1 Naive Quicksort

1: function QUICKSORT(list)
2:   if len(list) = 1 or 0 then
3:     return list
4:   end if
5:   pivot ← list[0]
6:   low, high ← PARTITION(list, pivot)
7:   low ← QUICKSORT(low)
8:   high ← QUICKSORT(high)
9:   return low + pivot + high
10: end function

Quicksort is a recursive sorting algorithm that begins by first selecting a pivot element, in this case the leftmost element. We partition the list into two arrays called low and high, with elements less than and greater than our pivot respectively. We sort this recursively using Quicksort, then combine the lists together with our pivot in the middle. In the base case when the list to be sorted has only one element, we simply return the list.

Since the element selected as the pivot at the beginning of the list varies depends on the list itself, we can select "good" and "bad" pivots. A "good" pivot is balanced and separates the list into problems of equal size to be fed back into Quicksort (i.e. it is the middle element). We can write this recurrence relation as:

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

\[ = O(n \log n) \]

This is our best case. In our worst case, we select a "bad" pivot that is maximally unbalanced and separates the list into one very large problem and one very small problem. Considering our algorithm above, the very worst case would be selecting the minimum element, and feeding back a problem of size \((n - 1)\) back into Quicksort. We can write this recurrence relation as:

\[ T(n) = T(n - 1) + O(n) \]

\[ = O(n^2) \]

When will we select a "bad" pivot? If our data is already sorted, we will consistently select our min value and our runtime will approach \(O(n^2)\). Much of our data in the real world comes in some sort of sorted form. For example, if we are servicing requests to a server, due to varying network latencies they may not arrive exactly in the same order that they are requested. However, our data will usually be almost sorted - and our version of Quicksort above will perform very poorly.
2 Randomized Quicksort

One optimization we can make is selecting our pivot at random, rather than deterministically. By doing so we can select "good" pivots even when our data is already sorted. However, we now also select "bad" pivots occasionally on randomly sorted data. How can we reliably analyze our new algorithm to determine how long it takes on average?

We begin our analysis by considering what quantity that we are trying to determine. For Quicksort, our runtime is determined by the number of comparisons that we make between two elements. We can consider some indicator variable $X_{i,j}$, which is 1 if the $i^{th}$ smallest element $e_i$ and $j^{th}$ smallest element $e_j$ are compared, since we never compare any two elements more than once. Then the total number of comparisons $X$ across all possible combinations of $i$ and $j$ can be given by

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

Our average number of comparisons is then the expected value of $X$, which is given by

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

How can we calculate $E[X_{i,j}]$ based on the values of $i$ and $j$? Let us consider three separate cases: if our pivot is selected between $e_i$ and $e_j$, then they will be separated into different bins and will never be compared. Thus $X_{i,j} = 0$. If our pivot is either exactly $e_i$ or $e_j$, then we are guaranteed that $i$ and $j$ will be compared because every element in the partition will be compared to the pivot, so $X_{i,j} = 1$. If our pivot is either smaller than $e_i$ or larger than $e_j$, then $E[X_{i,j}]$ is not affected because we simply defer the decision as $e_i$ ... $e_j$ are both dumped into the same partition.

Thus the probability that our indicator value $X_{i,j}$ is 1 given that our sorting algorithm ends, is then the probability that we select exactly either $i$ or $j$ as pivots out of the possible pivots from $[i..j]$. There are a total of $j - i + 1$ total values we can select from in this range, so our probability is then $\frac{2}{j-i+1}$. We can then reduce our expression to:

$$E[X] = \sum_{i=1}^{n-1} \frac{2}{2} + \frac{2}{3} + \frac{2}{4} + ... + \frac{2}{n - i + 1}$$

$$= \sum_{i=1}^{n-1} 2\left(\frac{1}{2} + \frac{1}{3} + \frac{1}{4} + ... + \frac{1}{n - i + 1}\right)$$

$$= \sum_{i=1}^{n-1} 2\mathcal{O}(\log n)$$

$$= \mathcal{O}(n \log n)$$

Note that $\frac{1}{2} + \frac{1}{3} + \frac{1}{4} + ... + \frac{1}{n} = \int_{2}^{n} \frac{1}{x} dx = \log(n) - \log(2) \in \mathcal{O}(n)$.

Thus we can see as our values of $n$ get large enough, our randomized Quicksort approaches an average performance of $\mathcal{O}(n \log n)$.