**Report of Data Structure**

Chonbuk National University

Department of Computer Science

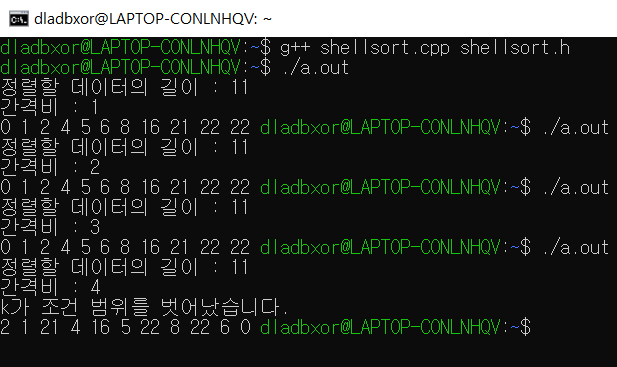
201514768

임유택

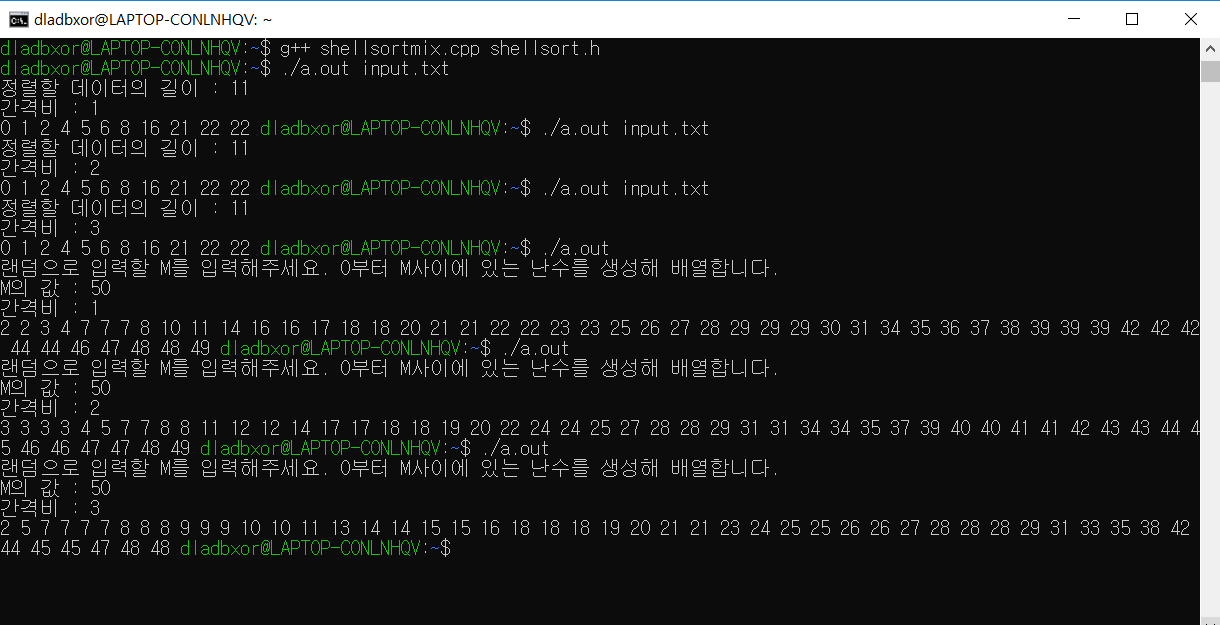
**Problem 1.1**

**1.1.1 shell sort**

**Read the file to sort the numbers**

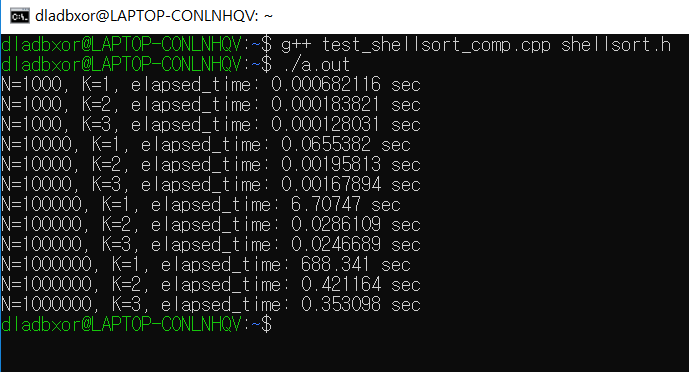


**If a factor of ‘argc’ is present, test from the input file; otherwise, test by generating random data**



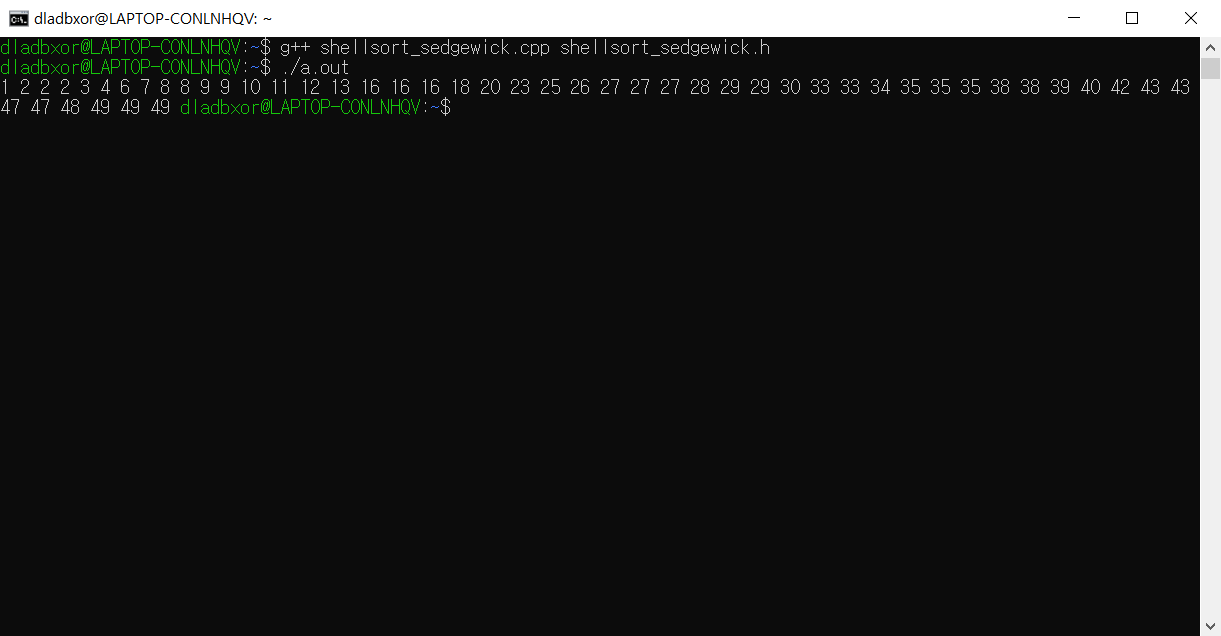
