**Running Time**

1. **String Splitting Operation**
   1. String[] sentence = string1.split(" ").
   2. The split operation takes **O(n)** time, where n is the number of characters in the input string. The algorithm goes through the entire sentence (string) to separate it into individual words.
2. **Going over groups of words (three at a time) -** for (int i = 0; i < sentence.length; i+=3){
   1. **The for-loop runs O(m/3) times,** as m is the inputted string length (after splitting it by spaces). Since (m/3) iterations are proportional to the input size, this loop still operates in O(m) time.
   2. My algorithm performs constant-time comparisons **per three words** to find the shortest word. This takes **O(1) time**.
3. **Building the Final Sentence**
   1. for (int i = 0; i < shortestWords.size(); i++) {

finalSentence += shortestWords.get(i);

if (i < shortestWords.size() - 1) {

finalSentence += " ";

}

}

* 1. This **for-loop** has a time complexity of **O(k^2)**, where **k** is the **length of the final sentence.** Each concatenation creates a new string, leading to **quadratic complexity.**

1. **Overall Time Complexity**
   1. **Separating the String into Separate Words: O(n)**
   2. **Iterating Through Words: O(m) => where (m <= n)**
   3. **Building the Final Sentence: O(k^2)**

* Hence, this algorithm's overall time complexity is **O(n + k^2).**
* **In most cases, the final sentence (k) length will be smaller than n (the input size), but string concatenation would take longer than expected for more significant inputs.**