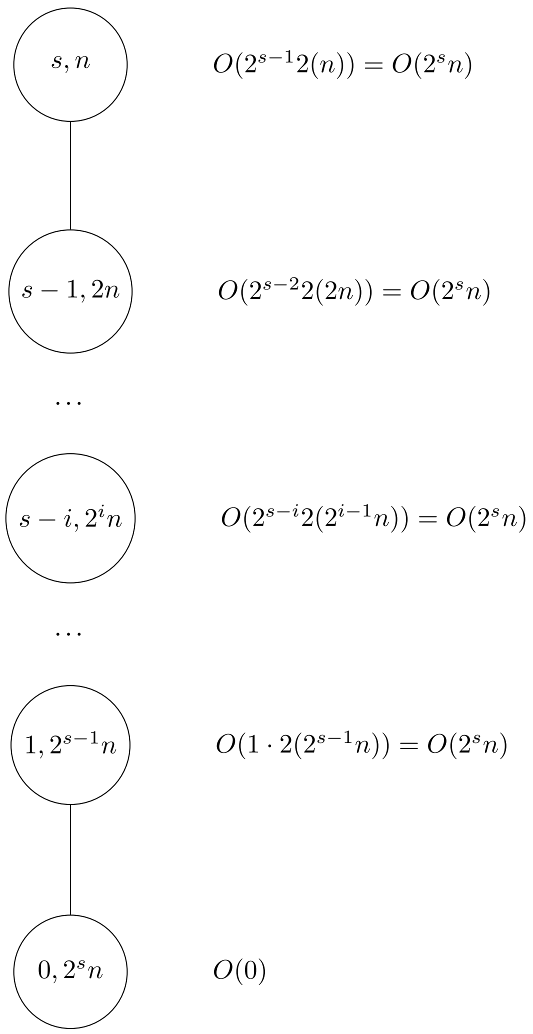
- Inductive Hypothesis: COMBINE correctly combines 2^k sorted arrays into one sorted array.

- Inductive Step: $s = k + 1$

  COMBINE first merges 2^{k+1-1} pairs of arrays of size $n/2$. By the correctness of MERGE, each of these initial merges results in a combined sorted array of size $2n$. At this point we have $2^{k+1-1} = 2^k$ sorted arrays of size $2n$, which we combine using COMBINE. We can apply the inductive hypothesis on this recursive call since the arrays are all of size 2^k. So we know that these 2^k arrays are correctly combined, meaning that our original 2^{k+1} arrays are correctly combined.

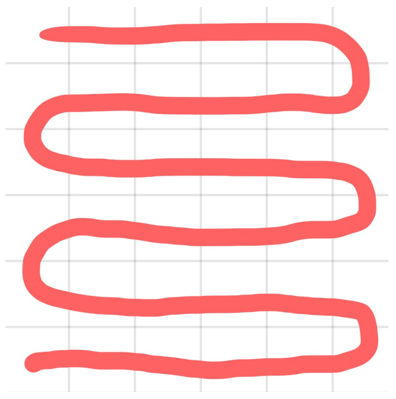
□

1(d) Let $T(s, n)$ denote the number of comparisons made by COMBINE given 2^s arrays of size n. At the first level of recursion, COMBINE makes $O(2n)$ comparisons for 2^{s-1} pairwise merges, resulting in $O(2^{s-1}2n) = O(2^s n)$ local comparisons. We then make a recursive call to COMBINE, inputting 2^{s-1} arrays of size $2n$. This results in the following recurrence: $T(s, n) = T(s-1, 2n) + O(2^s n)$. The recurrence tree is as follows, where the nodes represent recursive calls and the equations to the right of the nodes represent the amount of local work done at that level of recursion. Note that the last level does no comparisons because it is the base case.


s, n
$O(2^{s-1}2(n)) = O(2^s n)$
$s-1, 2n$
$O(2^{s-2}2(2n)) = O(2^s n)$
$\dots$
$s-i, 2^i n$
$O(2^{s-i}2(2^{i-1}n)) = O(2^s n)$
$\dots$
$1, 2^{s-1}n$
$O(1 \cdot 2(2^{s-1}n)) = O(2^s n)$
$0, 2^s n$
$O(0)$

Note that there are $s-1$ levels of the recursion tree excluding the base case. At each level, $O(2^s n)$ local comparisons are completed. This means that in total, COMBINE performs $O(2^s sn)$ comparisons.