**1.1.2 shell sort run time comparison**

**When the length N of the random array increases to 1000, 10000, 10000, or 1000000, and if K is 1, 2, 3, perform a test comparing the actual time taken to perform ‘shell sorting’ for each**

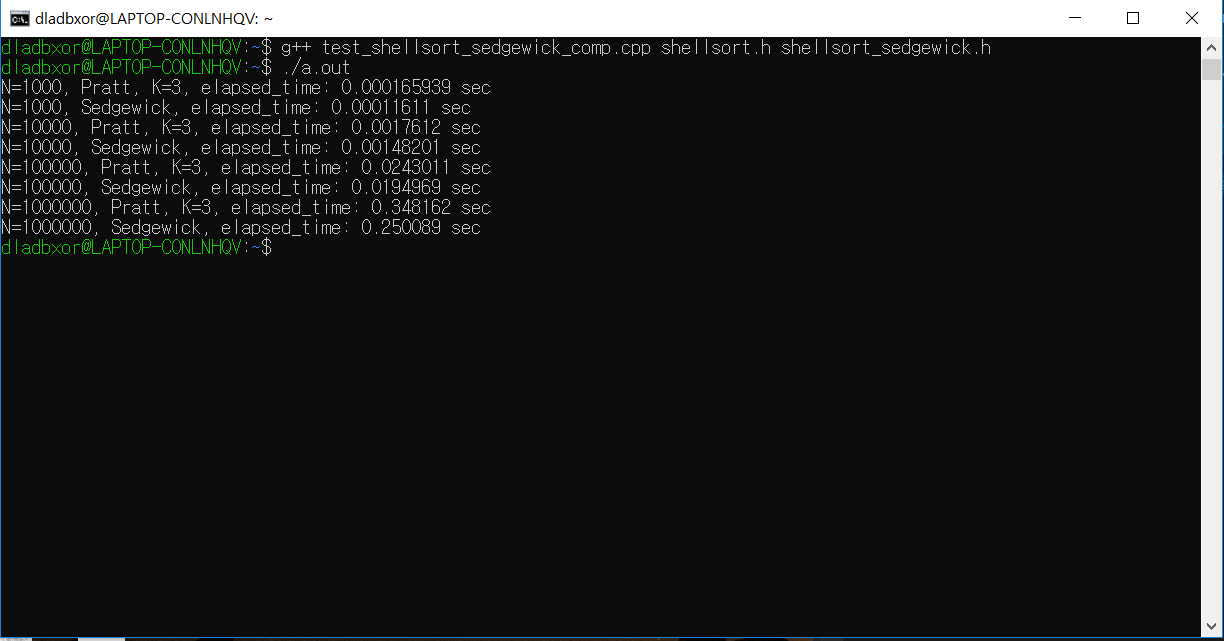


**1.1.3 shell sort : Sedgewick’s method**

**Implement ‘shell sort’ with ‘Gap sequences’ proposed by ‘Sedgewick’**



**When the length N of the random array increases to 1000, 10000, 100000, 1000000, and 1000000, compare the actual time taken to perform 'shell sort; Sedgewick' with 'shell sort' with K 3 above**

