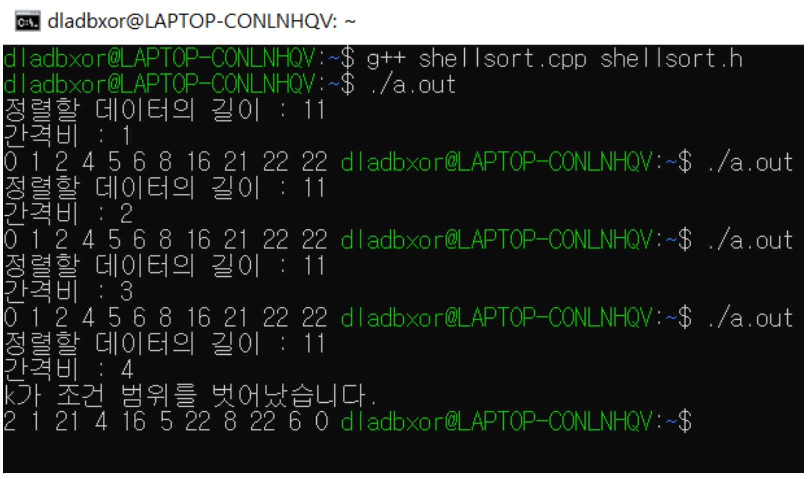
Report of Data Structure

Chonbuk National University
Department of Computer Science
201514768
임유택

Problem 1.1

1.1.1 shell sort

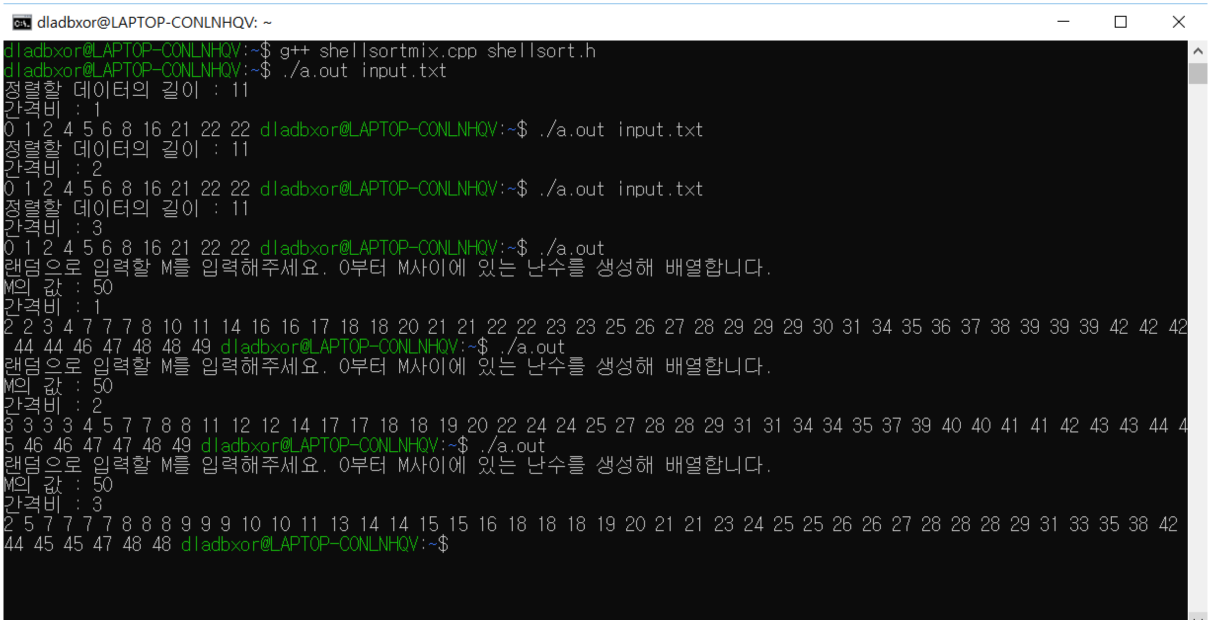
Read the file to sort the numbers



dladbxor@LAPTOP-CONLNHQV: ~

```
dladbxor@LAPTOP-CONLNHQV:~$ g++ shellsort.cpp shellsort.h
dladbxor@LAPTOP-CONLNHQV:~$ ./a.out
정렬할 데이터의 길이 : 11
간격비 : 1
0 1 2 4 5 6 8 16 21 22 22 dladbxor@LAPTOP-CONLNHQV:~$ ./a.out
정렬할 데이터의 길이 : 11
간격비 : 2
0 1 2 4 5 6 8 16 21 22 22 dladbxor@LAPTOP-CONLNHQV:~$ ./a.out
정렬할 데이터의 길이 : 11
간격비 : 3
0 1 2 4 5 6 8 16 21 22 22 dladbxor@LAPTOP-CONLNHQV:~$ ./a.out
정렬할 데이터의 길이 : 11
간격비 : 4
k가 조건 범위를 벗어났습니다.
2 1 21 4 16 5 22 8 22 6 0 dladbxor@LAPTOP-CONLNHQV:~$
```

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

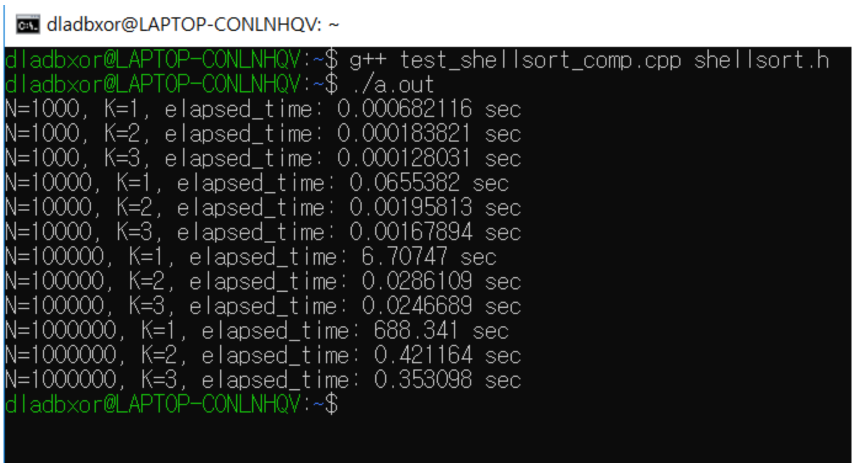


dladbxor@LAPTOP-CONLNHQV: ~

```
dladbxor@LAPTOP-CONLNHQV:~$ g++ shellsortmix.cpp shellsort.h
dladbxor@LAPTOP-CONLNHQV:~$ ./a.out input.txt
정렬할 데이터의 길이 : 11
간격비 : 1
0 1 2 4 5 6 8 16 21 22 22 dladbxor@LAPTOP-CONLNHQV:~$ ./a.out input.txt
정렬할 데이터의 길이 : 11
간격비 : 2
0 1 2 4 5 6 8 16 21 22 22 dladbxor@LAPTOP-CONLNHQV:~$ ./a.out input.txt
정렬할 데이터의 길이 : 11
간격비 : 3
0 1 2 4 5 6 8 16 21 22 22 dladbxor@LAPTOP-CONLNHQV:~$ ./a.out
랜덤으로 입력할 M를 입력해주세요. 0부터 M사이에 있는 난수를 생성해 배열합니다.
M의 값 : 50
간격비 : 1
2 2 3 4 7 7 7 8 10 11 14 16 16 17 18 18 20 21 21 22 22 23 23 25 26 27 28 29 29 29 30 31 34 35 36 37 38 39 39 39 42 42 42
 44 44 46 47 48 48 49 dladbxor@LAPTOP-CONLNHQV:~$ ./a.out
랜덤으로 입력할 M를 입력해주세요. 0부터 M사이에 있는 난수를 생성해 배열합니다.
M의 값 : 50
간격비 : 2
3 3 3 3 4 5 7 7 8 8 11 12 12 14 17 17 18 18 19 20 22 24 24 25 27 28 28 29 31 31 34 34 35 37 39 40 40 41 41 42 43 43 44 4
5 46 46 47 47 48 49 dladbxor@LAPTOP-CONLNHQV:~$ ./a.out
랜덤으로 입력할 M를 입력해주세요. 0부터 M사이에 있는 난수를 생성해 배열합니다.
M의 값 : 50
간격비 : 3
2 5 7 7 7 7 8 8 8 9 9 9 10 10 11 13 14 14 15 15 16 18 18 18 19 20 21 21 23 24 25 25 26 26 27 28 28 28 29 31 33 35 38 42
44 45 45 47 48 48 dladbxor@LAPTOP-CONLNHQV:~$
```

