3. Quicksort

Quicksort $QS(S)$

1. If $|S| = 1$ return S itself.
2. Pick a pivot element y
3. Compare elements with y and and let S_1 be the set of elements $< y$ and S_2 be the set of elements $> y$.

$$T(S) = T(S_1) + T(S_2) + \text{time taken for merging and comparisons}$$

The time taken for merging and comparisons can be considered to be $O(n)$

What is $T(S_1), T(S_2)$? That depends upon the pivot.
Assume we decide that we'll take the pivot as the first element, and we're using QS on a sorted array, it would lead to S_1 always being an empty set, and S_2 being $n-1, n-2, \ldots, 1$ elements with each depth of the recursion. As each step in the recursion takes $O(n)$ time, it would lead to the running time being $O(n^2)$.

Therefore, if we deterministically select the pivot as the first or last element, it has a worst case time complexity of $O(n^2)$

Worst Case Analysis

$$\max_{I \in \text{Input instances}} \{A(I)\}$$

In simpler terms, it's the max running time over all input instances for a certain algorithm.

So if we make an array of $I_1, I_2, \ldots, I_t$, where I_i is an array of length n, and we run the algorithm we're analyzing A on each of them, and the max running time is the worst case input.

For an instance of input I_j, we can find the algorithm that gives the best running time, similarly for a certain algorithm we can find the best input instances.

Now, we'll look at a randomized version of this algorithm which gives us a better *expected running time*.

RandQS

1. Pick a pivot at **random**.
2. We get the sets S_1, S_2

In quicksort, we aim to have

1. $|S_1| \approx |S_2|$.
2. Minimize the number of comparisons.
   These two aims are somewhat equivalent.

This will lead to a recurrence relation in expectation of

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

Resulting in an $n \log n$ algorithm in expectation.
(This is just an illustration and not a rigorous analysis)

Analysis of RandQS

Now we'll aim to analyze RandQS.
Let's define a random variable X_{ij}
Say we have the set $S = \{a_1, a_2, \dots, a_n\}$, upon which sorting we get the array $\{b_1, b_2, \dots, b_n\}$

We define X_{ij} as

$$X_{ij} = \begin{cases} 1 & \text{if } b_i \text{ and } b_j \text{ are compared at some point of the run} \\ 0 & \text{otherwise} \end{cases}$$

We can say that X_{ij} is a random variable as it's dependent upon b_i, b_j ever being a pivot, which is a random event.

The total number of comparisons is given by

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}$$

Therefore, the expected number of total comparisons is

$$E\left[\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} X_{ij}\right]$$

And using the linearity of expectation, we get

$$\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} E[X_{ij}]$$

Since

$$E[X_{ij}] = 1 \times Pr(X_{ij} = 1)$$

We can write the expected number of comparisons instead as

$$\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} Pr[X_{ij} = 1]$$

Claim:

$$Pr[X_{ij} = 1] = \frac{2}{j-i+1}$$

$X_{ij} = 1$ when either b_i or b_j is a pivot.

We want to ensure that b_i, b_j is chosen as a pivot before $\{b_{i+1}, b_{i+2}, \ldots, b_{j-1}\}$. This is because if any of them are picked, it would result in b_i, b_j being put into different sets and recursively solved independently, hence they wouldn't ever be compared in subsequent steps.

Now, lets consider the elements $\{b_i, b_{i+1}, \ldots, b_{j-1}, b_j\}$ There are a total of $j - i + 1$ elements, and we are only fine with 2 of these values being picked.
Hence, it results in the probability $\frac{2}{j-i+1}$

Expected Comparisons

Now, we can rewrite the expected number of comparisons as

$$= \sum_{i=1}^{j=n-1}\sum_{j=i+1}^{n} \frac{2}{j-i+1}$$
$$= \sum_{i=1}^{j=n-1}\sum_{k=2}^{n-i+1} \frac{2}{k} \leq \sum_{i=1}^{n}\sum_{k=1}^{n} \frac{2}{k} = 2n\sum_{k=1}^{n} \frac{1}{k} \approx n \ln n$$

Hence, the expected running time is $O(n \log n)$, which is a step up from the worst case of $O(n^2)$

Probability of deviation from expected running time

We can't only be satisfied with the expected running time of an algorithm though, we need to analyze the probability it deviates from the expected running time as well.

Now, we'll try to find the probability of the running time be $> 10 \cdot E[\text{Running time}]$, which we can find using Markov's Inequality.

$$Pr[X > 10 \cdot E[X]] \leq \frac{E[X]}{10 \cdot E[x]} = \frac{1}{10}$$