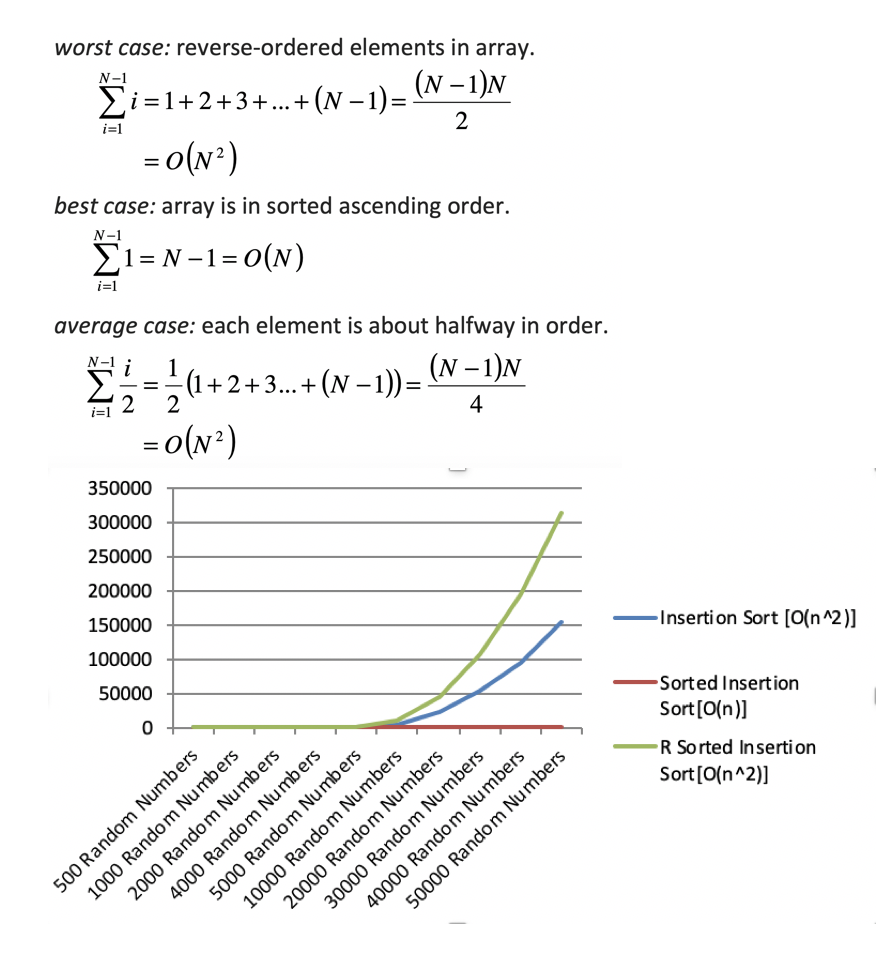
**Insertion Sort**

The **worst** case for Insertion Sort occurs when the array is in reverse order. To insert each number, the algorithm will have to shift over that number to the beginning of the array. Therefore, The maximum number of comparisons for an insertion sort will be the sum of the first n−1 integers which is O(n2).

In the **best case**, where the array was already sorted, no element will need to be moved, so the algorithm will just run through the array once and return the sorted array. The running time would be directly proportional to the size of the input, so we can say it will take  O(n) time.

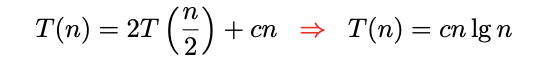
On **average**, half the elements in a list A1 ... Aj are less than element Aj+1, and half are greater. Therefore, the algorithm compares the (j + 1)th element to be inserted on the average with half the already sorted sub-list, so tj = j/2. Working out the resulting average-case running time yields a quadratic function of the input size, just like the worst-case running time.