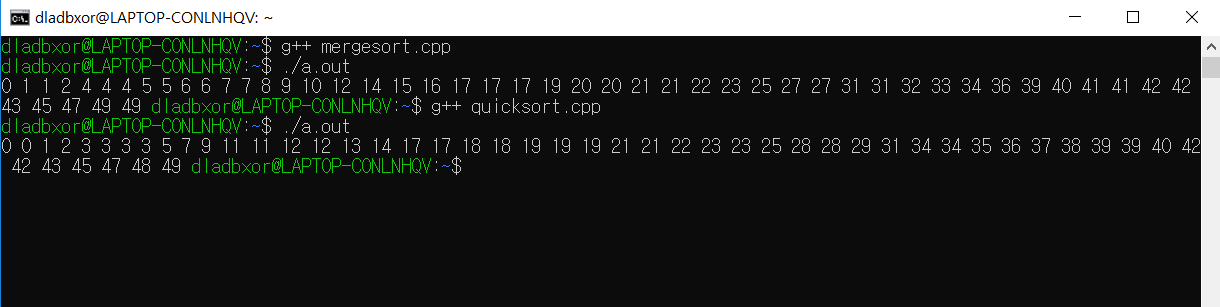


**1.1.4 Computational Complexity of shell sort**

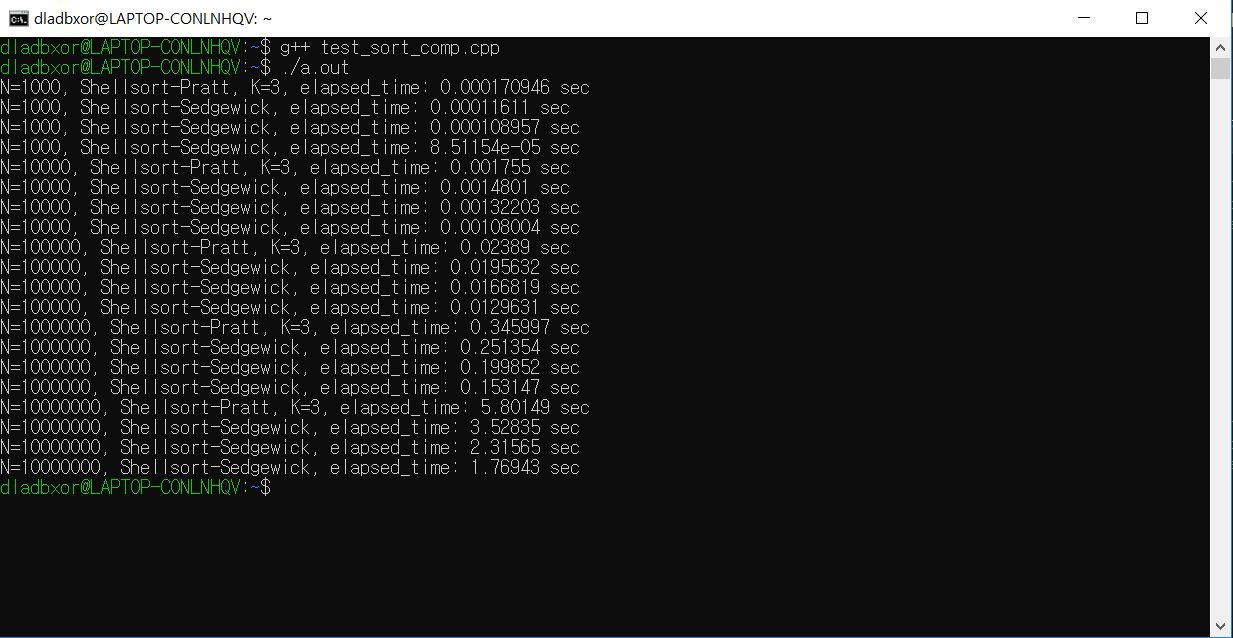
**Pratt shell sort -> 1 4 13 40 121… -> O(N^3/2)**

