Q1:

$T(n) = T(n/2) + 1$

$$T(n) = \begin{cases} 1 & n = 1 \\ T(n/2) + 1 & n > 1 \end{cases}$$

$T(n/2) = T(n/4) + 1 + 1$

$T(n/2^2) = T(n/2^3) + 3$

$\vdots$

$T(n) = T(n/2^k) + k$

$2^k = n \Rightarrow k = \log_2 n$

$T(n) = T(n/2^{\log_2 n}) + \log_2 n \Rightarrow$

$T(n) = T(1) + \log_2 n = 1 + \log_2 n = \underline{O(\log(n))}$

Q2: $T(n) = \begin{cases} 1 & n = 1 \\ 2T(n/2) + n & n > 1 \end{cases}$

$T(n/2) = 2T(n/4) + n/2$

$T(n/2^2) = 2T(n/2^3) + n/4$

$\vdots$

$T(n) = 2(2T(n/2^2) + n/2) + n$

$T(n) = 2(2(T(n/2^3) + n/2^2) + n/4) + n$

$\vdots$

$T(n) = 2^k T(n/2^k) + \underbrace{n + n + \ldots + n}_{k \text{ times}}$

$n/2^k = 1 \Rightarrow 2^k = n \Rightarrow k = \log_2 n$ ($2^k \to n$)

$T(n) = n \cdot \underbrace{T(n/2^{\log_2 n})}_{1} + n \cdot \log_2 n = n + n \cdot \log(n)$

$n \cdot \log n > n \Rightarrow$

$\Rightarrow \underline{\Omega(n \cdot \log(n))}$

Q3: (Basic operation = comparing)

Complexity of partition:

$$\sum_{i=0}^{n} 1 = \underbrace{1+1+\dots+1}_{n \text{ times}} \Rightarrow n \approx O(n)$$

Complexity of swap algorithm: $O(0)$

Complexity of kth Largest:

$$T(n) = T(n/2) + n \rightarrow \text{Best case}$$

$$T(n/2) = T(n/4) + n/2 \qquad T(n) = T(n/4) + n/2 + n$$

$$T(n/4) = T(n/8) + n/4$$

$$\vdots$$

$$T(n) = T(n/2^k) + n/2^k + \dots + n$$

$$T(1) = 1 \Rightarrow n/2^k = 1 \qquad T(n) = T(1) + n\left(\frac{1}{2^{\log_2 k}} + \dots + 1\right)$$

$$\Rightarrow n = 2^k$$

$$\Rightarrow k = \log_2 n \qquad = n \approx O(n)$$

$$T(n) = T(n-1) + n \rightarrow \text{worst case}$$

$$= T(n-1) + T(n-2) + n + n - 1$$

$$\vdots$$

$$T(n-1) + T(n-2) + \dots + T(1) + n + n - 1 + \dots + 1$$

$$\approx O(n^2)$$

Q_4: For Conquer and count function

$$\sum_{i=0}^{n/2} \sum_{k=n/2}^{n} 1 = \sum_{i=0}^{n/2} \underbrace{1+1+\cdots+1}_{n/2 \text{ times}} = \sum_{i=0}^{n/2} n/2 = \underbrace{n/2 + n/2 + \cdots + n/2}_{n/2 \text{ times}}$$

$$= \frac{n^2}{4} = O(n^2)$$

For divide function:

$$T(n) = 2T(n/2) + n^2$$

applying masters theorem:

$a=2 \quad b=2 \quad k=2 \quad p=0$

$a < b^k \Rightarrow \underline{O(n^2)}$

Q_5:

For Brute force algorithm

$$\sum_{i=0}^{n} 1 = \underbrace{1+1+\cdots+1}_{n \text{ times}} \Rightarrow \underline{O(n)}$$

For divide & conquer

$$T(n) \begin{cases} 2T(n/2) + 1 & n > 1 \\ 1 & n = 1 \end{cases}$$

$T(n) = 2(2T(n/2^2)+1)+1$

$T(n) = 2(2(2 \cdot T(n/2^3)+1)+1)+1$

$\vdots$

$T(n) = 2^k \cdot T(n/2^k) + 2^k + 2^{k-1} + \cdots + 1$

$T(n/2) = 2T(n/2^2) + 1$

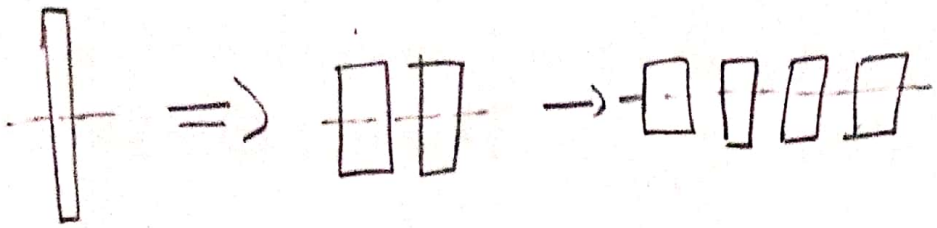
$T(n/2^2) = 2T(n/2^3) + 1$

$n/2^k = 1 \Rightarrow n = 2^k \Rightarrow k = \log_2 n$

$$\overbrace{2^{\log_2 n}}^{n} \cdot \overbrace{T(n/2^{\log_2 n})}^{1} + (1 - 2^{\log_2 n})/(1-2) = n + n = 2n \Rightarrow \boxed{O(n)}$$

explainings:

Q_1: For example a wire wants to cut for 8 piece process will like



So there are 3 cut.

it wire wanted to cut 9 pieces there will be 4 cutting and I revealed that relation between cut and piece counts are related like $2^k \leq n \leq 2^{k+1}$ for that I divide n for 2 while it can divide.