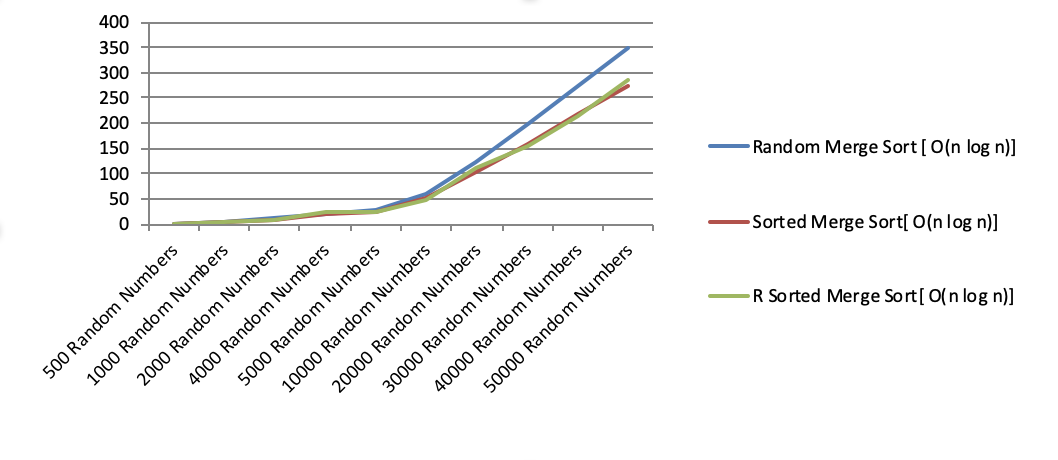
Merge Sort

In Merge Sort, we need to consider the two distinct processes that make up its implementation. First, the list is split into halves. We already computed (in a binary search) that we can divide a list in half logn times where n is the length of the list. The second process is the merge. Each item in the list will eventually be processed and placed on the sorted list. So the merge operation which results in a list of size n requires n operations. The result of this analysis is that logn splits, each of which costs n for a total of nlogn operations. In, merge sort, the best, worst, and average cases are similar.

The Θ(n log n) **best case**, **average case**, and **worst-case** complexity because the merging is always linear. Recall the basic recurrence:

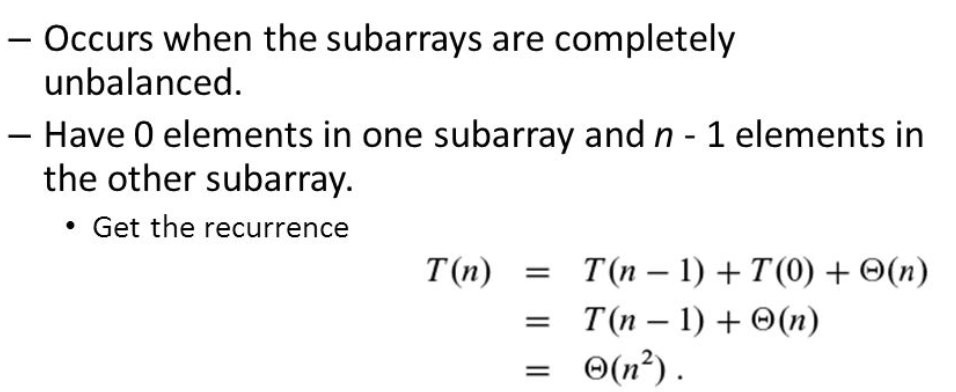


Therefore, merge sort is an O(nlogn) algorithm.



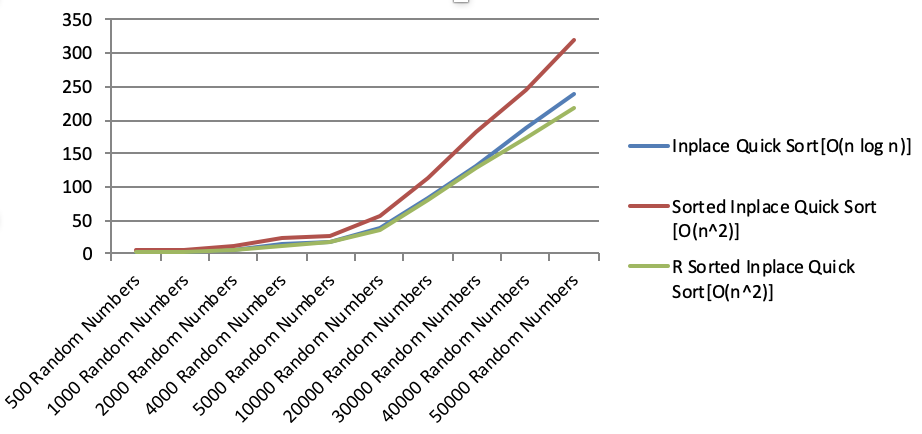
In-Place Quicksort

The **worst case** occurs when the partition process always picks greatest or smallest element as pivot. If we consider above partition strategy where last element is always picked as pivot, the worst case would occur when the array is already sorted in increasing or decreasing order.



The **best case** occurs when the partition process always picks the middle element as pivot

In **average case,** to sort an array of n distinct elements, quicksort takes O(n log n) time in expectation, averaged over all n! permutations of n elements with equal probability.



Modified Quick Sort

In Modified Quick Sort, we select the pivot using the "median of three" method instead of picking the middle element as the pivot.

The **worst case** quick-sort happens when the pivot we picked turns out to be the least element of the array to be sorted, in every step (i.e. in every recursive call). A similar situation will also occur if the pivot happens to be the largest element of the array to be sorted.

The **best case** of quick sort occurs when the pivot we pick happens to divide the array into two exactly equal parts, in every step.

In **average case**, we assume that each of the sizes for S1 is equally likely. This assumption is valid for our pivoting (median-of-three) strategy. Therefore, On average, the running time is O(N log N).

