* HW4 Yeşim Yalçın 20010 4004 0 34

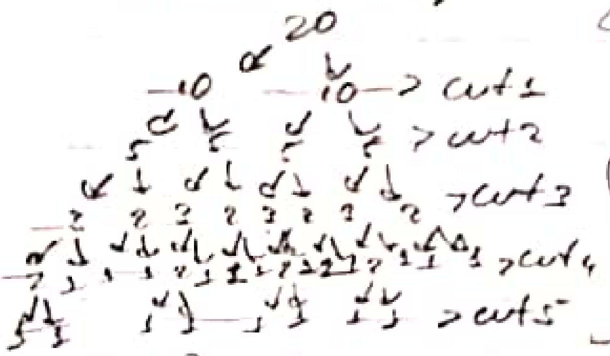
(1) n meter long wire ← Input
1 meter long pieces ← Wanted outcome

* Aproach:

→ We can divide the wire pieces by 2 recursively and get 1 meter long pieces in the end.

* Demonstration:

→ Assume $n = 20$



* 5 cuts are needed for a 20 meter long wire.
* Dividing the problem by two gives us $\Theta(\log n)$ time complexity.
* $\log_2^{20} \approx 5$. It is the same as our answer.

* Pseudocode:

```
cutting (wireSize)
    if (wireSize == 1)
        return 0          → meaning no cuts needed anymore
    else
        return cutting(wireSize / 2) + 1   → meaning a cut is happening
```

* Analization:

* $T(n) = T\left(\frac{n}{2}\right) + 1$

Dividing the problem by 2 each time

There are only comparisons happening in each time. It is $\Theta(1)$

Master Theorem:
$b = 2 \quad a = 1$
$f(n) = 1$
$\log_2^1 = 0$
$f(n) \in \Theta(n^0)$
$\boxed{T(n) \in \Theta(\log n)}$

(2) → n experiments peformed
→ success rates for each experiment is saved (n rate)
→ The worst and the best rates are needed.

Aproach:

* The best method I could think of would be using Quickselect algorithm because it does not require sorting and it guarantees $O(n)$ time complexity most of the time. However, it is not a divide and conquer algorithm. It is decrease and conquer algorithm because some resources accept an algorithm as divide and conquer only if the problem is divided into subproblems. In quickselect, only the size of the same problem is divided. However, some resources accept it as divide an conquer.
To avoid the conflict, I choose to use a mergesort algorithm then return the 0th also n-th index of the sorted array. Mergesort is indeed a divide and conquer algorithm.

Analization:

* Merge sort divides the array into two halves each time then recombines them in a sorted way. This gives the relation:

→ $T(n) = 2T(\frac{n}{2}) + n$

2 subproblems | size halved each time | Combining is done in $O(n)$ time

Master Theorem:

$a = 2$
$b = 2$
$\log_b^a = 1$
$f(n) = n$

$f(n) \in \Theta(n^1)$

$\boxed{T(n) \in \Theta(n \log n)}$

* Combination is done this way:
→ Compare the first element of left array with the first element of right array. Add the smallest one to the result array. If left one was chosen, go to the next element in the left array, if right one is chosen go to the next element in the right array. Continue till all elements are added to the result array.

* During comparison either left group's index or right group's index will be incremented each time and the loop will finish after one of them reaches the end. Meaning there will be max n comparisons.

3 Finding the k^{th} successful element

* Approach:

* This time I can use the Quickselect algorithm to find the kth successful experiment's rate because it is a decrease and conquer algorithm. As I have mentioned it guarantees $O(n)$ time complexity most of the time.
↳ mentioned in question 2.

* Analization:

* In Quickselect algorithm, we partition the array after choosing a pivot so it starts like quicksort. After partitioning, we know where the pivot should be placed. If it is equal to $k-1$, it means it is the searched element. If it is smaller than $k-1$, we can continue with the group that includes large elements else we continue with the group that includes smaller elements.

* Best Case: $\Theta(n)$ if the first selected pivot is the searched element.
* Worst Case: $\Theta(n^2)$ if the partitions are done poorly, in a way that makes one group empty each time.

↳ We can avoid this by having a randomized pivot choosing algoritm.

↳ Which guarantees us $O(n)$ time complexity like the best case for average and the worst cases as well.

→ It means there will be only 1 partitioning done. During partitioning, each element in the array is compared with the pivot therefore $\Theta(n)$. If pivot is at index 0:

$$\hookrightarrow \sum_{i=1}^{i=n-1} 1 \Rightarrow \frac{(n-1)-1+1}{2} = \frac{n-1}{2} \in \Theta(n)$$

4) Find the number of reverse-ordered pairs.

* Aproach:

* Finding the reverse-ordered pairs means finding the number of inversions. To do that we will do;
→ Recursively divide the array into two parts
→ After all divisions, recombine them and sort at the same time.
→ While sorting, compare one element of right array with left array. If the right array element is smaller, increment inversion counter by the number amount that is left in the left array. Insert the smaller element to a third combined array and continue for each element.
→ Do the same things for combined arrays till a sorted array of the initial array is created.
→ It is very similar to mergesort. Only difference is, you need to have additional logic to count the inversions in combination steps which is $O(n)$ time.

* Analization:

* In question 2, I had found running time of mergesort as $O(n \log n)$ this is the same because it has the same recurrence relation.
* The additional inversion counting logic does not change the basic operation $O(n)$ because it's logic is implemented into the comparison part but only incrementing operations are added which does not change anything.

(5) Compute a^n where $a>0$

★ Brute Force Way:

★ We can recursively or iteratively multiply a, n times and get the result. In the end there will be exactly n multiplications meaning $\boxed{T(n) \in \Theta(n)}$

★ Divide and Conquer Way

★ We can divide the problems into two subproblems and recursively find the solution of each subproblem then combine by multiplying them. However, this will require the same amount of multiplications therefore, there will be no difference between the brute force way.

$\hookrightarrow T(n) = 2T(\frac{n}{2}) + 1$

In each step only 1 comparisons happen as basic operation therefore it is $\Theta(1)$

Master's Theorem:
$b=2$
$a=2$
$f(n)=1$
$\log_b^a = 1$

$f(n) \in O(n)$

$\Downarrow$

$\boxed{T(n) \in \Theta(n)}$

$$\sum_{i=0}^{n-1} 1 = \frac{(n-1)-0+1}{2} = \frac{n}{2} \in \Theta(n)$$