1.1.2 shell sort run time comparison

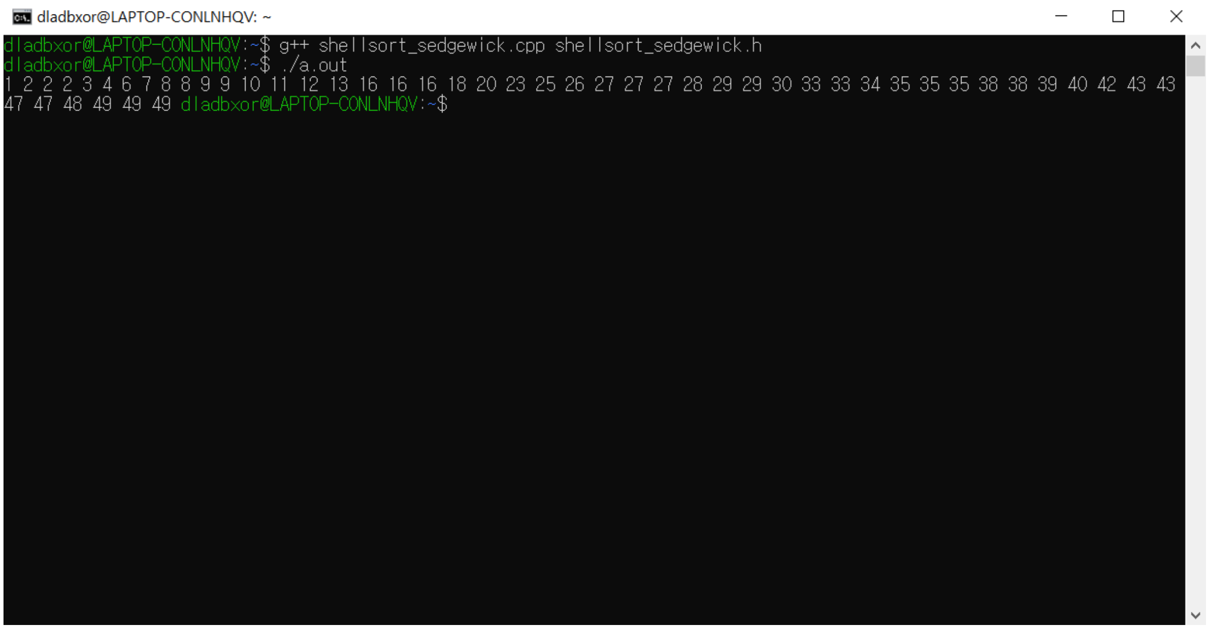
When the length N of the random array increases to 1000, 10000, 10000, or 1000000, and if K is 1, 2, 3, perform a test comparing the actual time taken to perform 'shell sorting' for each



```
dladbxor@LAPTOP-CONLNHQV: ~
dladbxor@LAPTOP-CONLNHQV:~$ g++ test_shellsort_comp.cpp shellsort.h
dladbxor@LAPTOP-CONLNHQV:~$ ./a.out
N=1000, K=1, elapsed_time: 0.000682116 sec
N=1000, K=2, elapsed_time: 0.000183821 sec
N=1000, K=3, elapsed_time: 0.000128031 sec
N=10000, K=1, elapsed_time: 0.0655382 sec
N=10000, K=2, elapsed_time: 0.00195813 sec
N=10000, K=3, elapsed_time: 0.00167894 sec
N=100000, K=1, elapsed_time: 6.70747 sec
N=100000, K=2, elapsed_time: 0.0286109 sec
N=100000, K=3, elapsed_time: 0.0246689 sec
N=1000000, K=1, elapsed_time: 688.341 sec
N=1000000, K=2, elapsed_time: 0.421164 sec
N=1000000, K=3, elapsed_time: 0.353098 sec
dladbxor@LAPTOP-CONLNHQV:~$
```

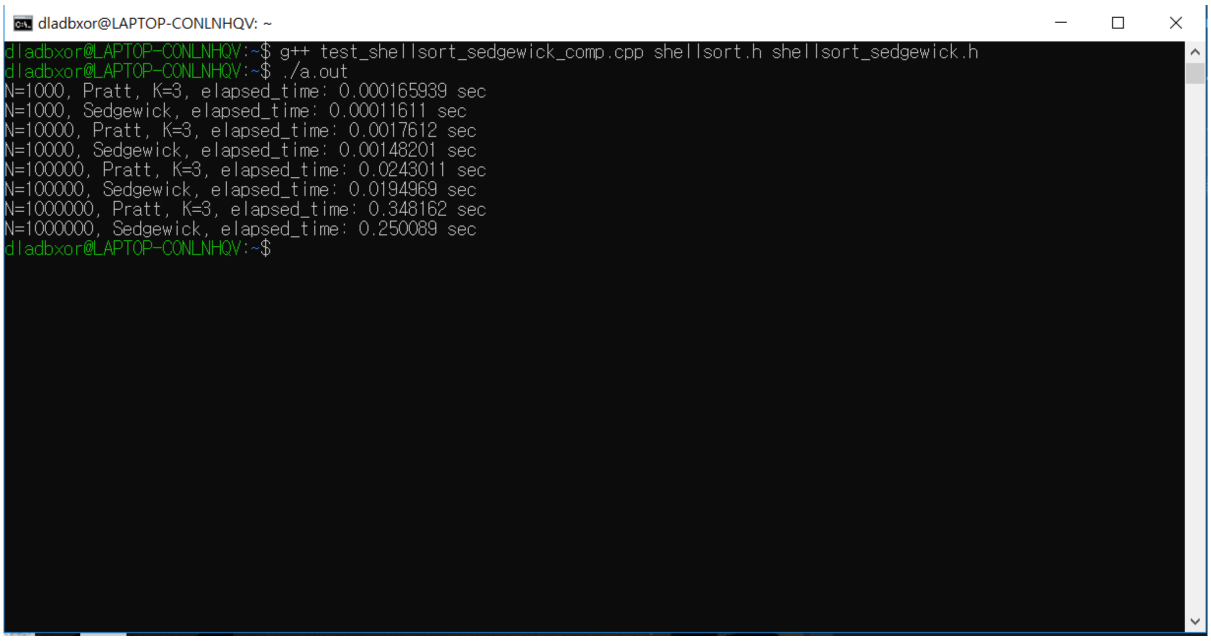
1.1.3 shell sort : Sedgewick's method

Implement 'shell sort' with 'Gap sequences' proposed by 'Sedgewick'



```
dladbxor@LAPTOP-CONLNHQV: ~
dladbxor@LAPTOP-CONLNHQV:~$ g++ shellsort_sedgewick.cpp shellsort_sedgewick.h
dladbxor@LAPTOP-CONLNHQV:~$ ./a.out
1 2 2 2 3 4 6 7 8 8 9 9 10 11 12 13 16 16 16 18 20 23 25 26 27 27 27 28 29 29 30 33 33 34 35 35 35 38 38 39 40 42 43 43
47 47 48 49 49 49 dladbxor@LAPTOP-CONLNHQV:~$
```

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



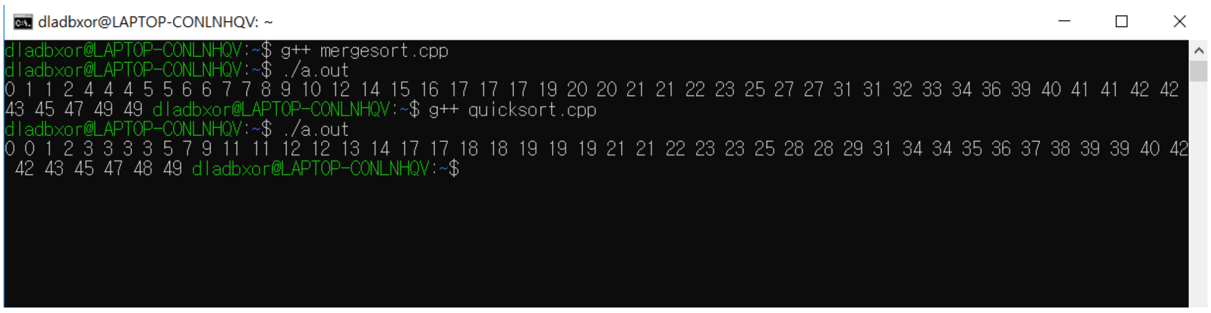
```
dladbxor@LAPTOP-CONLNHQV:~$ g++ test_shellsort_sedgewick_comp.cpp shellsort.h shellsort_sedgewick.h
dladbxor@LAPTOP-CONLNHQV:~$ ./a.out
N=1000, Pratt, K=3, elapsed_time: 0.000165939 sec
N=1000, Sedgewick, elapsed_time: 0.00011611 sec
N=10000, Pratt, K=3, elapsed_time: 0.0017612 sec
N=10000, Sedgewick, elapsed_time: 0.00148201 sec
N=100000, Pratt, K=3, elapsed_time: 0.0243011 sec
N=100000, Sedgewick, elapsed_time: 0.0194969 sec
N=1000000, Pratt, K=3, elapsed_time: 0.348162 sec
N=1000000, Sedgewick, elapsed_time: 0.250089 sec
dladbxor@LAPTOP-CONLNHQV:~$
```

