CS401 Final Project Documentation

A20354898
Wanglin Song

a. Problem Specifications

The project requires us to write an application to read data (including integers, floating point numbers and strings from input file), create lists (each list only contains one single type of data) from the input data.

Then we need to be able to sort the data lists (using at least 3 types of sorting algorithm, bubble sort, merge sort and heap sort) and search data from lists (using linear sort and binary search).

Furthermore, we should be able to modify existing lists and still be able to perform sort/search functions on them.

b. Software Specifications

There are 6 components in the main application:

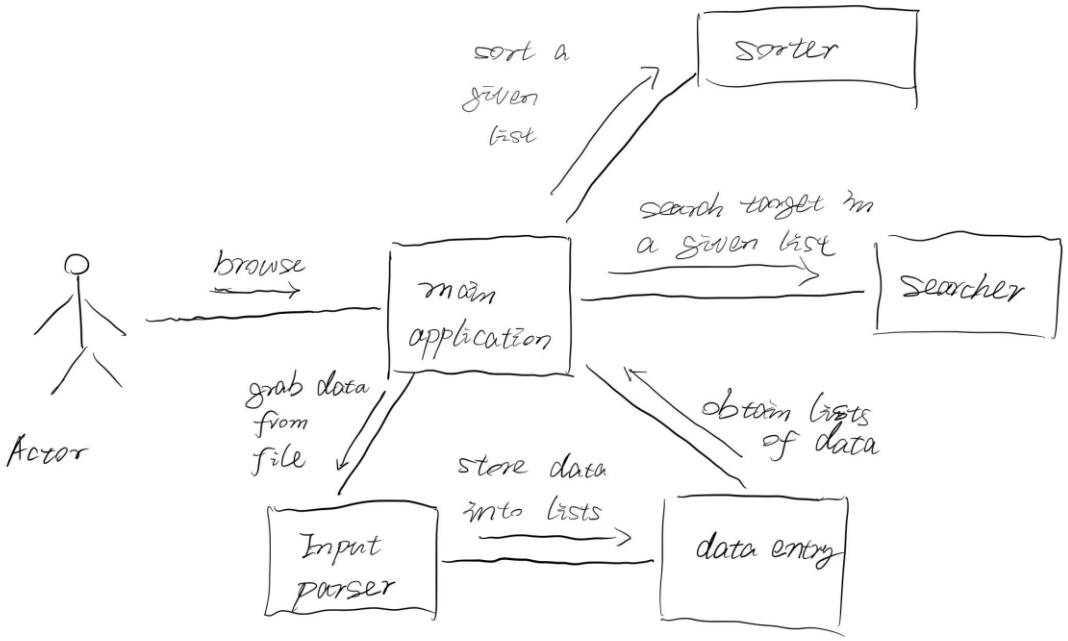
1) A data-entry hash-table called “data”, that stores lists of data read from files. We can grab a certain list by calling its key(index) in the hash-table. The data-entry hash-table can also rehash when it reaches 75% of its maximum capacity.
2) A console prompter (called “promp”) that read user input data from keyboard, parse it into different kinds of data (integer, floating point number or string) and return it.
3) A file parser (called “parser”) that read data from file to create new lists or extend existing lists.
4) An object (called “ops”) that contains the commonly used operations of main menu and returns the operation option entered by the user.
5) A data searcher (called “searcher”) that could do linear search and binary search to obtain index of target elements from a list in the data pile.
6) A data searcher (called “sorter”) that could sort a given list in the data pile with 4 types of sorting algorithms: bubble sort, merge sort, heap sort and quick sort.

The list structure used by the application is an array-based list (referenced from the textbook) that support $O(1)$ access, linear search and unlimited capacity.

Run the “main” function in default package to use the application.

c. Design diagram document and pseudo-code

Design diagram:



Pseudo-code for 3 types of sorting:

```
def bubbleSort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]


def mergeSort(nums):
    if len(nums)<2:
        return
    mid = len(nums)//2
    L, R = nums[:mid], nums[mid:]
    mergeSort(L)
    mergeSort(R)
    i, j, k = 0, 0, 0
    while i < len(L) and j <len(R):
        if L[i] < R[j]:
            nums[k] = L[i]
            i += 1
        else:
            nums[k] = R[j]
            j += 1
        k += 1
    while i < len(L):
```

```
            nums[k] = L[i]
            i += 1
            k += 1
        while j < len(R):
            nums[k] = R[j]
            j += 1
            k += 1


def heapSort(arr):
    def heapify(nums, n, i):
        largest, l, r = i, 2 * i + 1, 2 * i + 2
        if l < n and nums[i] < nums[l]:
            largest = l
        if r < n and nums[largest] < nums[r]:
            largest = r
        if largest != i:
            nums[i], nums[largest] = nums[largest], nums[i]
            heapify(nums, n, largest)
    n = len(arr)
    for i in range(n, -1, -1):
        heapify(arr, n, i)
    for i in range(n - 1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i]   # swap
        heapify(arr, i, 0)
```

