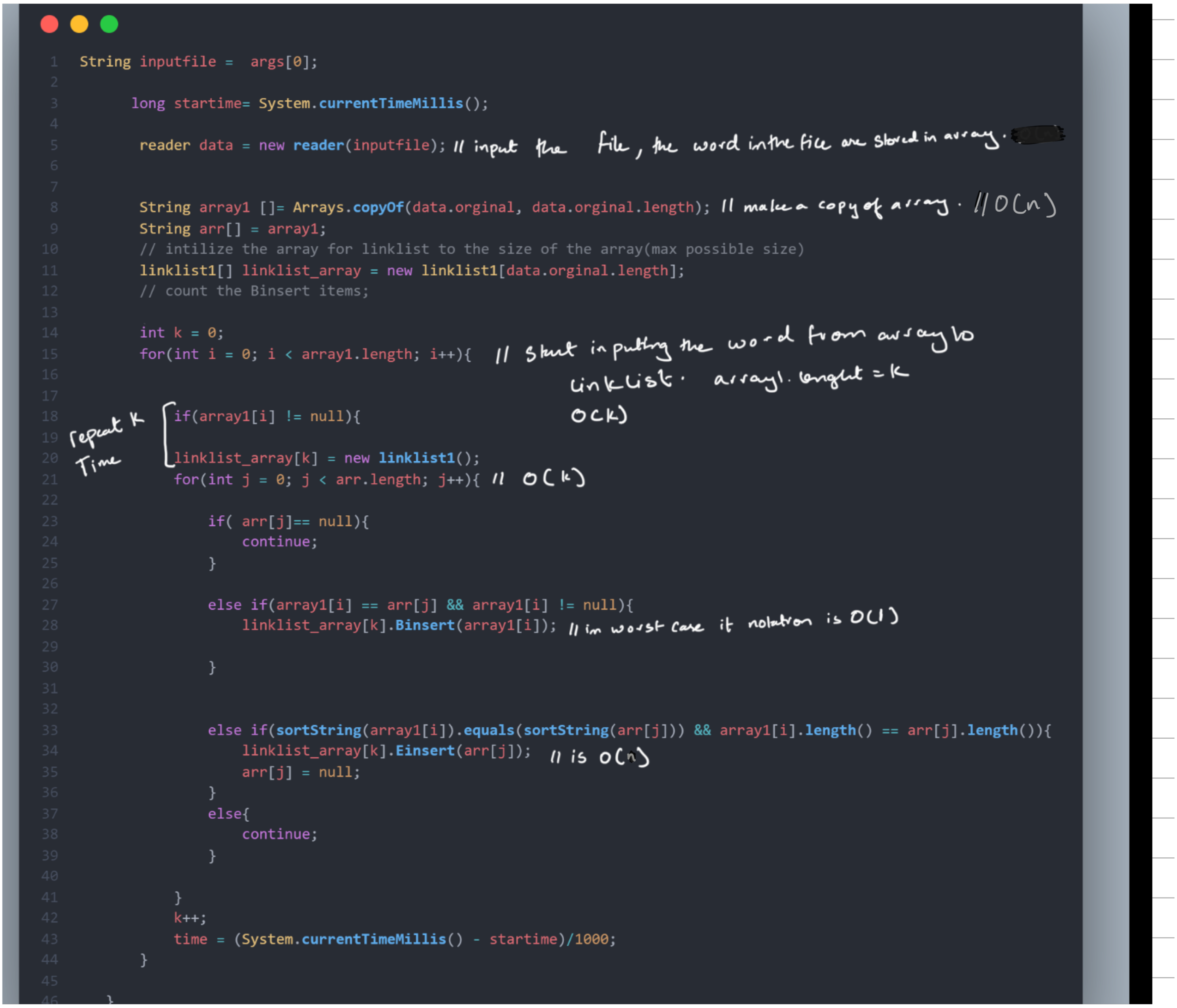
What is the worst-case complexity of your algorithm when checking if two words are anagrams of each other? Express this using big-O notation, and use the variable k to represent the number of letters in each word. Support this with a theoretical analysis of your code.



```
String inputfile =  args[0];

      long startime= System.currentTimeMillis();

       reader data = new reader(inputfile); // input the file, the word in the file are Stored in array. [redacted]


       String array1 []= Arrays.copyOf(data.orginal, data.orginal.length); // make a copy of array. // O(n)
       String arr[] = array1;
       // intilize the array for linklist to the size of the array(max possible size)
       linklist1[] linklist_array = new linklist1[data.orginal.length];
       // count the Binsert items;

       int k = 0;
       for(int i = 0; i < array1.length; i++){  // Start inputting the word from array to
                                                   linklist.  array1.lenght = k
                                                   O(k)
           if(array1[i] != null){
repeat K Time
           linklist_array[k] = new linklist1();
           for(int j = 0; j < arr.length; j++){ // O(k)

               if( arr[j]== null){
                   continue;
               }

               else if(array1[i] == arr[j] && array1[i] != null){
                   linklist_array[k].Binsert(array1[i]); // in worst case it notation is O(1)

               }


               else if(sortString(array1[i]).equals(sortString(arr[j])) && array1[i].length() == arr[j].length()){
                   linklist_array[k].Einsert(arr[j]);  // is O(n)
                   arr[j] = null;
               }
               else{
                   continue;
               }

           }
           k++;
           time = (System.currentTimeMillis() - startime)/1000;
       }

   }
```

So the overall time complexity is $O(k^4)$ for my importing code:

(1) O(k) → the copyof( ) that has a complexity of O(n) { can be remove able)

(2) O(k) → the outer loop that iterates one word at a time.

(3) O(k) → the inner loop that iterate over the list compare with that one word.

(4) Binsert are consider to have O(1) as it just swaping pointer

(5) / Einsert are consider to have O(n) as we need to iterate through the list.

Let N be the number of words in the input word list, and L be the maximum length of any word. What is the big-O running time of your program? Justify your answer using both a theoretical analysis and experimental data (i.e. timing data).