1.1.4 Computational Complexity of shell sort

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

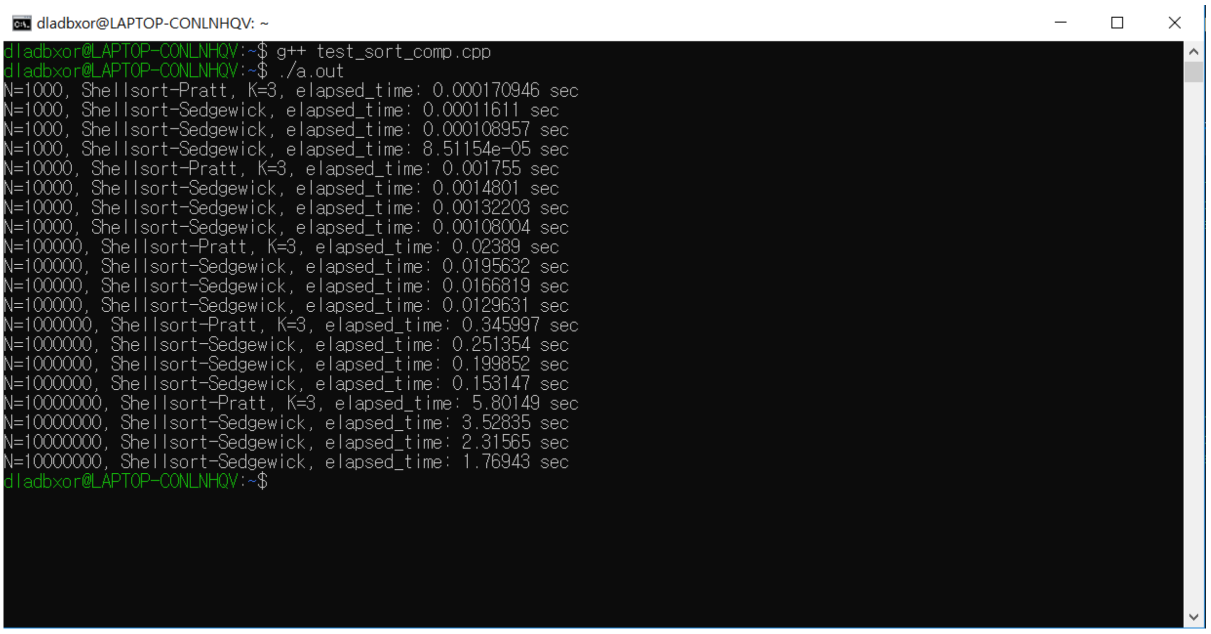
Sedgewick shell sort -> 1 8 23 77 281... -> O(N^4/3)

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



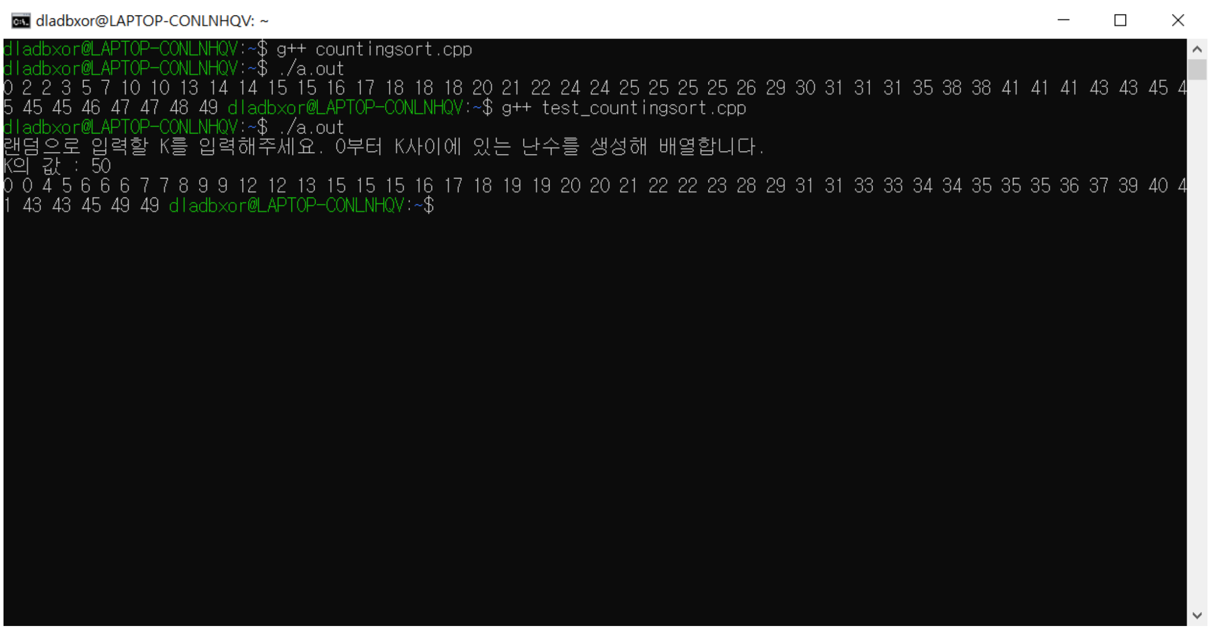
```
dladbxor@LAPTOP-CONLNHQV:~$ g++ mergesort.cpp
dladbxor@LAPTOP-CONLNHQV:~$ ./a.out
0 1 1 2 4 4 4 5 5 6 6 7 7 8 9 10 12 14 15 16 17 17 17 19 20 20 21 21 22 23 25 27 27 31 31 32 33 34 36 39 40 41 41 42 42
43 45 47 49 49 dladbxor@LAPTOP-CONLNHQV:~$ g++ quicksort.cpp
dladbxor@LAPTOP-CONLNHQV:~$ ./a.out
0 0 1 2 3 3 3 3 5 7 9 11 11 12 12 13 14 17 17 18 18 19 19 19 21 21 22 23 23 25 28 28 29 31 34 34 35 36 37 38 39 39 40 42
 42 43 45 47 48 49 dladbxor@LAPTOP-CONLNHQV:~$
```

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



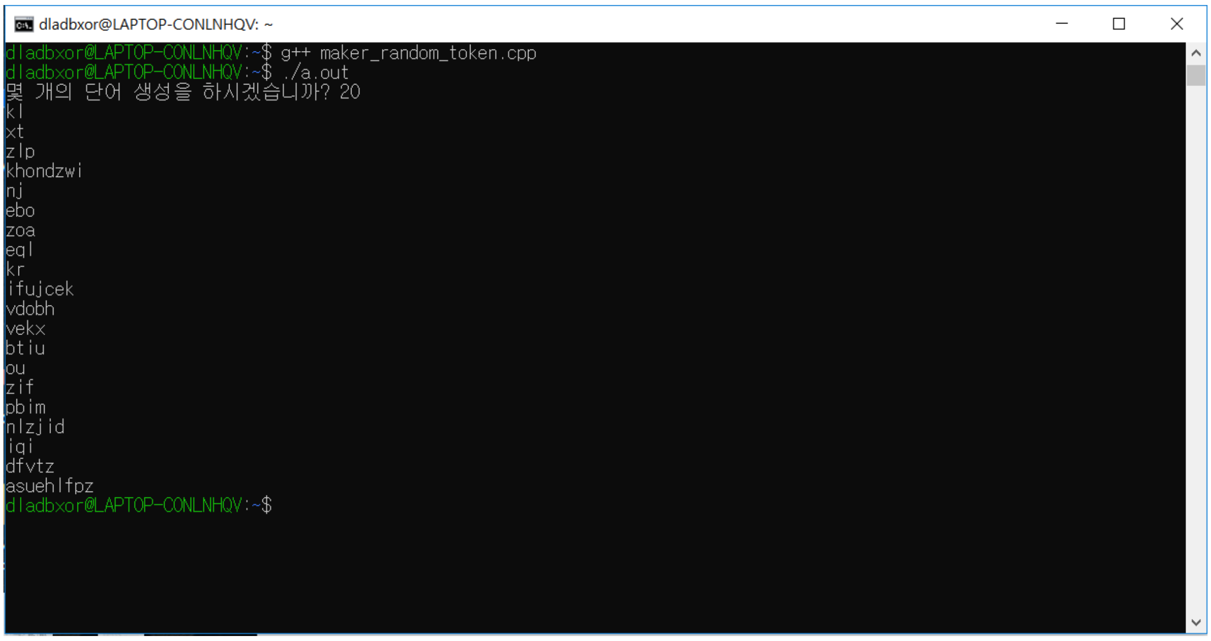
```
dladbxor@LAPTOP-CONLNHQV:~$ g++ test_sort_comp.cpp
dladbxor@LAPTOP-CONLNHQV:~$ ./a.out
N=1000, Shellsort-Pratt, K=3, elapsed_time: 0.000170946 sec
N=1000, Shellsort-Sedgewick, elapsed_time: 0.00011611 sec
N=1000, Shellsort-Sedgewick, elapsed_time: 0.000108957 sec
N=1000, Shellsort-Sedgewick, elapsed_time: 8.51154e-05 sec
N=10000, Shellsort-Pratt, K=3, elapsed_time: 0.001755 sec
N=10000, Shellsort-Sedgewick, elapsed_time: 0.0014801 sec
N=10000, Shellsort-Sedgewick, elapsed_time: 0.00132203 sec
N=10000, Shellsort-Sedgewick, elapsed_time: 0.00108004 sec
N=100000, Shellsort-Pratt, K=3, elapsed_time: 0.02389 sec
N=100000, Shellsort-Sedgewick, elapsed_time: 0.0195632 sec
N=100000, Shellsort-Sedgewick, elapsed_time: 0.0166819 sec
N=100000, Shellsort-Sedgewick, elapsed_time: 0.0129631 sec
N=1000000, Shellsort-Pratt, K=3, elapsed_time: 0.345997 sec
N=1000000, Shellsort-Sedgewick, elapsed_time: 0.251354 sec
N=1000000, Shellsort-Sedgewick, elapsed_time: 0.199852 sec
N=1000000, Shellsort-Sedgewick, elapsed_time: 0.153147 sec
N=10000000, Shellsort-Pratt, K=3, elapsed_time: 5.80149 sec
N=10000000, Shellsort-Sedgewick, elapsed_time: 3.52835 sec
N=10000000, Shellsort-Sedgewick, elapsed_time: 2.31565 sec
N=10000000, Shellsort-Sedgewick, elapsed_time: 1.76943 sec
dladbxor@LAPTOP-CONLNHQV:~$
```

