By: *Aastha Sharma*

Worst case analysis of Merge sort Quick sort and Binary Search.

What is the running time of an algorithm ?

As we all know that the running time of an algorithm is the time which is required to complete an algorithm, which means it depends on the number of operations which has to be performed or executed in the algorithm. The running time of an algorithm increases when the number of operations increases. Sometimes the size of the input is same,but the running time is different for different different inputs, when this type of Situations comes we perform - best case, average case and worst case analysis of an algorithm The best case analysis gives the best time to execute an algorithm which time means the best case gives the minimum time to execute an algorithm and the average case gives the average time required to perform an algorithm and the worst case gives the maximum time to execute the algorithm.

In this article we will discuss about the Worst case analysis of merge sort, quick sort and binary Search algorithm.

Worst case Analysis

So what is Worst case Analysis ?

As it's name suggests worst case which means the maximum time required to execute an algorithm is said to be the worst case And the time required is known as the worst case time analysis.

By: *Aastha Sharma*

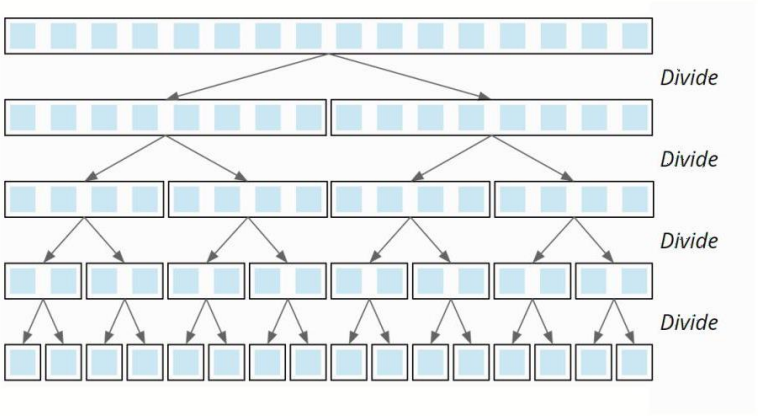
In real life, most of the time we do worst case analysis of an algorition. Worst case running time is the longest running time for any imput of size m. Knowing the worst case performance of an algorithm provides a guarantee that it will never take any time longer.

Merge Sort

IT is an Divide and Conquer algorithm in which the problem keeps on dividing into two,individual problems until its end point, where it can not be divided more and after that when the solution of each problem is ready then it perform "Merge and Sort where each Sub problem is merge in an sorted Order to required result of main problem.

Merge sort performs two steps.

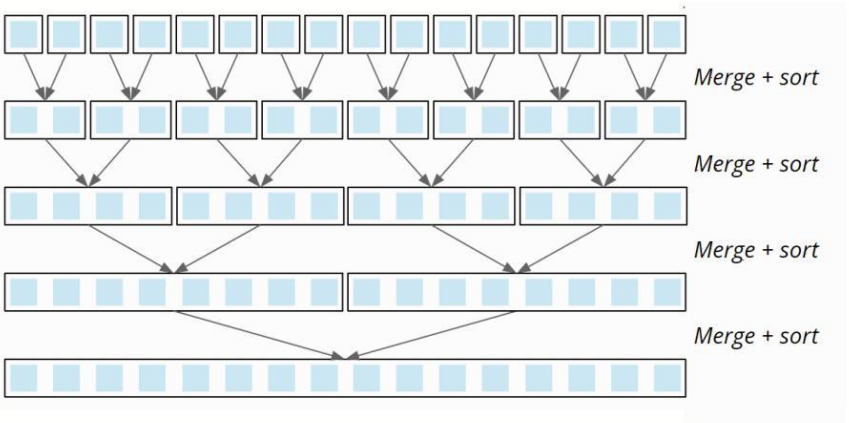
1- Divide- In which the problem is Subdivided into two halves and then the subproblem is again divided into two halves until it reaches to the single element, where the size of subproblem is 1.



The time complexity of division function of array above (having 16 elements and n= 16) is 8+4+2+1 = 15. In other words, when the size of the array is n , the time complexity of the divide function of Merge Sort is (n/2+ n/4 ...till 1) which is also $O(n)$.