**Sedgewick shell sort -> 1 8 23 77 281… -> O(N^4/3)**

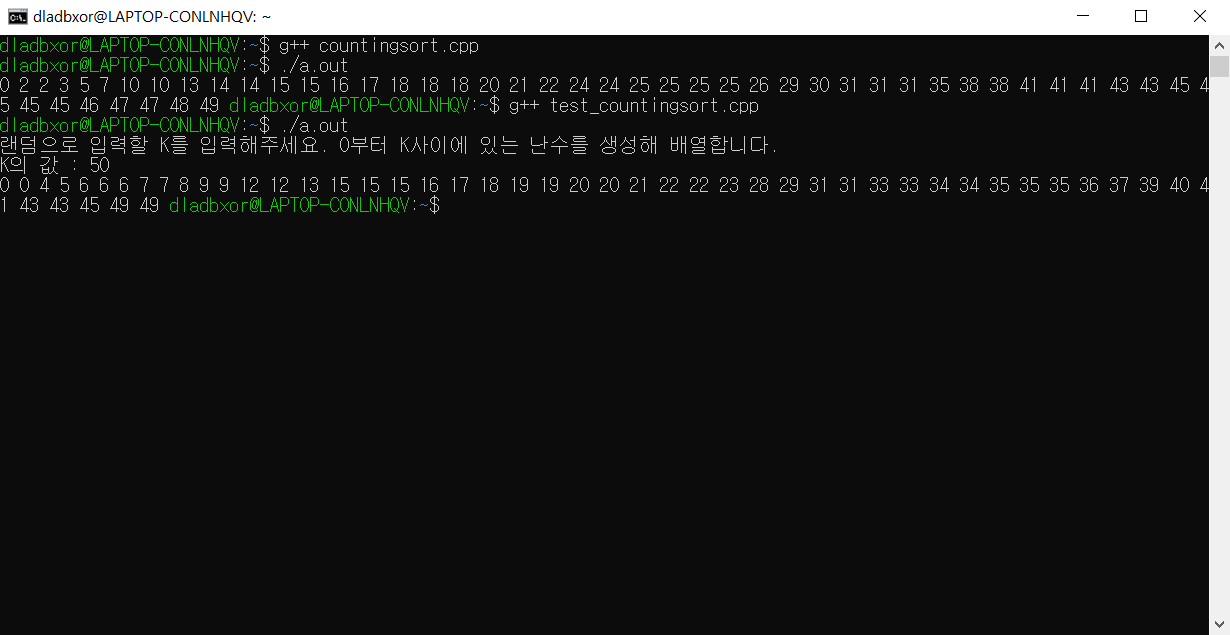
**1.2.1 Implement and test 'multiple sort' and 'quick sort’**

****

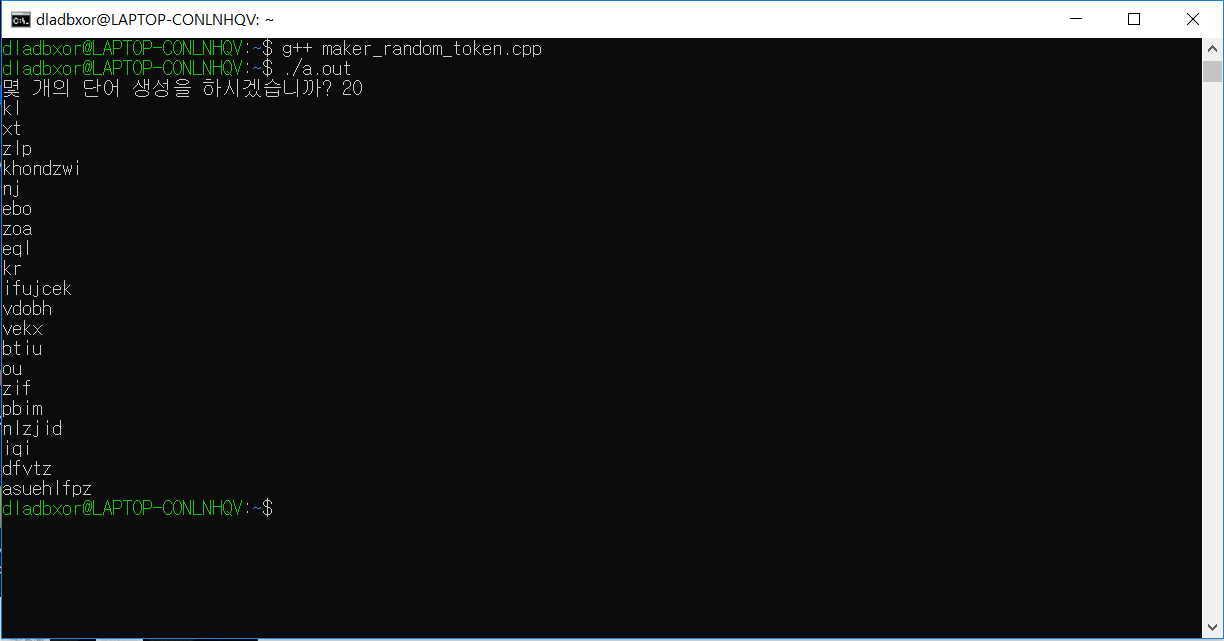
**1.2.2 Compare ‘insertion sort’, ‘merge sort’ and ‘quick sort’**