1.3.1 Implement ‘counting sort’



```
dladbxor@LAPTOP-CONLNHQV:~$ g++ countingsort.cpp
dladbxor@LAPTOP-CONLNHQV:~$ ./a.out
0 2 2 3 5 7 10 10 13 14 14 15 15 16 17 18 18 18 20 21 22 24 24 25 25 25 25 26 29 30 31 31 31 35 38 38 41 41 41 43 43 45 4
5 45 45 46 47 47 48 49 dladbxor@LAPTOP-CONLNHQV:~$ g++ test_countingsort.cpp
dladbxor@LAPTOP-CONLNHQV:~$ ./a.out
랜덤으로 입력할 K를 입력해주세요. 0부터 K사이에 있는 난수를 생성해 배열합니다.
K의 값 : 50
0 0 4 5 6 6 6 7 7 8 9 9 12 12 13 15 15 15 16 17 18 19 19 20 20 21 22 22 23 28 29 31 31 33 33 34 34 35 35 35 36 37 39 40 4
1 43 43 45 49 49 dladbxor@LAPTOP-CONLNHQV:~$
```

1.3.2 Implement 'random token maker'



dladbxor@LAPTOP-CONLNHQV: ~

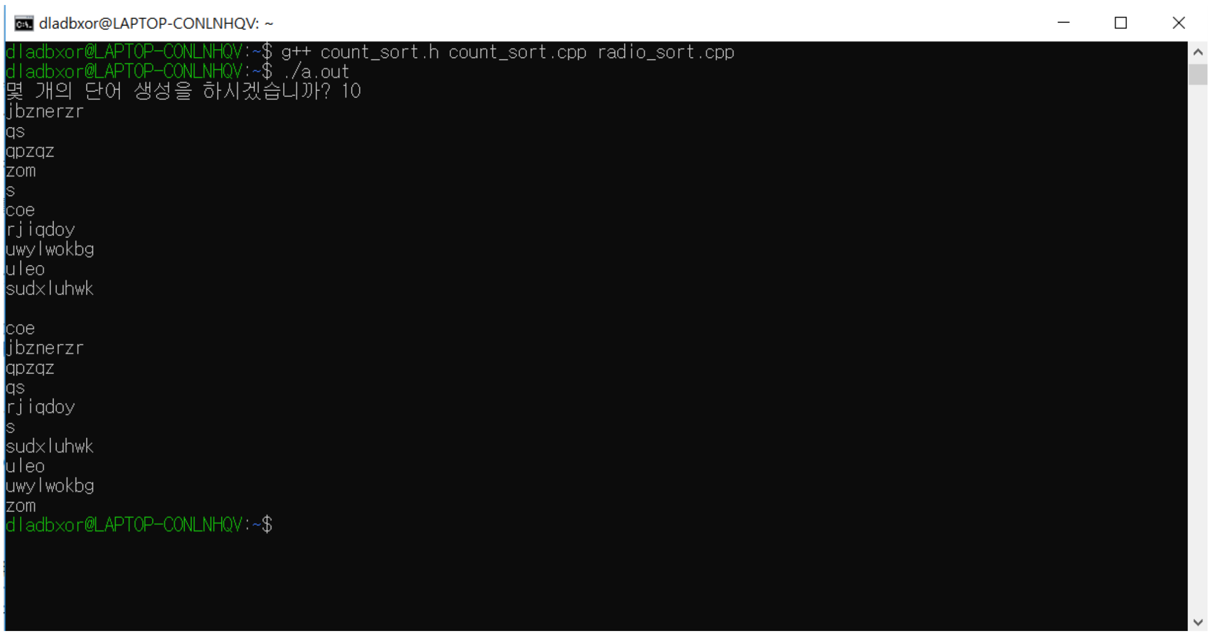
```
dladbxor@LAPTOP-CONLNHQV:~$ g++ maker_random_token.cpp
dladbxor@LAPTOP-CONLNHQV:~$ ./a.out
몇 개의 단어 생성을 하시겠습니까? 20
kl
xt
zlp
khondzwi
nj
ebo
zoa
eql
kr
ifujcek
vdobh
vekx
btiu
ou
zif
pbim
nlzjid
iqi
dfvtz
asuehlfpz
dladbxor@LAPTOP-CONLNHQV:~$
```

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



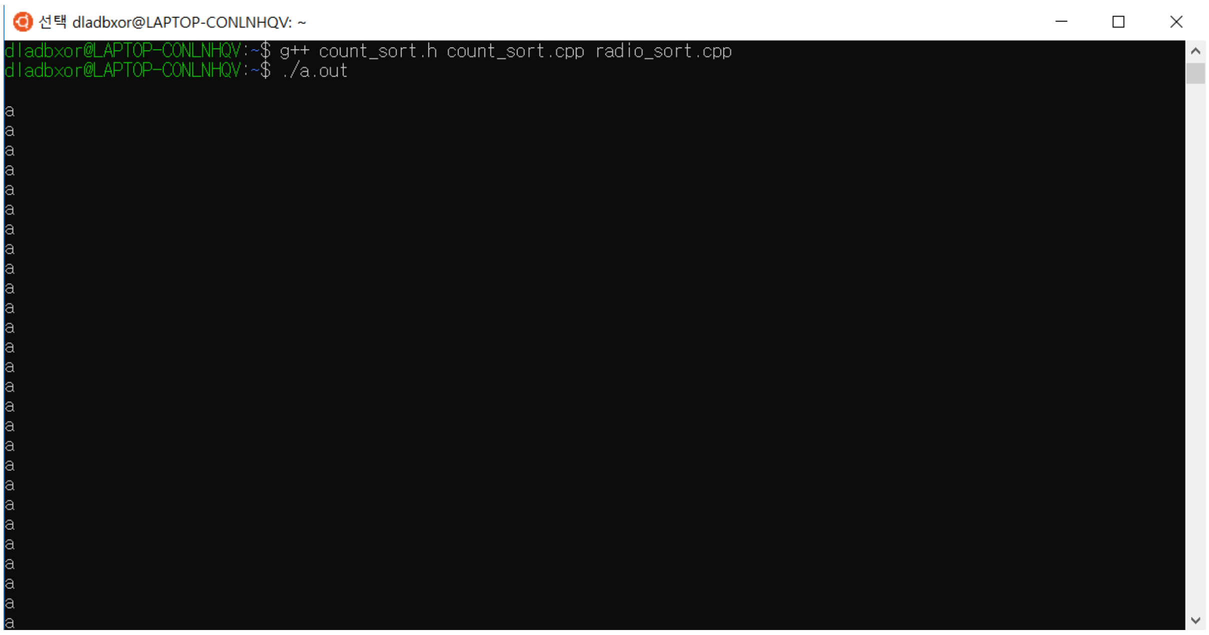
dladbxor@LAPTOP-CONLNHQV: ~

```
dladbxor@LAPTOP-CONLNHQV:~$ g++ count_sort.h count_sort.cpp radio_sort.cpp
dladbxor@LAPTOP-CONLNHQV:~$ ./a.out
몇 개의 단어 생성을 하시겠습니까? 10
jbznerzr
qs
qpzqz
zom
s
coe
rjiqdoy
uwylwokbg
uleo
sudxluhwk

coe
jbznerzr
qpzqz
qs
rjiqdoy
s
sudxluhwk
uleo
uwylwokbg
zom
dladbxor@LAPTOP-CONLNHQV:~$
```

