**TIME COMPLEXITY**

1. Time Complexity can also be elaborated by estimating the counts of the number of elementary steps that are performed by any algorithm to finish the execution.
2. We can see that in each iteration, the number will be reduced to the previous half until the whole condition is violated. Such algorithms of breaking a number or set of numbers into halves fall under logarithmic complexity O(log N).

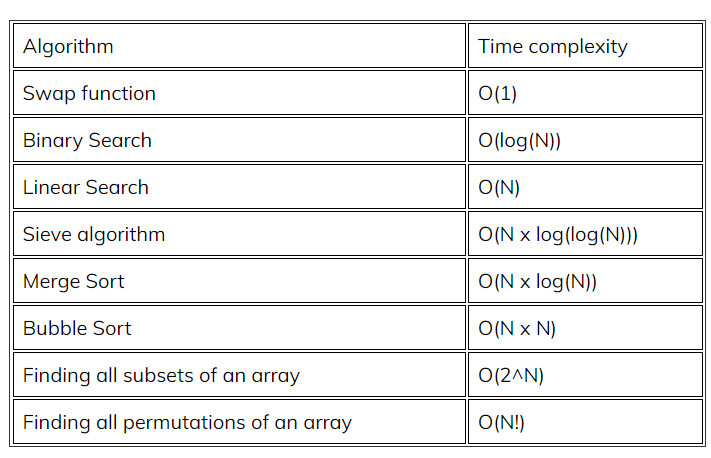
while(n>0)

{

cout<<N<<" ";

N=N/2;

