EE2410 Data Structure Coding HW #6 – Sorting Hashing (Chapter 7 ~ 8)

due date 6/23/2024 (Sun.), 23:59

You should submit:

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

(b) Show the execution trace of your program, i.e., write a client main() to demonstrate all functions you designed using example data.

Sorting:

1. (50%)

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

1. You need to write the 5 sorting function templates (refer to example programs in textbook or pptx)
2. Randomly generate a list of 20 characters as an input unsorted list.
3. Randomly generate a list of 20 integers as an input unsorted list.
4. Randomly generate a list of 20 floating point numbers as an input unsorted list.
5. Randomly generate a list of 20 string objects as an input unsorted list.

Show your results using the above 4 lists in your program.

sol:

Execution trace:

```
sunpierce@pierces-Air output % cd "/Users/sunpierce/C_C++/EE-DS/HW_6/output"
sunpierce@pierces-Air output % ./"6.1_sorting"
================================================================================ characters ================================================================================
Random Characters: z b o w i e u a c f z m t b n v j f z i
By Insertion Sort: a b b c e f f i i j m n o t u v w z z z
Random Characters: g b g d c z h w c x d h b u c l i l d s
By Quick Sort: b b c c c d d d g g h h i l l s u w x z
Random Characters: t s h y u j p t w y b n t k g i g o l z
By Iterative Merge Sort: b g g h i j k l n o p s t t t u w y y z
Random Characters: e p r s r l c x p v s v s s s b g n d f
By Recursive Merge Sort: b c d e f g l n p p r r s s s s s v v x
Random Characters: s r f d o s v j l o f d s k k f w o e k
By Heap Sort: d d e f f f j k k k l o o o r s s s v w
================================================================================ integers ================================================================================
Random Integers: 0 30 56 2 40 13 95 0 80 72 49 95 92 36 38 23 80 47 6 67
By Insertion Sort: 0 2 4 6 13 23 30 36 38 40 47 49 56 67 72 80 80 92 95 95
Random Integers: 4 75 80 24 76 94 64 83 24 37 38 13 73 37 27 83 98 72 99 57
By Quick Sort: 13 24 24 27 32 37 37 38 57 64 72 73 75 76 80 83 83 94 98 99
Random Integers: 4 56 23 7 62 23 90 7 87 98 79 69 24 65 48 39 0 25 88 11
By Iterative Merge Sort: 0 7 7 11 23 23 24 24 25 39 48 56 62 65 69 79 87 88 90 98
Random Integers: 4 31 93 31 79 21 54 28 71 6 17 46 93 81 39 73 34 8 91 33
By Recursive Merge Sort: 6 8 11 17 21 28 31 31 33 34 39 46 54 71 73 79 81 91 93 93
Random Integers: 4 79 34 88 35 89 84 7 98 87 94 95 14 89 63 71 88 61 96 69
By Heap Sort: 7 14 34 35 61 63 69 71 76 79 84 87 88 88 89 89 94 95 96 98
================================================================================ float ================================================================================
Random Floats: 0 97.2828 32.4873 14.8558 80.8713 3.99575 56.5727 17.955 69.0943 67.0748 26.201 59.6135 23.8747 61.3079 1.50369 72.5015 32.8918 12.7133 72.7403 46.0481
By Insertion Sort: 1.50369 3.99575 12.7133 14.8558 17.955 23.8747 26.201 31.0293 32.4873 32.8918 46.0481 56.5727 59.6135 61.3079 67.0748 69.0943 72.5015 72.7403 80.8713 97.2828
Random Floats: 31.0293 9.54193 71.2 58.5959 21.6644 13.1214 31.2639 53.0819 47.7367 10.4961 8.02778 22.8345 79.5281 28.4376 50.8617 32.4541 55.8644 12.7372 73.9028 83.6852
By Quick Sort: 8.02778 9.54193 10.4961 12.7372 13.1214 21.6644 22.8345 28.4376 31.2639 32.4541 47.7367 50.8617 53.0819 55.8644 58.5959 71.2 73.9028 79.5281 83.6852 97.5133
Random Floats: 31.0293 6.27403 47.7018 24.7769 25.1934 25.0057 70.575 54.6035 20.5242 51.0123 63.5457 13.1615 5.32487 95.0075 91.7608 24.5433 98.6077 98.7795 86.4843 42.0209
By Iterative Merge Sort: 5.32487 6.27403 13.1615 20.5242 24.5433 24.7769 25.0057 25.1934 42.0209 44.4775 47.7018 51.0123 54.6035 63.5457 70.575 86.4843 91.7608 95.0075 98.6077 98.7795
Random Floats: 31.0293 34.0335 0.942147 34.6726 43.1708 72.3914 82.4086 41.8554 63.5893 45.6909 26.6044 39.7122 43.3486 59.3237 52.9459 61.9501 94.7079 55.2958 56.2747 8.08156
By Recursive Merge Sort: 0.942147 8.08156 26.6044 26.7684 34.0335 34.6726 39.7122 41.8554 43.1708 43.3486 45.6909 52.9459 55.2958 56.2747 59.3237 61.9501 63.5893 72.3914 82.4086 94.7079
Random Floats: 31.0293 96.7519 9.74315 53.1237 49.2243 13.4465 95.343 30.404 99.6147 24.9551 20.971 59.5511 75.8048 52.1011 63.6609 48.355 1.74032 49.4844 83.551 41.7158
By Heap Sort: 1.74032 9.74315 13.4465 16.9824 20.971 24.9551 30.404 41.7158 48.355 49.2243 49.4844 52.1011 53.1237 59.5511 63.6609 75.8048 83.551 95.343 96.7519 99.6147
================================================================================ strings ================================================================================
Random Strings:  geeuk yyjad wwazo epymp lmekt ctuvn pcjka avkbm srzih nqckf hqkcm nxoul zeddi wuglr soeyy shdeg sysjb zyurs hmdya
By Insertion Sort: avkbm ctuvn epymp geeuk hmdya hqkcm lmekt nqckf nxoul omzwk pcjka shdeg soeyy srzih sysjb wuglr wwazo yyjad zeddi zyurs
Random Strings: omzwk ufwqk pvlfa xymjo prtab aigmz efxjz tbbpo csufc kopnh stbtz hclvn fpoep jnujq gccrz wqlwk oyiil wlceb qrdbm wyfyb
By Quick Sort: aigmz csufc efxjz fpoep gccrz hclvn jnujq kopnh oyiil prtab pvlfa qrdbm stbtz tbbpo tuxvz ufwqk wlceb wqlwk wyfyb xymjo
Random Strings: omzwk xvbay mzjoi bknwq zzaws isdpg tsjmd ouune nljaj gyvrs isczi gwppp aetor pirdh bdxpo qmiwd nrdcp hcdwi dqhxr qulob
By Iterative Merge Sort: aetor bdxpo bknwq dqhxr gwppp gyvrs hcdwi iszci isdpg miamr mzjoi nljaj nrdcp ouune pirdh qmiwd qulob tsjmd xvbay zzaws
Random Strings: omzwk minah zlupj zsuui uwqsr vwsol phfbl ixzcu kmtlr degtf upmru rpnyz snlxq qdexh kmcis xotny ggpbj cuxwj efmov lujmd
By Recursive Merge Sort: cuxwj degtf efmov ggpbj ixzcu kmcis kmtlr lujmd minah phfbl qdexh rpnyz snlxq upmru uwqsr vwsol xhvei xotny zlupj zsuui
Random Strings: omzwk kiaoh iquyg mgirg rsnyd fpypj xbytb pkhjk ksobx avzll tjiqf vddxs pgnxu tfomc papaa oxnfa uliug rmirr cptbe czpws
By Heap Sort: avzll cptbe czpws fpypj iquyg kiaoh ksobx mgirg oxnfa papaa pgnxu pkhjk rmirr rsnyd rwkkp tfomc tjiqf uliug vddxs xbytb
sunpierce@pierces-Air output %
```