1.3.3.3 Different settings for ‘N’

N = 100000



선택 dladbxor@LAPTOP-CONLNHQV: ~

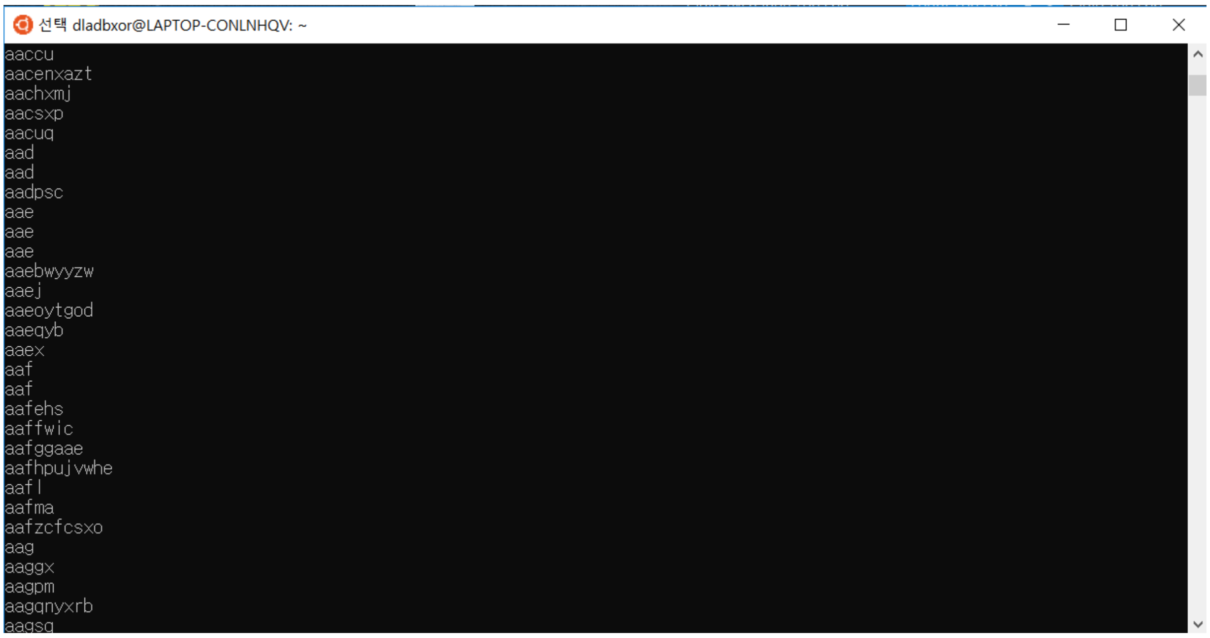
```
dladbxor@LAPTOP-CONLNHQV:~$ g++ count_sort.h count_sort.cpp radio_sort.cpp
dladbxor@LAPTOP-CONLNHQV:~$ ./a.out

a
a
a
a
a
a
a
a
a
a
a
a
a
a
a
a
a
a
a
a
a
a
a
a
a
a
a
```

.

.

.



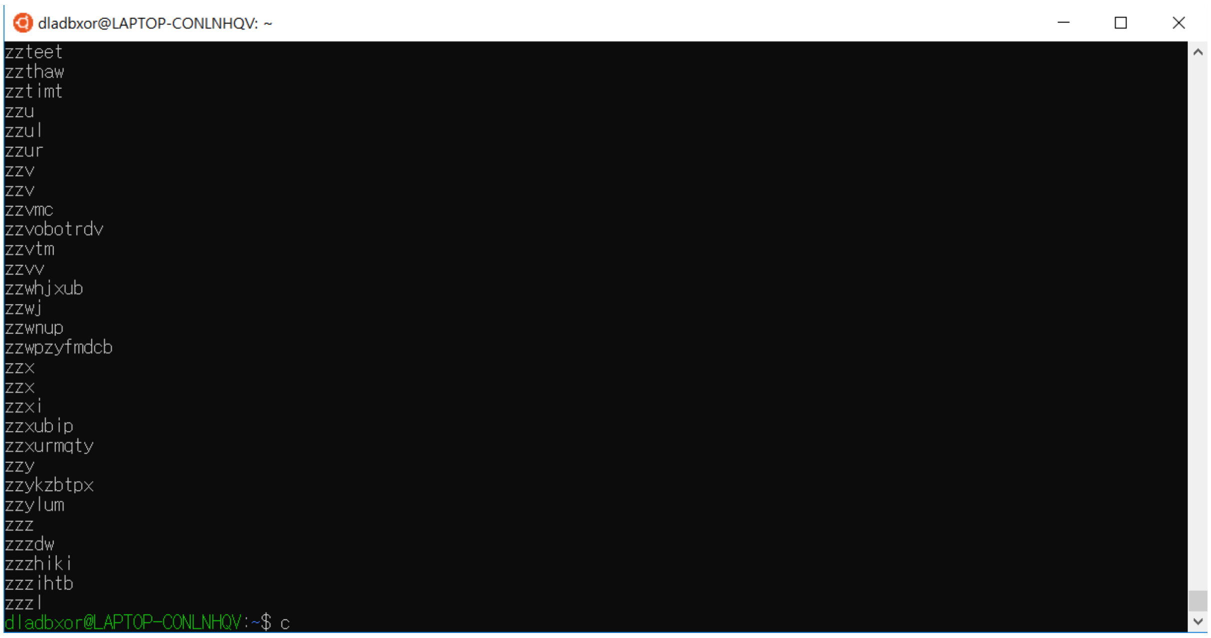
선택 dladbxor@LAPTOP-CONLNHQV: ~

```
aaccu
aacenxazt
aachxmj
aacsxp
aacuq
aad
aad
aadpsc
aae
aae
aae
aaebwyyzw
aaej
aaeoytgod
aaeqyb
aaex
aaf
aaf
aafehs
aaffwic
aafggaae
aafhpujvwhe
aafl
aafma
aafzcfcsxo
aag
aaggx
aagpm
aagqnyxrb
aagsq
```

.

.

.

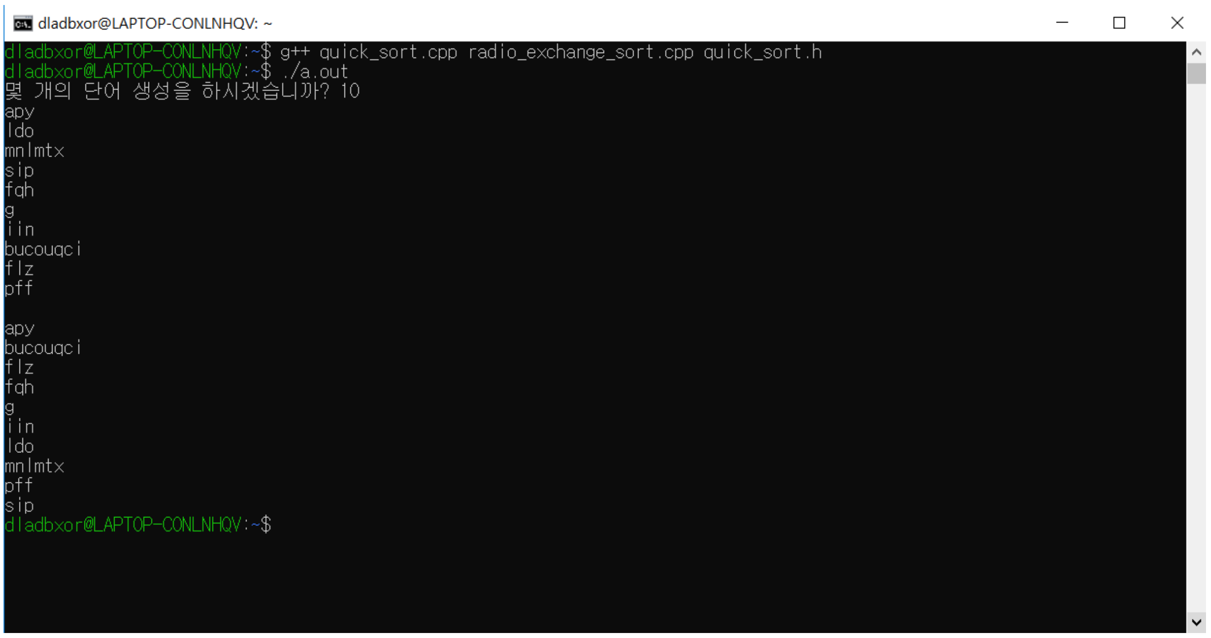


```
dladbxor@LAPTOP-CONLNHQV: ~
zzteet
zzthaw
zztimt
zzu
zzul
zzur
zzv
zzv
zzvmc
zzvobotrdv
zzvtm
zzvv
zzwhjxub
zzwj
zzwnup
zzwpzyfmdcb
zzx
zzx
zzxi
zzxubip
zzxurmqty
zzy
zzykzbtpx
zzylum
zzz
zzzdw
zzzhiki
zzzihtb
zzzl
dladbxor@LAPTOP-CONLNHQV:~$ c
```

1.3.4.1 Implement MSD radix exchange sort

In code..