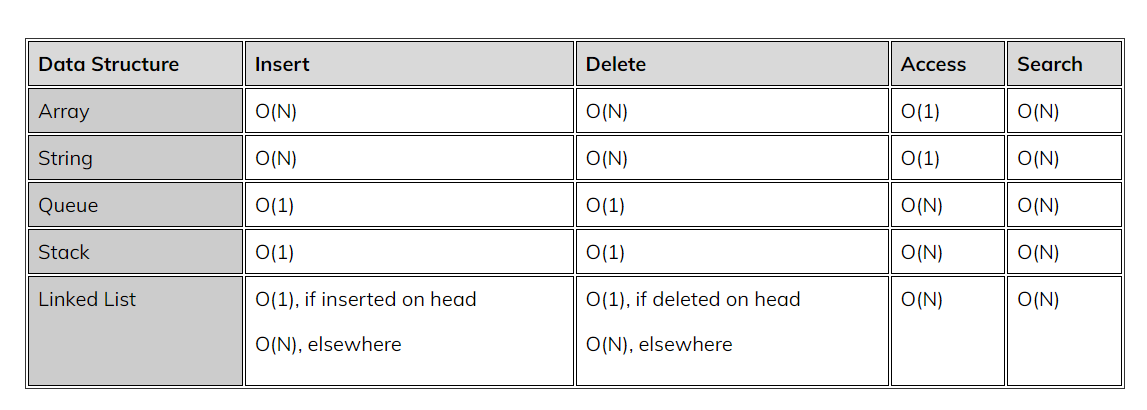
}

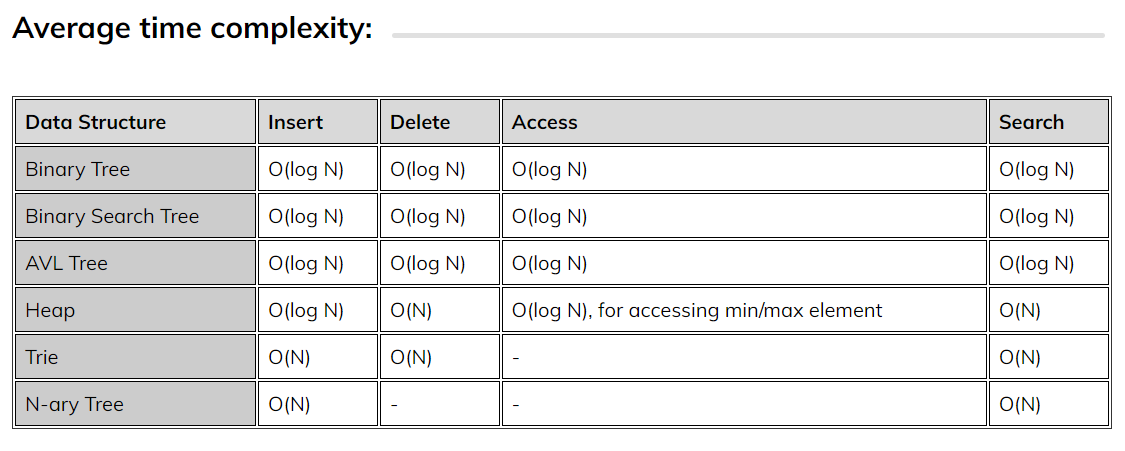


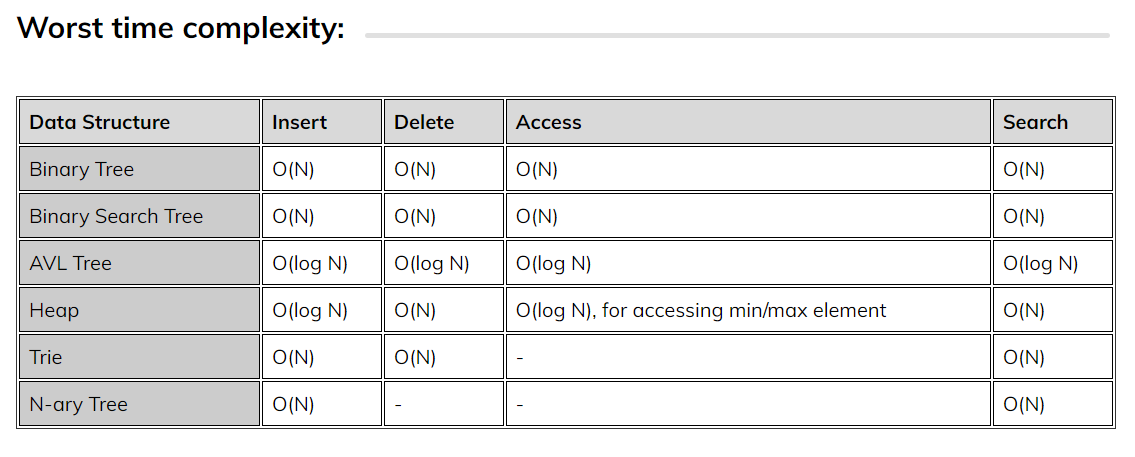
1. O(1) > O(log(n)) > O(n) > O(n x log(n)) O(n2) > O(n3) > **O(2n) > O(n!)** > O(nn)
2. **List an example where increasing the Space complexity led to improvements in time complexity.**  
   A famous example for this case is merge sort that consumes O(n) memory and O(n \* log(n)) time, while bubble sort takes O(1) memory, but you have to instead pay in the form O(n2) time. This tradeoff allows a significant boost from an O(n2) time algorithm to an O(n \* log(n)) time algorithm. There are an almost infinite number of such cases, for example, all dynamic programming algorithms.
3. **Which algorithm is better, one with time complexity O(2n) or one with time complexity O(n!)?**

