EECS2040 Data Structure Hw #6 (Chapter 7 Sorting, Chapter 8 Hashing)
due date 6/27/2021
by 107061123, 孫元駿

Part 2 Coding

You should submit:

(a) All your source codes (C++ file).

(b) Show the execution trace of your program.

1. (50%) Sorting:

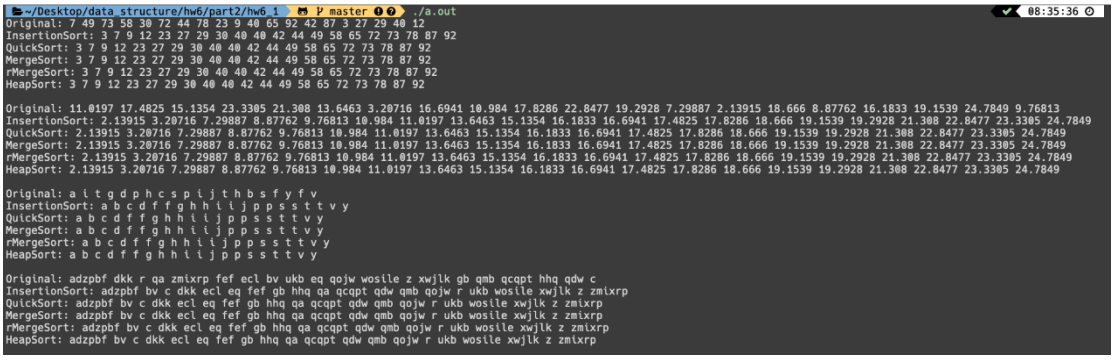
   Write a C++ program to perform 5 different sorting , insertion sort, median-of-three quick sort, iterative merge sort, recursive merge sort, and heap sort, on lists of characters, integer, floating point numbers, and C++ strings.

   a. You need to write the 5 sorting function templates (refer to example programs in textbook or pptx)
   b. Randomly generate a list of 20 characters as an input unsorted list.
   c. Randomly generate a list of 20 integers as an input unsorted list.
   d. Randomly generate a list of 20 floats as an input unsorted list.
   e. Randomly generate a list of 20 string objects as an input unsorted list.

   Show your results using the above 3 lists in your program.

   How to run my program:

   The program will randomly generate a list of 20 intergers, a list of 20 floats, a list of 20 characters and a list of 20 strings. Then they will respectively sort by five ways and print the list out.



2. (50%) Hashing:

Write a C++ program to implement two simple symbol tables (dictionaries) using hash table with linear probing for collision and hash table with chaining. For simplicity,

a. Consider storing only the key (need not consider the (key, value) pair) in the symbol tables.
b. Furthermore, the key is a variable-length character array with the first character of the key is an alphabet, e.g., abc, abcde, b, bye, cool,…
c. Consider a simple hash function using only the first character of key to hash, so h(abcde) = h(abc), h(b) = h(bye),.., etc. Therefore, collision can happen frequently.
d. The initial hash table size can be set to 26 since we have 26 alphabets which are the hashed keys.

Create 2 symbol table classes for linear probing and chaining, respectively. Both must implement at least the following functions:

Constructor,
Insert(key)
Search(key)

You may add other functions needed in your program.

Your main function may contains code like:

SymbolTable1 d1;
Setup at least 10 key objects
Insert those 10 keys into d1.
Display d1
Demo the search function of d1 (try at least 5 keys)

SymbolTable2 d2;
Setup at least 10 key objects
Insert those 10 keys into d1.
Display d2
Demo the search function of d2(try at least 5 keys)

How to run my program:

First input the number of key to store, and input all the key. Then print out all key in the dictionary. Last, input how many keys do you want to get, and insert the keys, then it will show if the key is exist or not.

It will execute two time, first time is linear probing, the second time is chaining.

```
~/Desktop/data_structure/hw6/part2/hw6_2  master !? ./a.out
LinearProbing
12
ga
d
a
g
l
a2
a1
a3
a4
z
za
e

Demo:
0: a
1: a2
2: a1
3: d
4: a3
5: a4
6: ga
7: g
8: za
9: e
10:
11: l
12:
13:
14:
15:
16:
17:
18:
19:
20:
21:
22:
23:
24:
25: z

Get how many keys: 5
What do you want to search: a
d1.Get(a) = a
What do you want to search: a2
d1.Get(a2) = a2
What do you want to search: d
d1.Get(d) = d
What do you want to search: za
d1.Get(za) = za
What do you want to search: abcdefg
d1.Get(abcdefg) = No search

Chaining
12
ga
```

```
d1.Get(za) = za
What do you want to search: abcdefg
d1.Get(abcdefg) = No search

Chaining
12
ga
d
a
g
l
a2
a1
a3
a4
z
za
e

[0] -> a4 -> a3 -> a1 -> a2 -> a -> 0
[1] -> 0
[2] -> 0
[3] -> d -> 0
[4] -> e -> 0
[5] -> 0
[6] -> g -> ga -> 0
[7] -> 0
[8] -> 0
[9] -> 0
[10] -> 0
[11] -> l -> 0
[12] -> 0
[13] -> 0
[14] -> 0
[15] -> 0
[16] -> 0
[17] -> 0
[18] -> 0
[19] -> 0
[20] -> 0
[21] -> 0
[22] -> 0
[23] -> 0
[24] -> 0
[25] -> za -> z -> 0

Get how many keys: 5
What do you want to search: a
d2.Get(a) = a
What do you want to search: a2
d2.Get(a2) = a2
What do you want to search: d
d2.Get(d) = d
What do you want to search: za
d2.Get(za) = za
What do you want to search: abcdefg
d2.Get(abcdefg) = No search

~/Desktop/data_structure/hw6/part2/hw6_2  master
```