Q_2: It is mergesort algorithm and returning the best and worst results.

Mergesort = dividing array 2 piece while it can divide after that sorting the divided pieces. and sorting divided arrays with themselves while conneding again.

Q_3: Question 3 uses partition function like quicksort partition function sets an element to its correct position and put largest of its right, smallest to the left. And returns the index of setted element.

If partition element equals wanted value (kth) it means code found value and it can return the value. else if k is bigger than index, code looks for right. difference between quicksort is code do not have to check left side because wanted value is on right.

Note: I write to find smallest latter that I realized to find largest then I wrote a helper function for main.

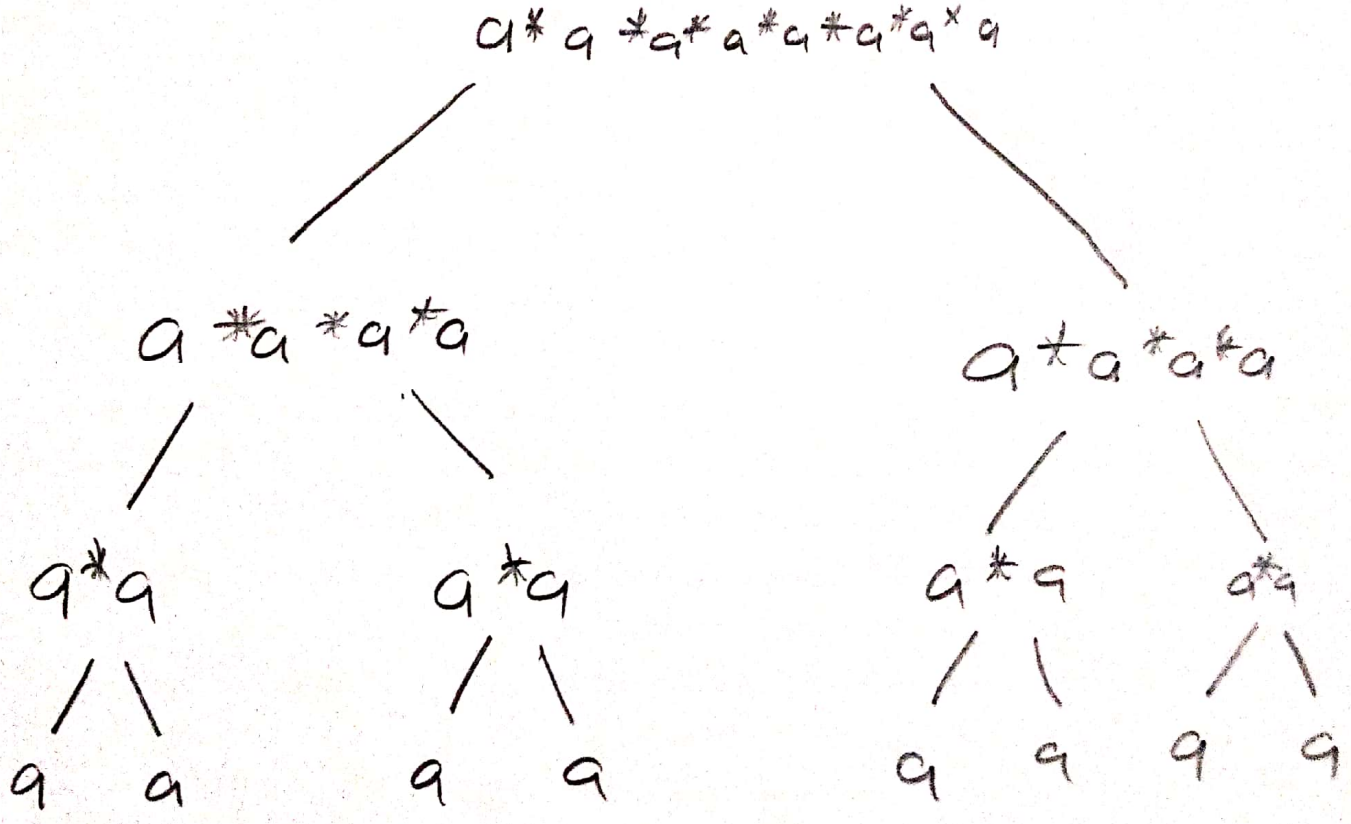
Q4: divide function for divide the array.

left inv + right inv + conquerAndCount returns the total inversion count.

conquerAndcount looks to left and right and compares the values. And no repetation of comparing.

Q5: Brute force algorithm = $\underbrace{a*a*+\cdots+a*}_{n \text{ times}}$

Divide and conquer:



(returning the products of a's every time)