**1.3.1 Implement ‘counting sort’**



**1.3.2 Implement ‘random token maker’**

****

**1.3.3.1 Implement MSD radix sort**

**In code..**

**1.3.3.2 Implement MSD radix sort (tokens.txt)**

**Randomly create ‘token’ and put it in ‘tokens.txt’ and pull it out again to sort**

****

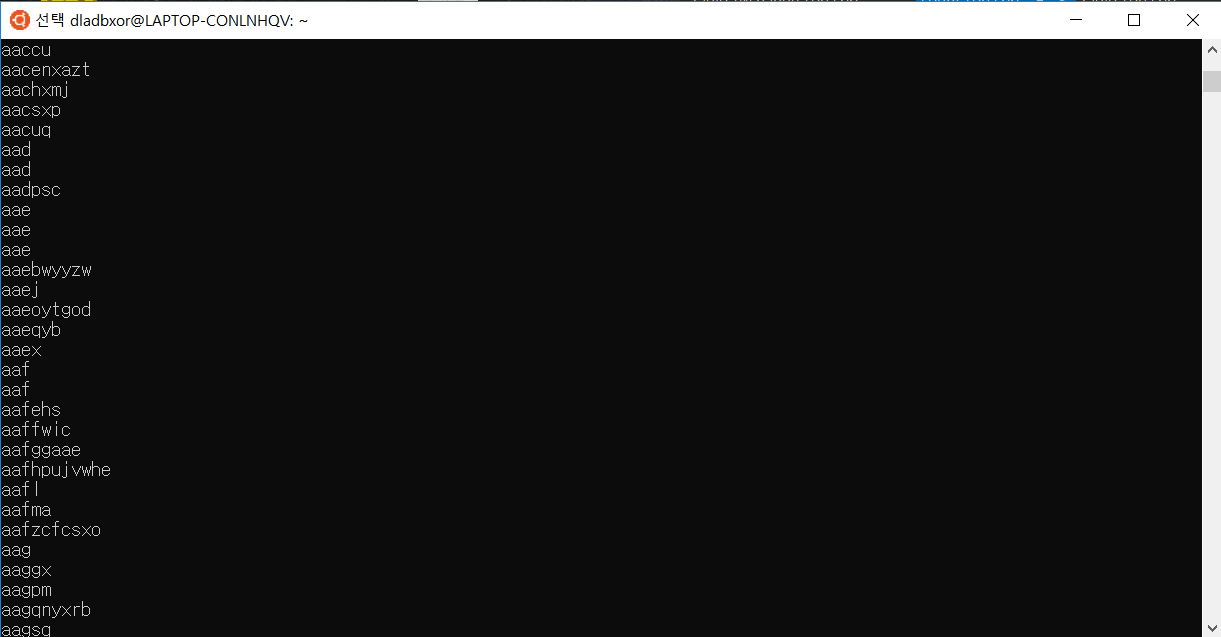
**1.3.3.3 Different settings for ‘N’**

**N = 100000**

****

**.**

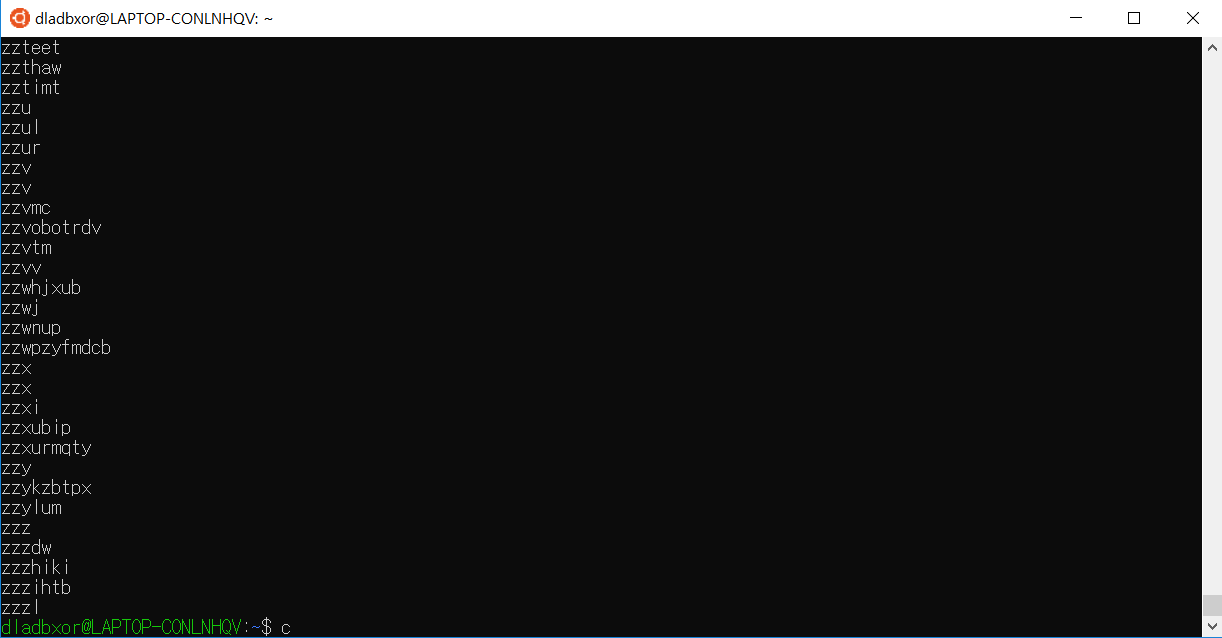
