|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |
|  |
| JAVA KEYWORDS  DOCUMENTATION    **SUBMITTED BY: METATITANS**  **Aadil A A**  **Meera Javad**  **Swetha S**  **Christo Shaji** |
|  |  |  |
|  |  |  |
|  |  |  |

Space Complexity

Space complexity refers to the amount of memory or space required by an algorithm to solve a problem. It is a measure of the amount of memory used by an algorithm to store data, temporary variables, and other information during the execution of the algorithm.

Space complexity is typically expressed in terms of the amount of memory required by an algorithm as a function of the input size. The input size is usually denoted by the variable n, and space complexity is denoted by the notation O(f(n)), where f(n) is some function of n that describes the amount of memory required by the algorithm.  
  
Types of space complexities:  
  
1. Constant Space Complexity: An algorithm has a constant space complexity if the amount of memory used by the algorithm is independent of the input size. For example, an algorithm that sorts an array in place has a constant space complexity of O(1).  
  
2. Linear Space Complexity: An algorithm has a linear space complexity if the amount of memory used by the algorithm is proportional to the input size. For example, an algorithm that creates an array of n elements has a linear space complexity of O(n).  
  
3. Quadratic Space Complexity: An algorithm has a quadratic space complexity if the amount of memory used by the algorithm is proportional to the square of the input size. For example, an algorithm that creates a two-dimensional array of size n x n has a quadratic space complexity of O(n^2).  
  
4. Exponential Space Complexity: An algorithm has an exponential space complexity if the amount of memory used by the algorithm grows exponentially with the input size. For example, an algorithm that generates all subsets of a set of n elements has an exponential space complexity of O(2^n).  
  
It is important to consider space complexity when designing and analyzing algorithms, especially for problems that involve large data sets. Algorithms with low space complexity are usually preferred, as they require less memory and can be executed more efficiently.  
  
  
How to calculate space complexity.  
  
To calculate the space complexity of an algorithm, you need to determine how much memory the algorithm requires as a function of the input size. Here are the steps to follow:  
  
Identify the variables and data structures used by the algorithm: Look for any variables or data structures used by the algorithm, such as arrays, lists, variables, and pointers.  
  
Determine the space requirements of each variable and data structure: Determine how much memory is required to store each variable and data structure. This will depend on the data type and size of the variable or data structure.  
  
Count the number of times each variable or data structure is used: Count how many times each variable or data structure is used in the algorithm. This will help you determine the overall space requirements of the algorithm.  
  
Sum up the space requirements of all variables and data structures: Add up the space requirements of all variables and data structures used by the algorithm. This will give you an estimate of the total space required by the algorithm.  
  
Express the space complexity in terms of the input size: Finally, express the space complexity as a function of the input size. This is usually done using big O notation, which gives an upper bound on the space requirements of the algorithm as the input size grows.  
  
For example, consider the following algorithm that creates an array of n integers and then iterates over the array:

int[] createArray (int n) {  
int[] arr = new int[n];  
for (int i = 0; i < n; i++) {  
arr[i] = i;  
}  
return arr;  
}  
To calculate the space complexity of this algorithm, we can follow these steps:  
  
Variables and data structures: The algorithm uses an integer variable n and an integer array arr.  
  
Space requirements: The integer variable n requires a constant amount of memory (e.g., 4 bytes), and the integer array arr requires n \* sizeof(int) bytes of memory.  
  
Variable usage: The variable n is used once to initialize the array, and the array arr is used n times in the loop.  
  
Total space requirements: The total space required by the algorithm is 4 + n \* sizeof(int) bytes.  
  
Space complexity: The space complexity of the algorithm can be expressed as O(n) since the space required grows linearly with the input size.  
  
  
  
  
Need for analyzing space complexity  
  
Space complexity is an important metric for evaluating the efficiency of an algorithm, as it measures the amount of memory required by the algorithm to solve a problem. Here are some reasons why space complexity is important:  
  
Memory constraints: Many computing systems, such as embedded systems, mobile devices, and cloud computing platforms, have limited memory resources. Algorithms with high space complexity may not be suitable for these systems and may cause the system to run out of memory or slow down.  
  
Time-space trade-off: There is often a trade-off between time complexity and space complexity. Algorithms with high space complexity may be faster than algorithms with low space complexity, and vice versa. By analyzing the space complexity of an algorithm, we can choose the best algorithm for a given problem based on the available resources.  
  
Algorithm optimization: Analyzing the space complexity of an algorithm can help identify opportunities for optimization. For example, if an algorithm requires a large amount of memory, we may be able to optimize the algorithm by reducing the number of data structures or variables used.  
  
Algorithm design: Space complexity can also influence the design of algorithms. For example, if we know that memory is a scarce resource, we may design algorithms that use constant space or optimize the use of memory.  
  
Overall, understanding space complexity is important for designing efficient algorithms that are well-suited to the available computing resources and constraints.