2- Merge&sort - In this step the subproblem has to be merged in a sorted order to achive the final result of main problem.

By: *Aastha Sharma*



This algorithm loops through $log(n) - 1$ times and the time complexity of every loop is $O(n)$, so the time complexity of the entire function is $O(n \times (log(n)))$.

The complexity of the entire algorithm is the sum of the complexity of two steps which is $O(n \times logn)$. This happens to be the worst case of Merge Sort as well.

When will be the worst case of mege sort occurs ?

The worst case of merge sort occurs when you have to do maximum number of comparisons.

When every element of subarray compared at least once, then it will gives the worst case time running time of merge sort.

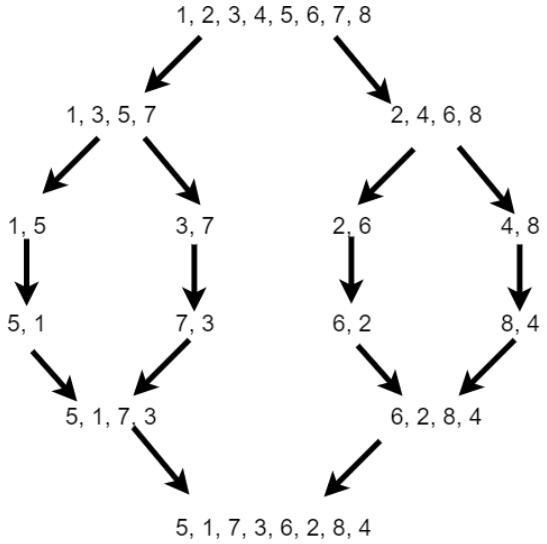
For this the left and right sub arrays should store alternate element of sorted array.

Consider an example - [1 2 3 4 5 6 7 8] is an Sorted array.

for the worst case left & right,Subarray should have alternate elements of these sorted list i.e left sublist should be[1 3 5,7] & right subarray should be [0,2,4,6]

So that every element of list Sub list will be compared at least once & gives the maximum number of comparison which will results in worst case complexity.
For this list Input array should be [4,0,6,2,5,1,7,3]

If we use this algorithm on array {1,2,3,4,5,6,7,8}, we'll find {5,1,7,3,6,2,8,4} as the combination that'll produce worst complexity for merge sort as shown below:



In this article, we discussed the worst time complexity of Merge Sort, which is $O(n \times logn)$. It occurs when the left and right sub-arrays in all merge operations have alternate elements.

Quick Sort

It is an divide & conquer Sorting method, in which we sort list by using an Pivot element which means we Select an element as an Pivot and then divide the list around the Pivot element and recur for Subarray on left of light of pivot.

When Does the Worst Case of Quicksort Occur?

IT depends on the strategy for choosing pivot. In early versions of Quick Sort where the leftmost (or rightmost) element is chosen as a pivot, the worst occurs in the following cases.

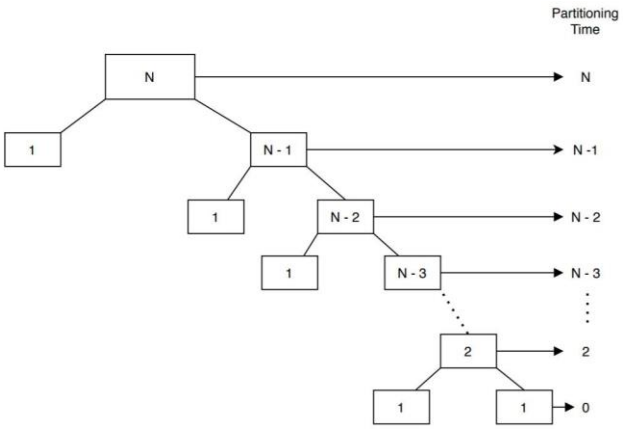
1) Array is already sorted in the same order.
2) Array is already sorted in reverse order.
3) All elements are the same (a special case of cases 1 and 2)

Let's assume the input of the Quicksort is a sorted array and we choose the leftmost element as a pivot element. In this case, we'll have two extremely unbalanced arrays. One array will have one element and the other one will have $(N-1)$ elements.