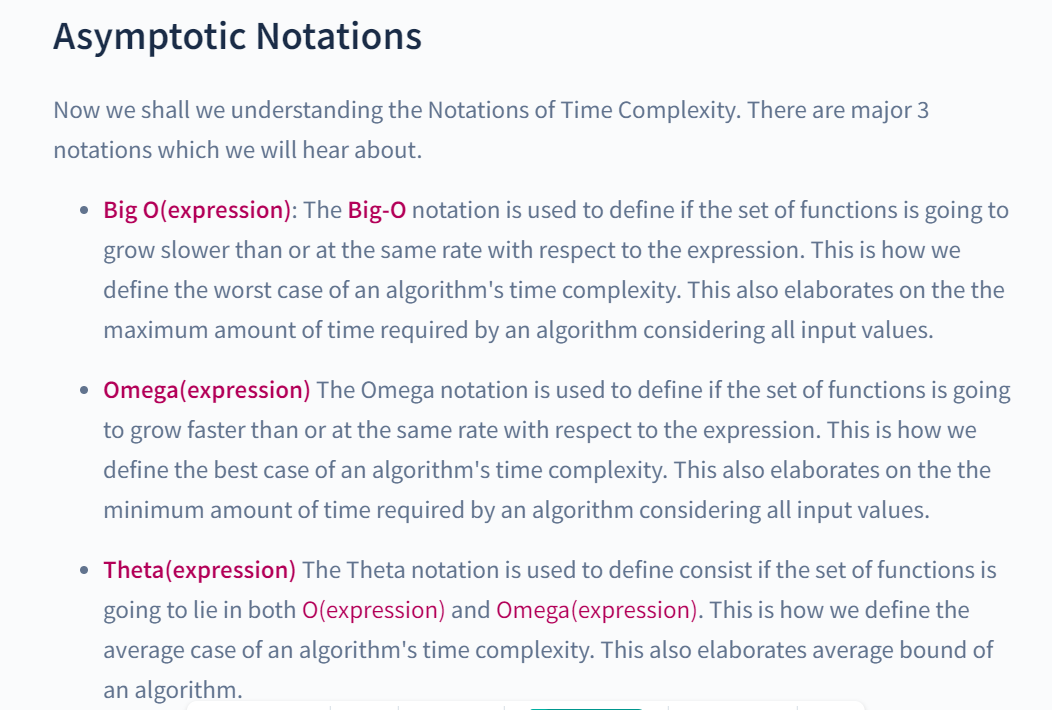
**(*Explain with an example*)**

1. **What is the need to optimize time and space?**Optimization of time is essential in writing efficient programs. A merge sort on normal computers can beat bubble sort on supercomputers. Similarly, using large memory can also result in runtime errors.  
   Thus the efficiency of programs plays a vital role. A good program is both time and space-efficient.
2. pow(i,n) takes log(n) time complexity as it uses binary exponentiation
3. if two recursive calls take place together as in the Fibonacci series then the time complexity is O(2^n)

****

****

****



**SPACE COMPLEXITY**

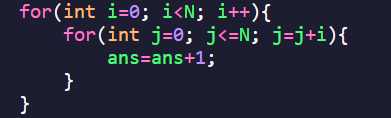
1. Auxiliary space is the extra space used by an algorithm, which gives us space complexity on adding with the space taken by the input values.

Space complexity = Auxiliary Space + Space taken by input values

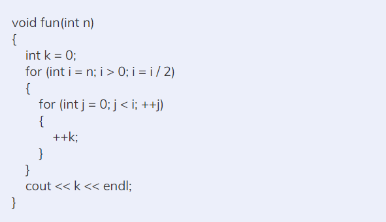
1. Stack Memory: It’s a memory usage method that allows temporarily accessing and storing memory chunks in a last-in-first-out manner, i.e., one can access the last inserted chunk of data first.
2. Value of n and TC until which it can be done. CPU does 108 operations in 1 sec
3. n <= 10 O(n!)
4. n <= 25 O(2n)
5. n <= 100 O(n4)
6. n <= 1000 O(n3)
7. n <= 105  O(nlogn)

**QS ON TIME AND SPACE COMPLEXITY**

1. Time complexity will be O(n\*logn)
   1. Whenever you see a loop for j=j+i so in such scenarios consider the loop as (j/i)
   2. And complexity for j/i until n is O(logn)



1. TC of O(logn)\*O(logn) is always O(n)
   1. Dry run with an example to understand better



1. For any recurrence relation

Time complexity is O((no. of recursive call)n)