**.**

**.**

**.**

**.**

**.**

****

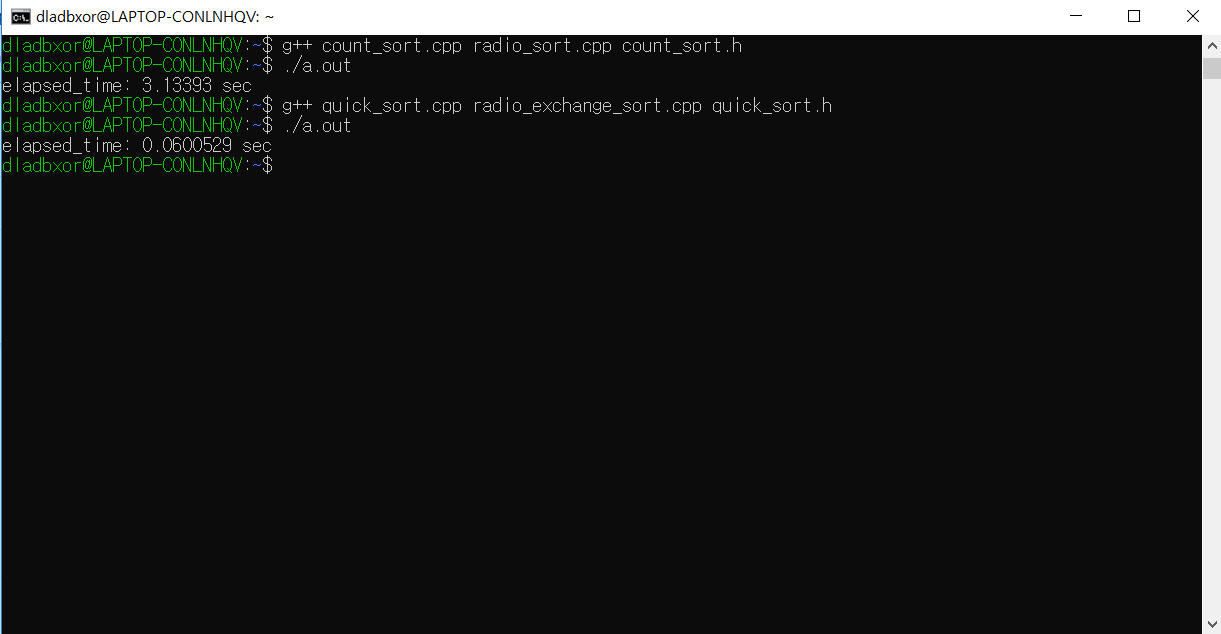
**1.3.4.1 Implement MSD radix exchange sort**

**In code..**

**1.3.4.2 Implement MSD radix exchange sort (tokens.txt)**

****

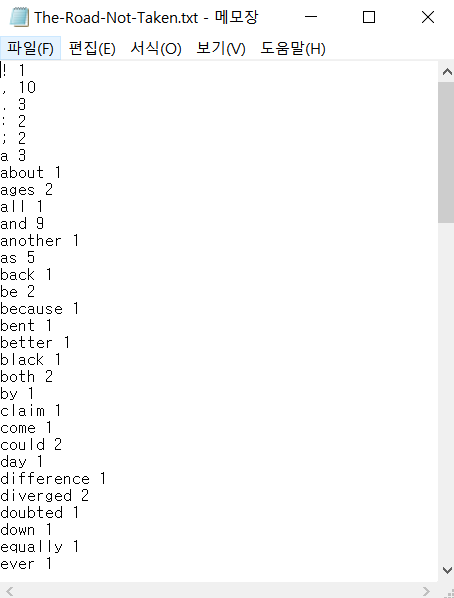
**1.3.4.3 Comparison of ‘MSD radix sort’ and ’MSD radix exchange sort’**

