Burak Yildirim
190104609

Q1) Algorithm is recursive. Base condition occurs if the lenght of the wire is less than or equal to 1, that means no further cutting is required. In the else part, algorithm calls itself with the ceiling of the n/2 because if the lenght is even, ceiling doesn't change the result but if the length is odd, ceiling rolls it up to the next integer so that algorithm can work properly. For example, if n=5 and I take floor instead of ceiling the result would be 2 and the algorithm would stop after 2 recursions. But the answer is not 2, it is 3 so if I take ceiling 5/2 would return 3 and 3/2 would return 2, so the algorithm would stop after 3 recursions instead of 2 which is the wright answer.

Time complexity:

$T(n) = T(n/2) + 1 \quad a = 1 \quad b = 2 \quad f(n) = 1$

$n^{\log_b a} = n^{\log_2 1} = n^0 = 1 \Rightarrow$ case 2 of master theorem

$T(n) \in \Theta(n^{\log_b a} \cdot \log n) \Rightarrow \Theta(1 \cdot \log n) \Rightarrow \Theta(\log n)$ //

Q2) I used merge sort as a template but applied some modifications. First, in the base condition I return the minimum / maximum of the elements at the given low and high positions. Second, in the else part I replaced the merge part with a return statement that returns the minimum / maximum of the results of the left and right sides.

Time complexity:

$T(n) = 2T(n/2) + 1 \quad a = 2 \quad b = 2 \quad f(n) = 1$

$n^{\log_b a} = n^{\log_2 2} = n \Rightarrow$ case 1 of master Theorem

$T(n) \in \Theta(n^{\log_b a}) \Rightarrow \Theta(n^{\log_2 2}) \Rightarrow \Theta(n)$ //

Q3) In this algorithm, we first get the min and max values of the array. Then in the outer loop it calculates the mean of the min and max values and initializes the less and equal with 0. In the inner loop it checks all the array elements. If they are less than mid, algorithm increases the less by 1 and if they are equal to mean value, algorithm increases the equal by 1. After all the elements of the array are checked the inner loop stops. After that algorithm checks if less < k <= less + equal. If this is the case then this means that mean value is our answer. If less >= k, then this means that our answer is smaller than mean value. And if less < k and less + equal < k, then this means our answer is greater than mean value. In smaller than mean situation, algorithm assigns mean - 1 to the max and in greater than mean situation, it assigns mean + 1 to the min.

Time complexity:

$$\sum_{j=0}^{\log k} \sum_{i=0}^{n-1} 1 = \sum_{j=0}^{\log k} n = n \cdot \log k \Rightarrow T(n) \in \Theta(n \cdot \log k)$$

Note: Even if the outer loop's condition is less <= high, it's just a safety net for k values that exceeds n. In normal cases outer loop executes log k times.

Q4) I used merge sort as a template and changed the return part. In return part I return the number of reverse ordered pairs in the subarray by sorting the subarray after checking if i < j and array[i] > array[j] and if it is I increment counter.

Time complexity:

$T(n) = 2\,T(n/2) + n$ → because combine-results has 3 consecutive loops that run n/2 times.

$a = 2 \quad b = 2 \quad f(n) = n$

$n^{\log_b a} = n^{\log_2 2} = n \Rightarrow$ case 2 of master theorem

$T(n) \in \Theta(n \cdot \log n)$

Q5) In brute force part, function has a for loop that iterates n (power) times and multiplies the base with itself.

Time complexity:

$$\sum_{i=0}^{n-1} 1 = n \qquad T(n) \in \Theta(n)$$

In divide and conquer part, function's base case is when power <= 1, it doesn't need to multiply any more, so it returns base number and in the else part I recursively call the function with the same base and with floor and ceil of power/2, and return the multiplication of the results of those recursive calls.

Time complexity:

$T(n) = 2\,T(n/2) + 1 \qquad a = 2 \quad b = 2 \quad f(n) = 1$

$n^{\log_b a} = n^{\log_2 2} = n \Rightarrow$ case 1 of master theorem

$T(n) \in \Theta(n)$