Hashing:

2. (50%)

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** where 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)

sol:

```
Execution trace:
```

```
sunpierce@pierces-Air output % cd "/Users/sunpierce/C_C++/EE-DS/HW_6/output"
sunpierce@pierces-Air output % ./"6.2_hash"
============== Linear probing hash table ==============
Successful insertion for key [aafe]
Successful insertion for key [add]
Successful insertion for key [akls]
Successful insertion for key [anhlk]
Successful insertion for key [akd]
Successful insertion for key [edl]
Successful insertion for key [g]
Successful insertion for key [klja]
Successful insertion for key [qoi]
Successful insertion for key [zlk]
Hash Table:
    i     ht[i]
    0      aafe
    1       add
    2      akls
    3     anhlk
    4       akd
    5       edl
    6         g
    7         -
    8         -
    9         -
   10      klja
   11         -
   12         -
   13         -
   14         -
   15         -
   16       qoi
   17         -
   18         -
   19         -
   20         -
   21         -
   22         -
   23         -
   24         -
   25       zlk
Key [anhlk] found
Key [akd] found
Key [aafe] found
Key [qo] not found
Key [b] not found
```

```
============== Chaining hash table ==============
Successful insertion for key [aafe]
Successful insertion for key [add]
Successful insertion for key [akls]
Successful insertion for key [anhlk]
Successful insertion for key [akd]
Successful insertion for key [edl]
Successful insertion for key [g]
Successful insertion for key [klja]
Successful insertion for key [qoi]
Successful insertion for key [zlk]
Hash Table:
    0  aafe --> add --> akls --> anhlk --> akd --> NULL
    1  NULL
    2  NULL
    3  NULL
    4  edl --> NULL
    5  NULL
    6  g --> NULL
    7  NULL
    8  NULL
    9  NULL
   10  klja --> NULL
   11  NULL
   12  NULL
   13  NULL
   14  NULL
   15  NULL
   16  qoi --> NULL
   17  NULL
   18  NULL
   19  NULL
   20  NULL
   21  NULL
   22  NULL
   23  NULL
   24  NULL
   25  zlk --> NULL
Key [anhlk] found
Key [akd] found
Key [aafe] found
Key [qo] not found
Key [b] not found
sunpierce@pierces-Air output %
```