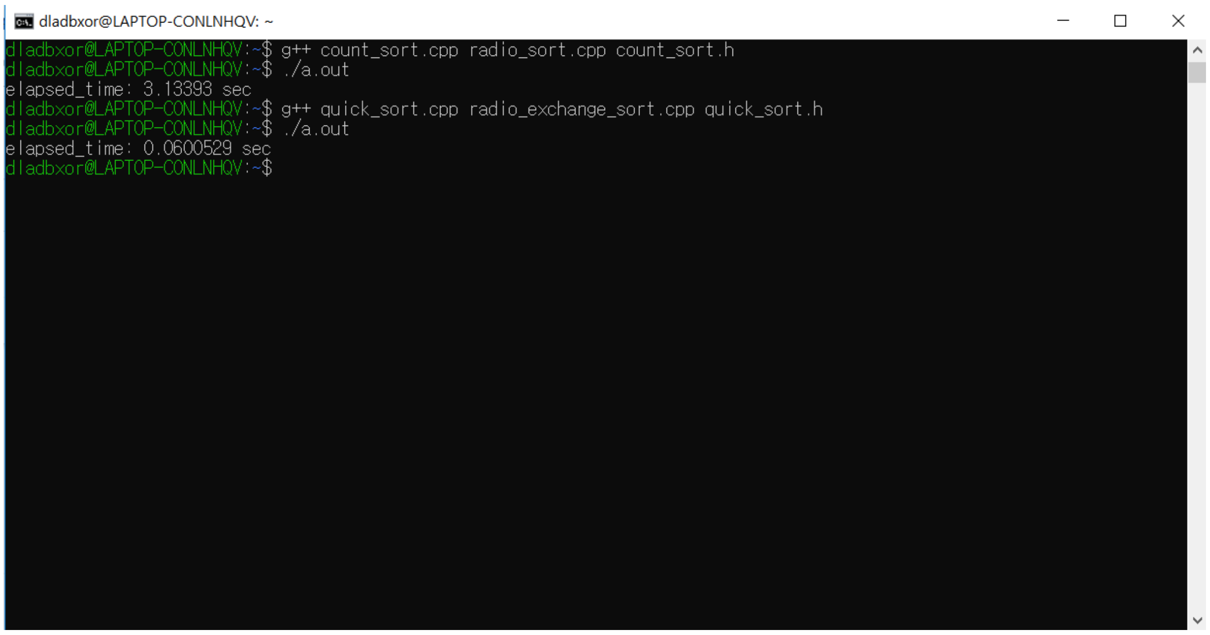
1.3.4.2 Implement MSD radix exchange sort (tokens.txt)



```
dladbxor@LAPTOP-CONLNHQV: ~
dladbxor@LAPTOP-CONLNHQV:~$ g++ quick_sort.cpp radio_exchange_sort.cpp quick_sort.h
dladbxor@LAPTOP-CONLNHQV:~$ ./a.out
몇 개의 단어 생성을 하시겠습니까? 10
apy
ldo
mnlmtx
sip
fqh
g
iin
bucougci
flz
pff

apy
bucougci
flz
fqh
g
iin
ldo
mnlmtx
pff
sip
dladbxor@LAPTOP-CONLNHQV:~$
```

1.3.4.3 Comparison of 'MSD radix sort' and 'MSD radix exchange sort'

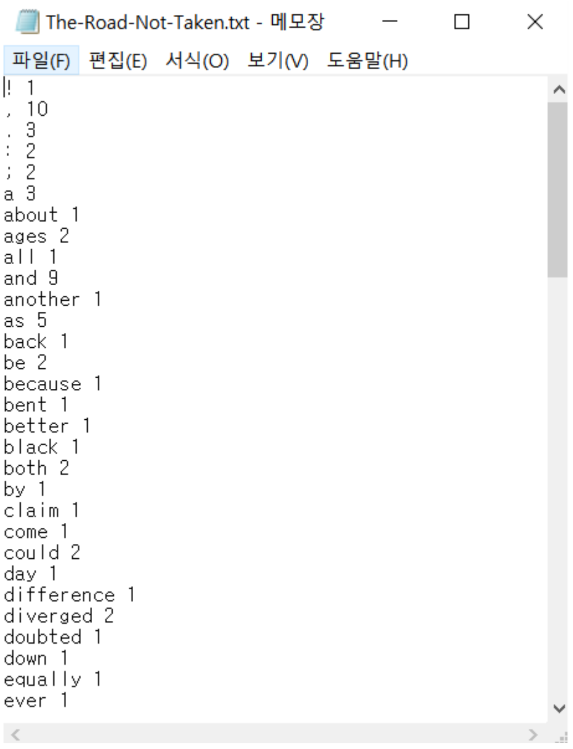


```
dladbxor@LAPTOP-CONLNHQV:~$ g++ count_sort.cpp radio_sort.cpp count_sort.h
dladbxor@LAPTOP-CONLNHQV:~$ ./a.out
elapsed_time: 3.13393 sec
dladbxor@LAPTOP-CONLNHQV:~$ g++ quick_sort.cpp radio_exchange_sort.cpp quick_sort.h
dladbxor@LAPTOP-CONLNHQV:~$ ./a.out
elapsed_time: 0.0600529 sec
dladbxor@LAPTOP-CONLNHQV:~$
```

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

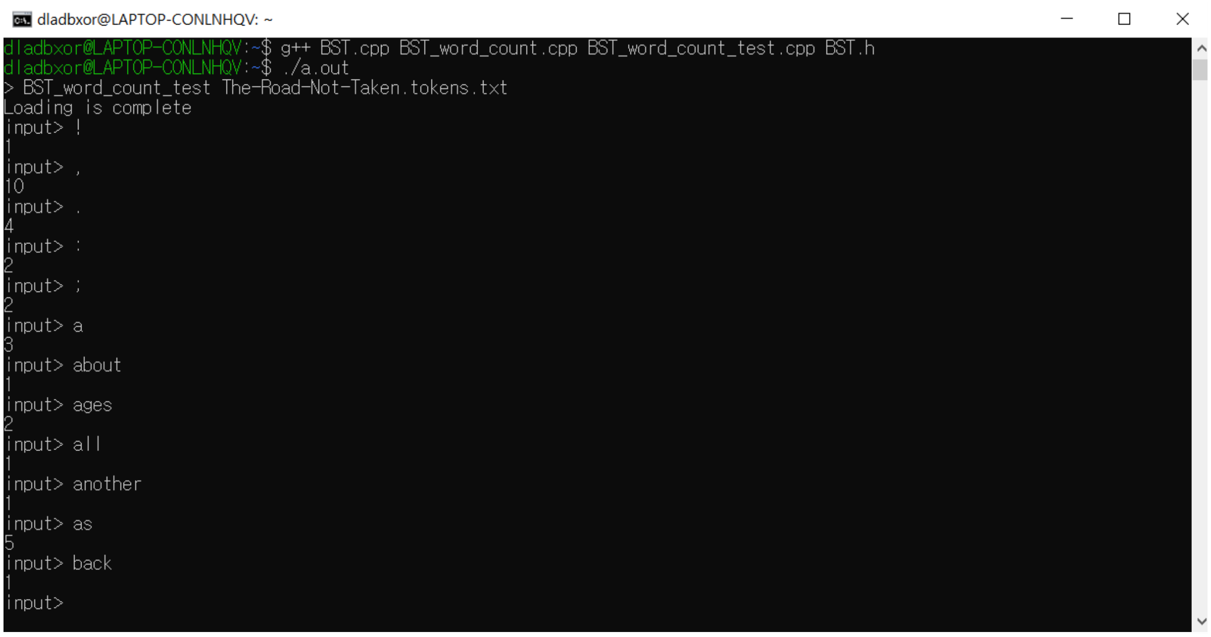
In code..