****

**1.4.1 Binary Search Tree : Search, Insert, Delete, Update**

**In code..**

**1.4.2 Calculating the ‘Word Count’ using the Binary Search Tree**

****

**1.4.3 Binary search from the result of ‘Word Count’**

****

****

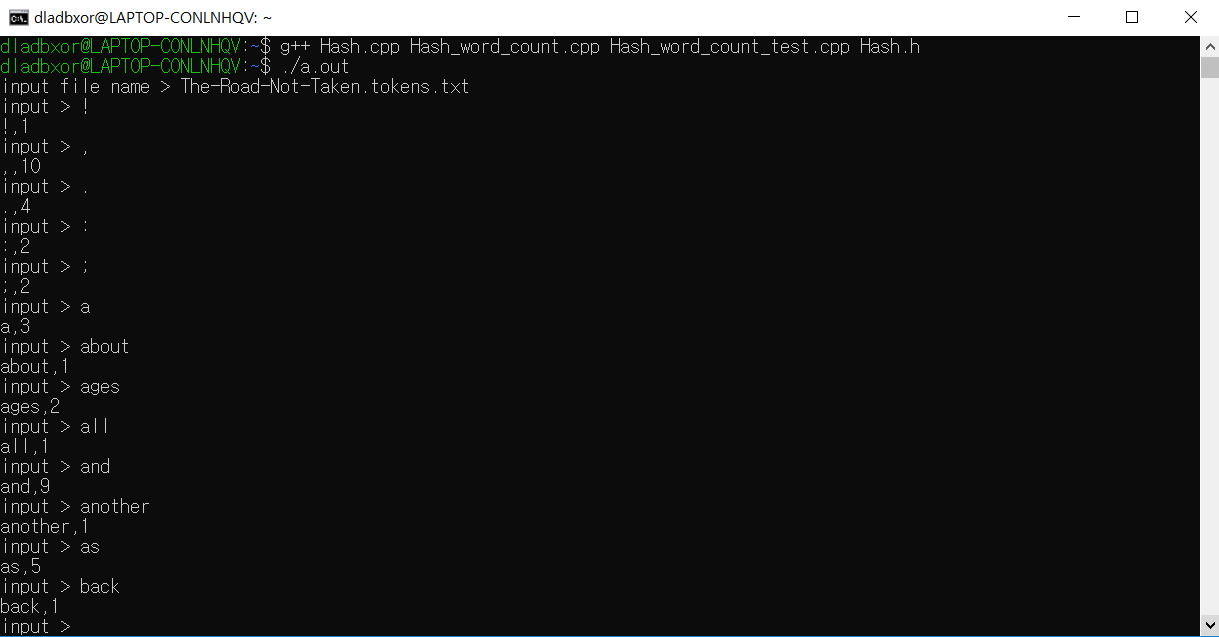
**1.4.4 Binary Search from ‘Word Count’ result : Expansion of large data**

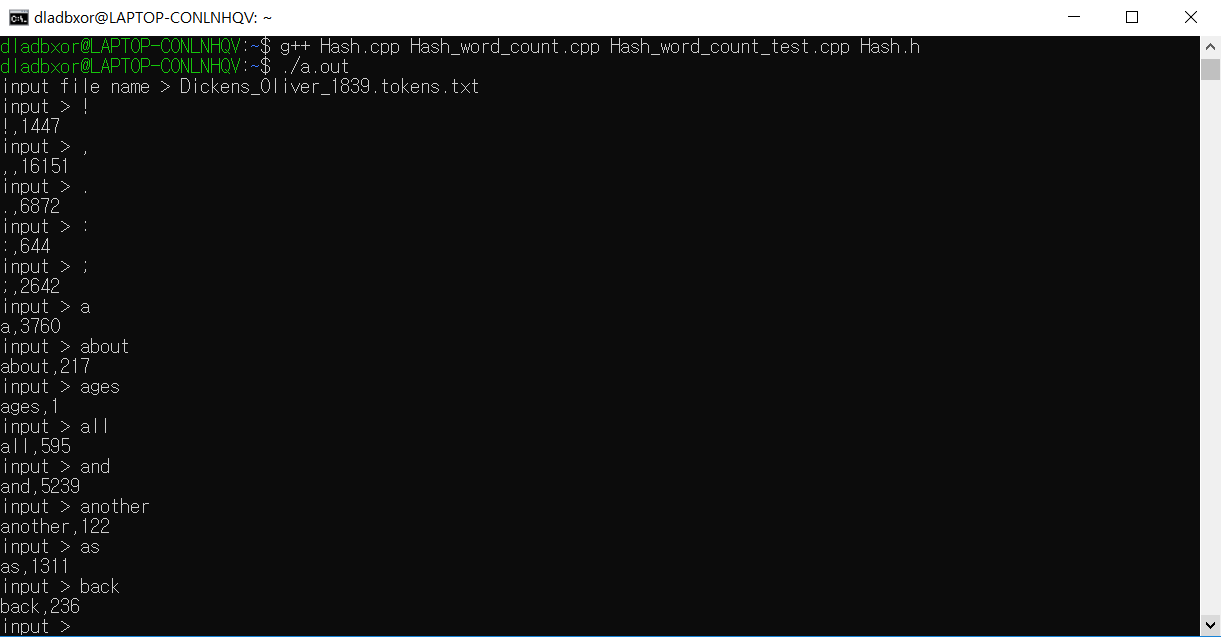
**If there is an infinite amount of data, it is necessary to limit the scope of the search. There are 26 alphabets from 'a to z'. After creating 26 arrays, you can place 'count' in each room according to the first syllable of the word and limit the range to words starting with a, b, etc. For example, If I want to find the word 'but', it is possible to limit the range of words in the total array by adding the count in the room of the word 'b' that starts with 'a'**