Similarly, when the given input array is sorted reversely and we choose the rightmost element as the pivot element, the worst case occurs. Again, in this case, the pivot elements will split the input array into two unbalanced arrays.

Except for the above two cases, there is a special case when all the elements in the given input array are the same. In such a scenario, the pivot element can't divide the input array into two and the time complexity of Quicksort increases significantly.

Let's assume that we choose a pivot element in such a way that it gives the most unbalanced partitions possible:



All the numbers in the box denote the size of the arrays or the subarrays.

Let's consider an input array of size N. The first partition call takes N times to perform the partition step on the input array.

Each partition step is invoked recursively from the previous one. Given that, we can take the complexity of each partition call and sum them up to get our total complexity of the Quicksort algorithm.

Therefore, the time complexity of the Quicksort algorithm in worst case is

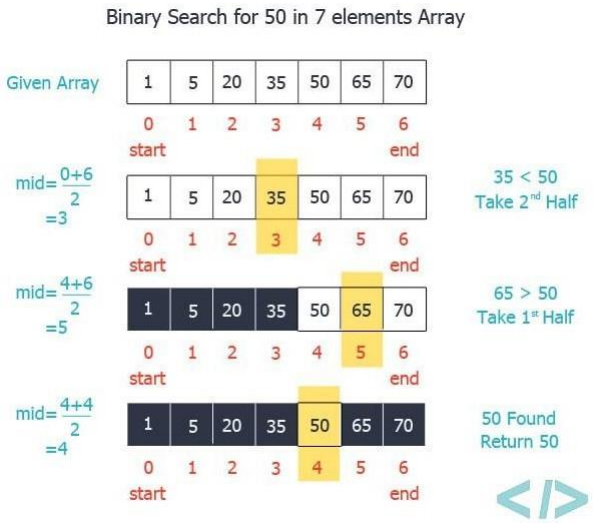
$$[\mathbf{N} + (\mathbf{N}-\mathbf{1}) + (\mathbf{N}-\mathbf{2}) + (\mathbf{N}-\mathbf{3}) + ... + \mathbf{2}] = [\frac{\mathbf{N(N+1)}}{\mathbf{2}} - \mathbf{1}] = \mathcal{O}(\mathbf{N^2})$$

Binary Search

Binary search is the search technique that works efficiently on sorted lists. Hence, to search an element into some list using the binary search technique, we must ensure that the list is sorted.

By: *Aastha Sharma*

Binary search follows the divide and conquer approach in which the list is divided into two halves, and the item is compared with the middle element of the list. If the match is found then, the location of the middle element is returned. Otherwise, we search into either of the halves depending upon the result produced through the match.



When worst case of Binary Search occurs ?

When The element is to search is in the first index or last index

In this case, the total number of comparisons required is logN comparisons. the worst case occurs, when we have to keep reducing the search space till it has only one element. The worst-case time complexity of Binary search is O(logn).

The first or last element will give the worst case complexity in binary search as you'll have to do maximum no of comparisons. Example:

1	2	3	4	5	6	7	8	9

Here searching for 1 will give you the worst case, with the result coming in 4th pass.

1	2	3	4	5	6	7	8

In this case, searching for 8 will give the worst case, with the result coming in 4 passes.

In the second case searching for 1 (the first element) can be done in just 3 passes. (compare 1 & 4, compare 1 & 2 and finally 1)

So, if no. of elements are even, the last element gives the worst case.

How many iterations are there? After the first iteration, the number of active elements is at most n/2. After another, the number is at most n/4.

In general, after i iterations, the number drops to at most n/2 i . Suppose that there are h iterations in total. It holds that (think: why?) n 2 h ≥ 1 which gives h ≤ log2 n.

It thus follows that the worst-case time of binary search is at most f2(n) = 2 + 6 log2 n. This is a performance guarantee that holds on all values of n.

Conclusion

1. **Worst Case Time Complexity of Merge sort is** O(N logN).
2. **Worst Case Time Complexity of Quick sort is** $O(n^2)$.
3. **Worst Case Time Complexity of Binary Search is** O(log(n)).