**Asymptotic Notation (Revision)**

Asymptotic complexity is the equivalent idealization for analysing algorithms; it is a strong indicator of performance on large-enough problem sizes and reveals an algorithm's fundamental limits.

The asymptotic behaviour of a function f(n) refers to the growth of f(n) as n gets large. We typically ignore small values of n, since we are usually interested in estimating how slow the program will be on large inputs.

The slower the asymptotic growth rate, the better the algorithm (although this is often not the whole story).

**Worst-Case and Average-Case Analysis**

An algorithm runs in time T(n), T(n) is an upper bound on the running time that holds for all inputs of size n. This is called worst-case analysis. The algorithm may very well take less time on some inputs of size n, but it doesn't matter. Maximum time is known as quadratic (Worst-case).

**Worst case -** the largest element would be in the middle node of the list meaning half of the list has to be traversed to find it therefore O(n/2) is the complexity.

**Best Case –** the largest element would be present in the beginning if the list is sorted, complexity would be O(1).

A popular alternative to worst-case analysis is average-case analysis. In average case we calculate the expected time spent on a randomly chosen input. A good example of this is the popular quicksort algorithm, the worst-case running time on an input sequence of length n is proportional to n2 but the expected running time is proportional to n log n.

**Practical Example**

In bubble sort, when the input array is already sorted, the time taken by the algorithm is linear (the best case).

But, when the input array is in reverse condition, the algorithm takes the maximum time (quadratic) to sort the elements (the worst case).

When the input array is neither sorted nor in reverse order, then it takes average time. These durations are denoted using asymptotic notations.

**Big-O Notation (O-notation)**

Big-O notation represents the upper bound of the running time of an algorithm. It gives the worst-case complexity of an algorithm. Since it gives the worst-case running time of an algorithm, it is widely used to analyse an algorithm as we are always interested in the worst-case scenario. We compute the Big-O of an algorithm by counting how many iterations an algorithm will take in the worst-case scenario with an input of N.

**Omega Notation (Ω-notation)**

Omega notation represents the lower bound of the running time of an algorithm. It provides the best-case complexity of an algorithm. We compute the big-Ω by counting how many iterations an algorithm will take in the best-case scenario based on an input of N

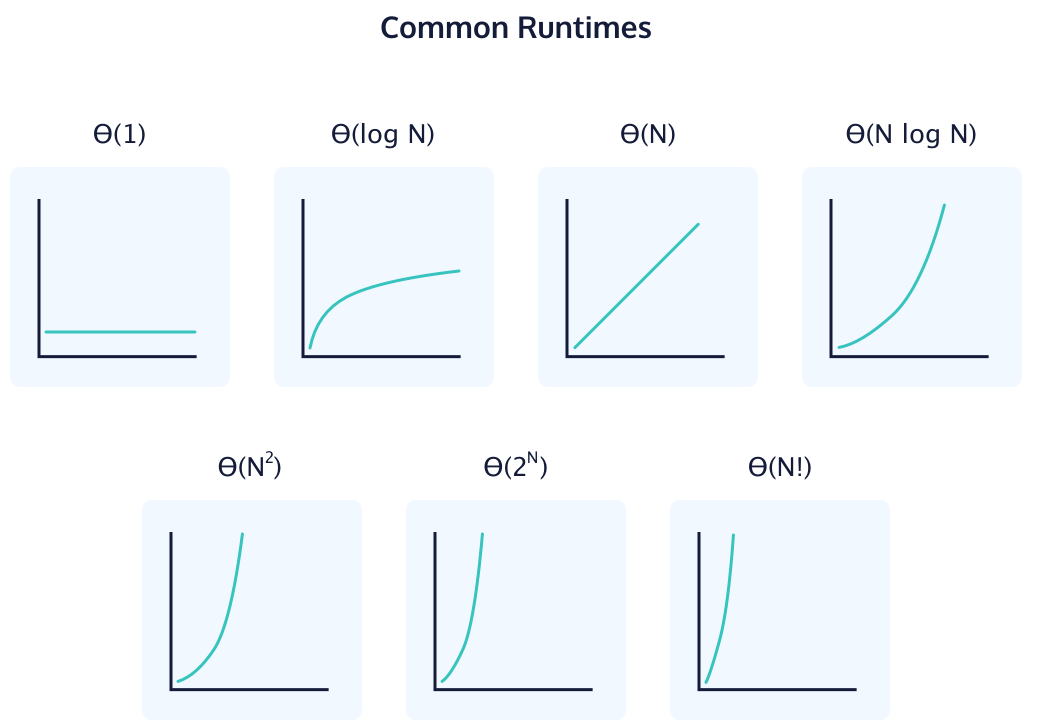
**Theta Notation (Θ-notation)**

Theta notation encloses the function from above and below. Since it represents the upper and the lower bound of the running time of an algorithm, it is used for analysing the average-case complexity of an algorithm.