**1.5.1 Implement Hash Class**

**In code..**

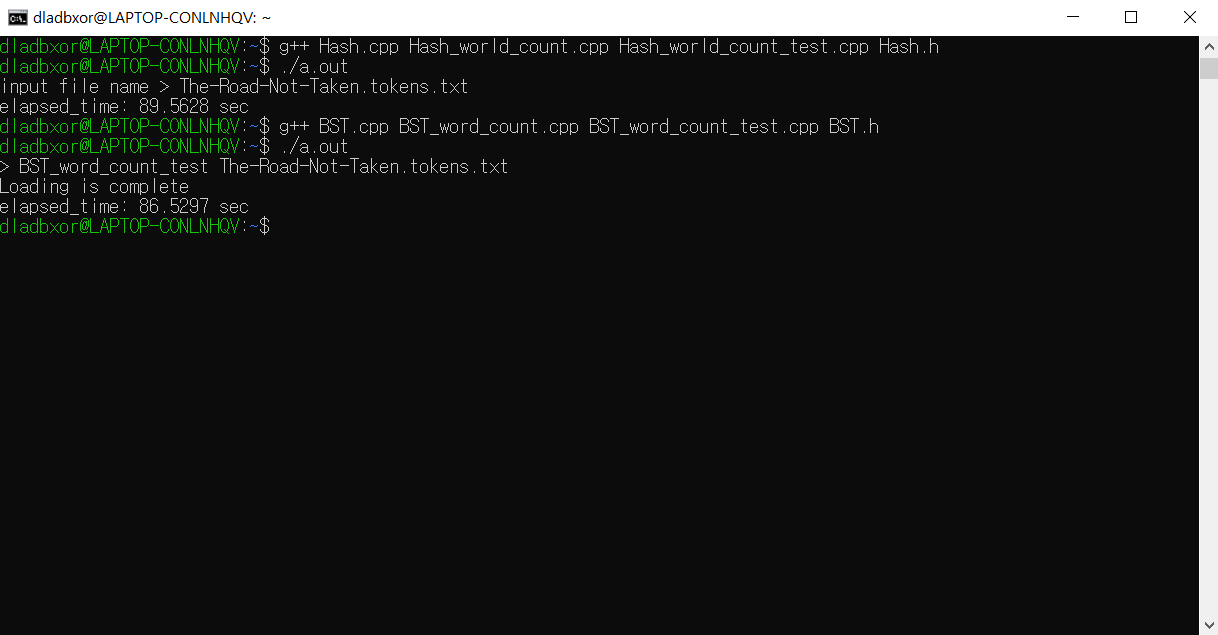
**1.5.2 Calculate and test ‘Word Count’ using a Hash**

****

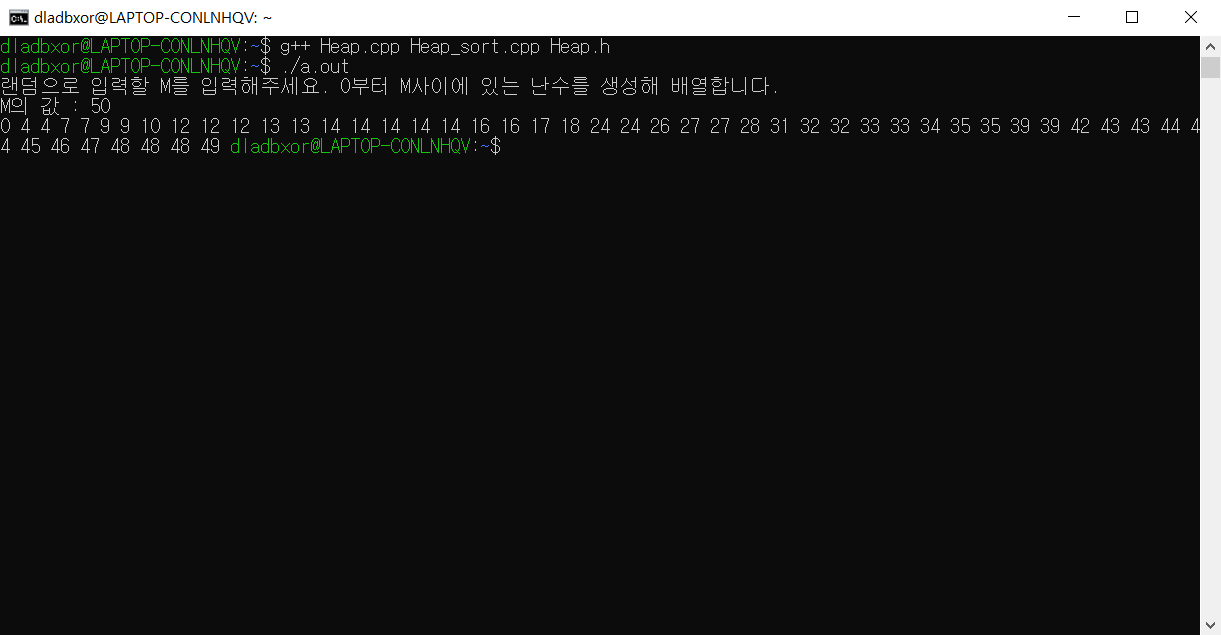
****

**1.5.3 Compare ‘Hash’ and ‘Binary Search Tree’**

**We made ‘random token maker’ in 1.3.2. Use ‘token.txt’ with a ‘random token’ to compare ‘Binary Search tree’ and ‘Hash’**

****

**1.6.1 ~ 1.6.2 Implement ‘bulid heap’ and ‘remove heap’**

****