2(a) The following example suffices because the rope goes through every single cell. Note that the example can be generalized to a grid of size $n \times n$ for any integer n. So the asymptotic number of cells queried by following the rope around is $\Omega(n^2)$.



2(b) Checking the middle column we can tell directly if the endpoint of the rope is in one of those cells. We can tell if the endpoint is to the right of the middle column or to the left of the column by iterating over the middle column and keeping track of how many times the rope crosses its left boundary. If this number is odd, the end of the rope is on the right side of the boundary otherwise it is on the left.

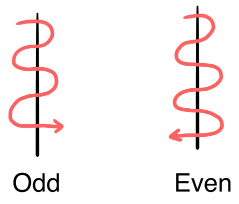
2(c) Call all vertical lines that divide the cells $\ell_1, \cdots, \ell_{n-1}$ (i.e. for every $n \times n$ grid, there are $n-1$ vertical lines that divide the cells). For every ℓ_i, the rope crosses it n_i times. In the given example, the rope crosses the ℓ_1 line 5 times, thus $n_1 = 5$; the rope crosses the ℓ_2 line 5 times, thus $n_2 = 5$, etc. We will now describe our algorithm using the above notation.

Algorithm 2: FindEnd(1,n): find the end of rope from column 1 through n

```
if n= 1 then
    check every cell and return the end.
else
    check every cell in middle column
    if found end then
        return the end
    else
        Compute $n_{n/2}$, the number of times the rope crosses $\ell_{n/2}$
        if $n_{n/2}$ is even then
            recurse on the left side of the graph, FindEnd(1, $n/2$)
        else
            recurse on the right side of the graph, FindEnd($n/2$, n)
```

2(d) **Claim:** If n_i is even, then the end of the rope lies on the left side of ℓ_i; if n_i is odd, then the end of the rope lies on the right side of ℓ_i.

Proof. Since the rope starts at the top left corner, the rope always crosses any ℓ_i from left to right the first time. Consequentially, the second time the rope crosses any ℓ_i, it's from right to left. Following this pattern, we can see that if n_i is even, it means that the last time the rope crosses ℓ_i, it's from right to left. This implies that the end of the rope has to lie on the left side of ℓ_i. Similarly, if n_i is odd, it means that the last time the rope crosses ℓ_i, it's from left to right. This implies that the end of the rope has to lie on the right side of ℓ_i. The following picture describes the situation. □



With this claim, we can now use induction to argue that our divide and conquer algorithm works correctly. We perform induction on the number of columns k remaining.

Base Case ($k = 1$)**:** When the submatrix has a single column, the algorithm correctly looks up the endpoint.

Induction Hypothesis: Assume that we correctly identify the endpoint when there are less than k columns remaining.

Induction Step: Given a matrix that has k columns which is guaranteed to contain the end of the rope, the algorithm checks the middle column and correctly decides whether the endpoint is on the left or the right submatrix according to the above claim. By the induction hypothesis, the algorithm will correctly identify the endpoint of the rope in the submatrix as it has fewer than k columns.

2(d) At most 21 cells.

2(e) Every recursive call reduces the number of columns in the grid by a factor of 2. Let $T(k)$ denote the number of cells queried by the algorithm on a grid of size $n \times k$. Then we have $T(k) = T(\frac{k}{2}) + n$ and $T(1) = n$, which solves to $T(k) = O(n \log k)$. The final number of cells queried is $T(n) = O(n \log n)$.