**Algorithmic Common Runtimes**

The common algorithmic runtimes from fastest to slowest are:

* constant: Θ(1)
* logarithmic: Θ(log N)
* linear: Θ(N)
* polynomial: Θ(N^2)
* exponential: Θ(2^N)
* factorial: Θ(N!)



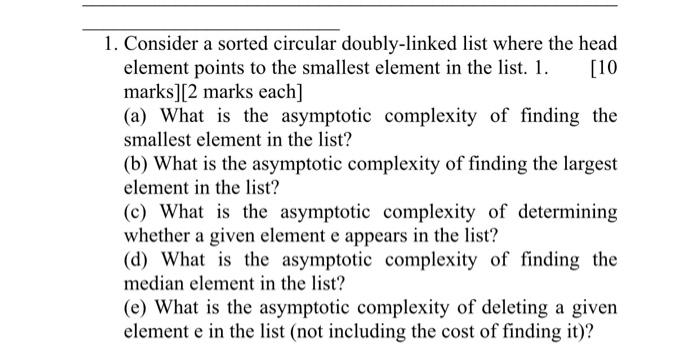
**Text

Description automatically generatedExam Questions**

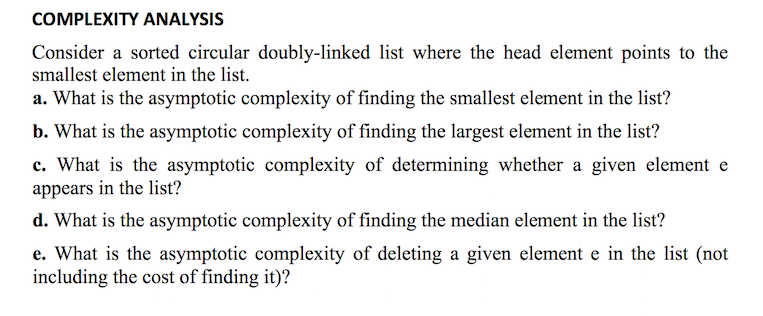
**Answer:** O(N + M) time, O(1) space

**Explanation:** The first loop is O(N) and the second loop is O(M). N and M are independent variables so we can’t say which one is leading term. Therefore, Time complexity of the given problem will be O(N+M).

Since variables size does not depends on the size of input, therefore Space Complexity will be constant or O(1)



1. O(1) as doubly linked list is sorted, the smallest element is present at the first location can be accessed direct through head pointer in one go
2. O(n) as the list is sorted, the largest element is present in the last of the list. We must traverse the whole list, to reach there which in turns gives us time complexity of O(n) where n is number of nodes in a list.
3. O(n) is the worst case as we may have to traverse the whole linked list, the element may be present at end. It can also be at the beginning; however, we always consider the worst case; meaning the asymptotic complexity is O(n).
4. O(n/2) as we only have to traverse half the list as we have to find middle element.
5. O(1) as we may already be present at that node which we want to delete and the cost of finding is not included.



1. **Worst Case -** the smallest element would be in the middle node of the list meaning half of the list has to be traversed to find it therefore O(n/2) is the complexity.

**Best Case -** they would be present in the beginning if the list is sorted, complexity would be O(1).

1. **Worst case -** the largest element would be in the middle node of the list meaning half of the list has to be traversed to find it therefore O(n/2) is the complexity.

**Best Case –** the largest element would be present in the beginning if the list is sorted, complexity would be O(1).

1. If the element is in the middle node of the list (worst case) half of the list must be traversed to find it therefore O(n/2) is the complexity.
2. If the list is sorted, then the medial element must be present in the middle of the list therefore the complexity of locating that element in the list is O(n/2).
3. Worst case - the element would be in the middle of the list therefore the complexity of the locating and the deletion of an element is O(n/2).

Best case - the element would be present in the beginning or the ending of the list therefore the complexity of locating and deleting an element is O(1).

Notes on Question - A circular double linked list is a linked list in which the first node of the linked list points to the last node through the previous pointer and the last node points to the first node through the next pointer.