1.4.2 Calculating the 'Word Count' using the Binary Search Tree

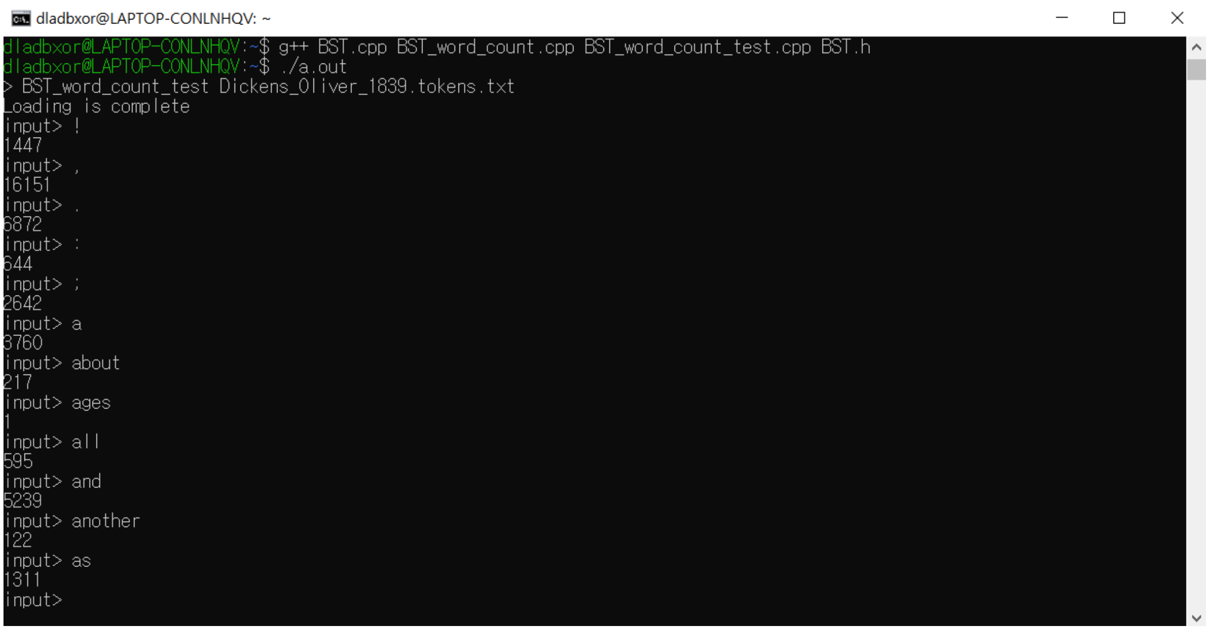

The-Road-Not-Taken.txt - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```
! 1
, 10
. 3
: 2
; 2
a 3
about 1
ages 2
all 1
and 9
another 1
as 5
back 1
be 2
because 1
bent 1
better 1
black 1
both 2
by 1
claim 1
come 1
could 2
day 1
difference 1
diverged 2
doubted 1
down 1
equally 1
ever 1
```

1.4.3 Binary search from the result of 'Word Count'



```
dladbxor@LAPTOP-CONLNHQV: ~
dladbxor@LAPTOP-CONLNHQV:~$ g++ BST.cpp BST_word_count.cpp BST_word_count_test.cpp BST.h
dladbxor@LAPTOP-CONLNHQV:~$ ./a.out
> BST_word_count_test The-Road-Not-Taken.tokens.txt
Loading is complete
input> !
1
input> ,
10
input> .
4
input> :
2
input> ;
2
input> a
3
input> about
1
input> ages
2
input> all
1
input> another
1
input> as
5
input> back
1
input>
```



```
dladbxor@LAPTOP-CONLNHQV: ~
dladbxor@LAPTOP-CONLNHQV:~$ g++ BST.cpp BST_word_count.cpp BST_word_count_test.cpp BST.h
dladbxor@LAPTOP-CONLNHQV:~$ ./a.out
> BST_word_count_test Dickens_Oliver_1839.tokens.txt
Loading is complete
input> !
1447
input> ,
16151
input> .
6872
input> :
644
input> ;
2642
input> a
3760
input> about
217
input> ages
1
input> all
595
input> and
5239
input> another
122
input> as
1311
input>
```

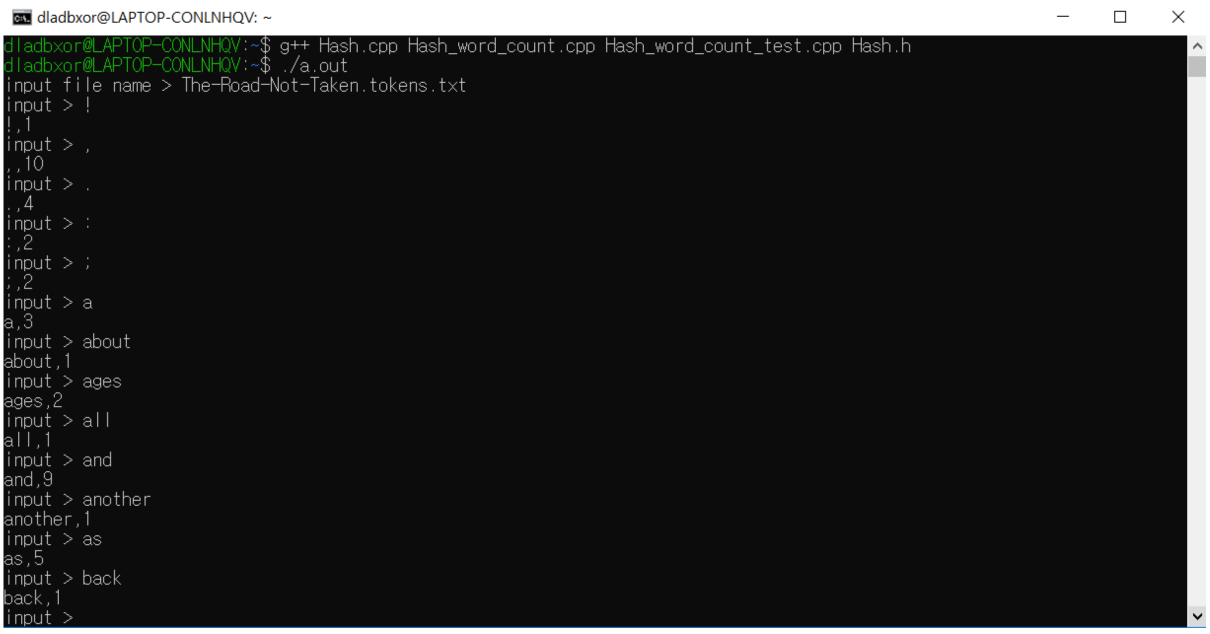
1.4.4 Binary Search from 'Word Count' result : Expansion of large data

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'

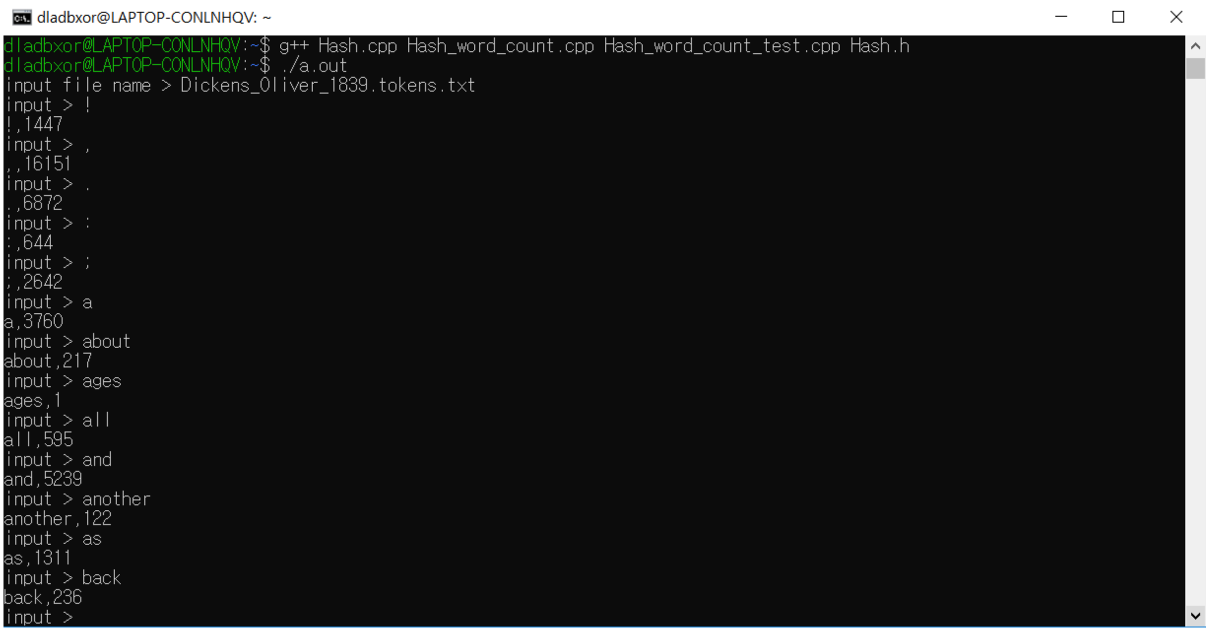
1.5.1 Implement Hash Class

In code..

1.5.2 Calculate and test 'Word Count' using a Hash



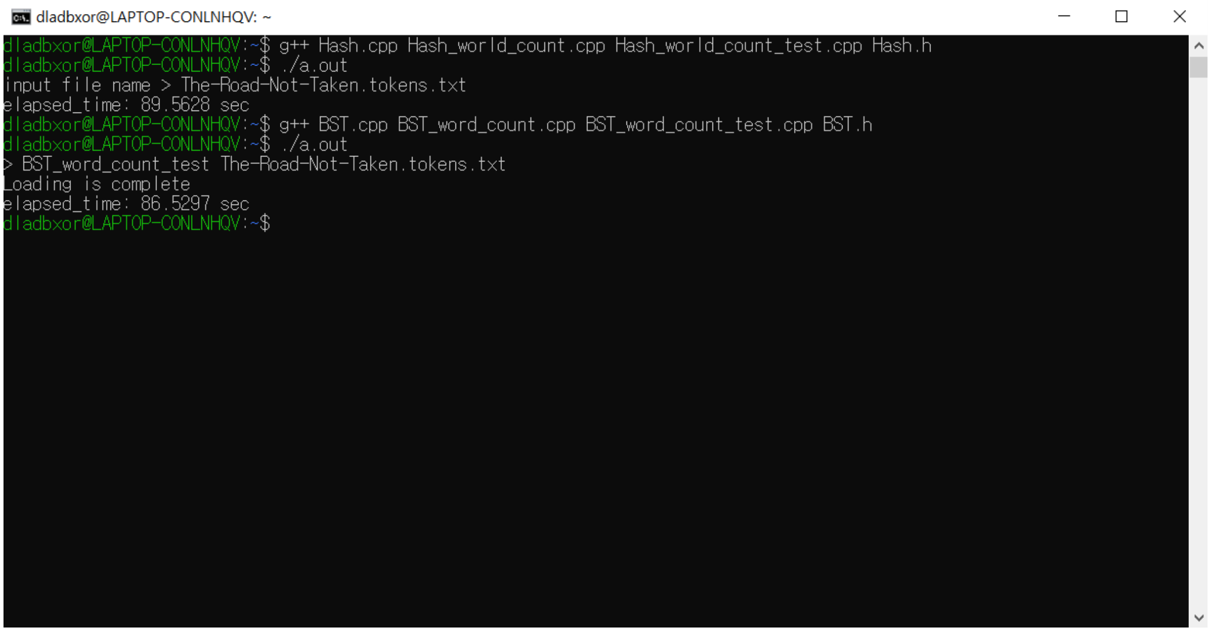
```
dladbxor@LAPTOP-CONLNHQV:~$ g++ Hash.cpp Hash_word_count.cpp Hash_word_count_test.cpp Hash.h
dladbxor@LAPTOP-CONLNHQV:~$ ./a.out
input file name > The-Road-Not-Taken.tokens.txt
input > !
!,1
input > ,
,,10
input > .
.,4
input > :
:,2
input > ;
;,2
input > a
a,3
input > about
about,1
input > ages
ages,2
input > all
all,1
input > and
and,9
input > another
another,1
input > as
as,5
input > back
back,1
input >
```



```
dladbxor@LAPTOP-CONLNHQV:~$ g++ Hash.cpp Hash_word_count.cpp Hash_word_count_test.cpp Hash.h
dladbxor@LAPTOP-CONLNHQV:~$ ./a.out
input file name > Dickens_Oliver_1839.tokens.txt
input > !
!,1447
input > ,
,,16151
input > .
.,6872
input > :
:,644
input > ;
;,2642
input > a
a,3760
input > about
about,217
input > ages
ages,1
input > all
all,595
input > and
and,5239
input > another
another,122
input > as
as,1311
input > back
back,236
input >
```

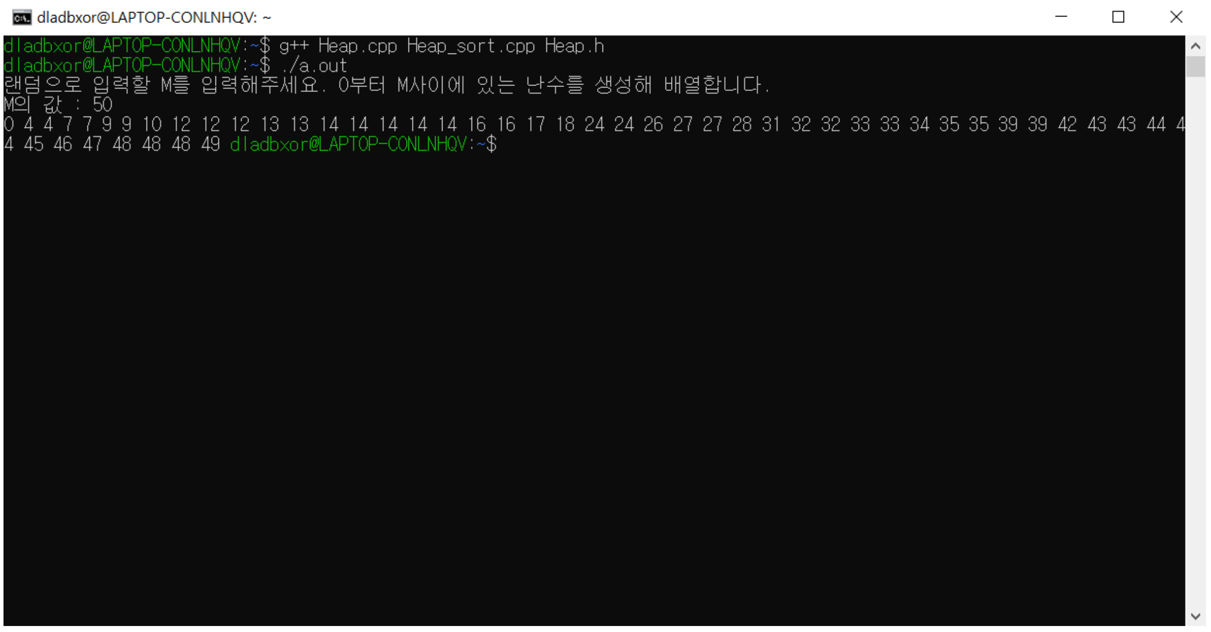
1.5.3 Compare ‘Hash’ and ‘Binary Search Tree’

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



```
dladbxor@LAPTOP-CONLNHQV: ~
dladbxor@LAPTOP-CONLNHQV:~$ g++ Hash.cpp Hash_world_count.cpp Hash_world_count_test.cpp Hash.h
dladbxor@LAPTOP-CONLNHQV:~$ ./a.out
input file name > The-Road-Not-Taken.tokens.txt
elapsed_time: 89.5628 sec
dladbxor@LAPTOP-CONLNHQV:~$ g++ BST.cpp BST_word_count.cpp BST_word_count_test.cpp BST.h
dladbxor@LAPTOP-CONLNHQV:~$ ./a.out
> BST_word_count_test The-Road-Not-Taken.tokens.txt
Loading is complete
elapsed_time: 86.5297 sec
dladbxor@LAPTOP-CONLNHQV:~$
```

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



```
dladbxor@LAPTOP-CONLNHQV: ~
dladbxor@LAPTOP-CONLNHQV:~$ g++ Heap.cpp Heap_sort.cpp Heap.h
dladbxor@LAPTOP-CONLNHQV:~$ ./a.out
랜덤으로 입력할 M를 입력해주세요. 0부터 M사이에 있는 난수를 생성해 배열합니다.
M의 값 : 50
0 4 4 7 7 9 9 10 12 12 12 13 13 14 14 14 14 14 16 16 17 18 24 24 26 27 27 28 31 32 32 33 33 34 35 35 39 39 42 43 43 44 4
4 45 46 47 48 48 48 49 dladbxor@LAPTOP-CONLNHQV:~$
```