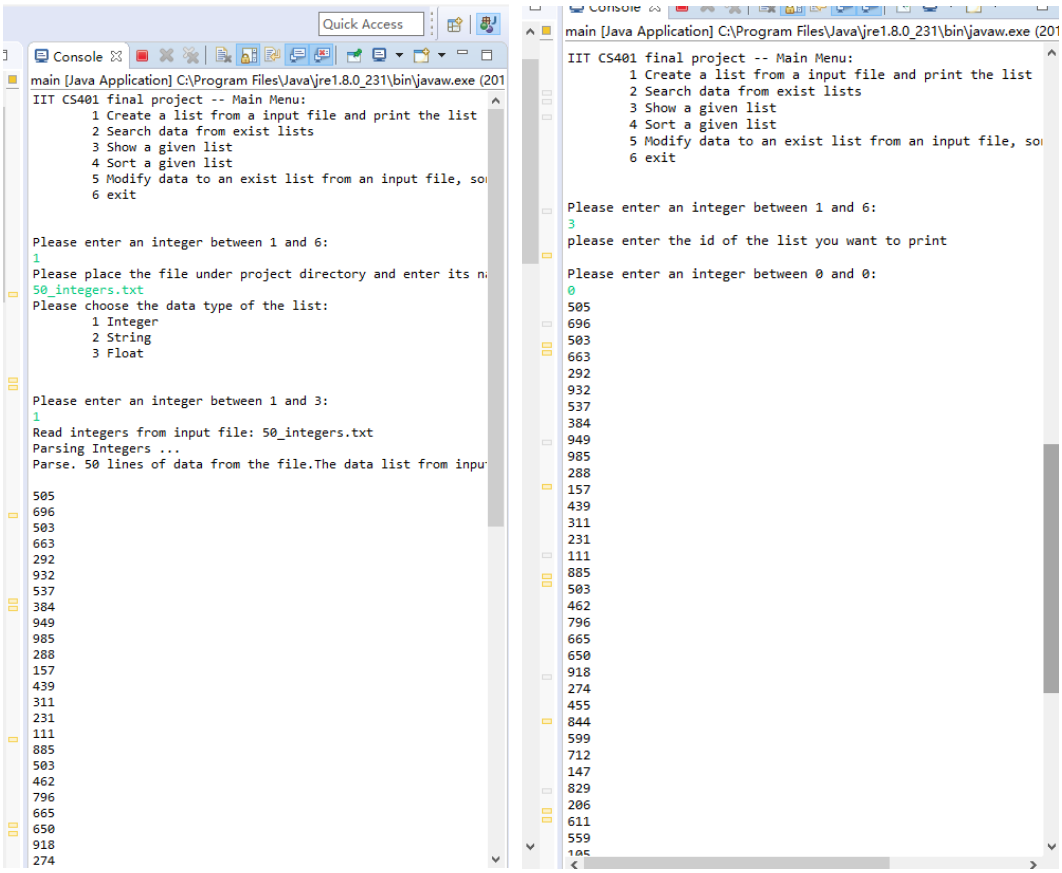
d. Source code (see project directory)

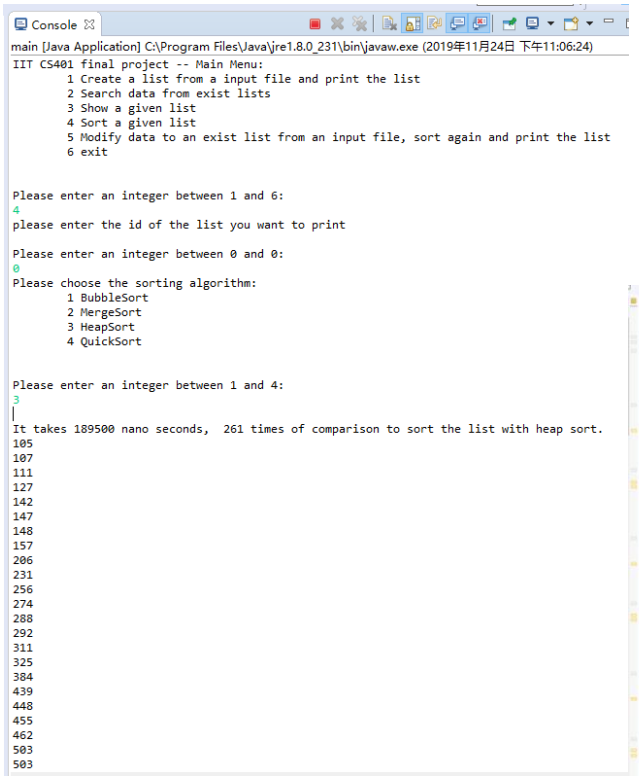
e. Operation document:

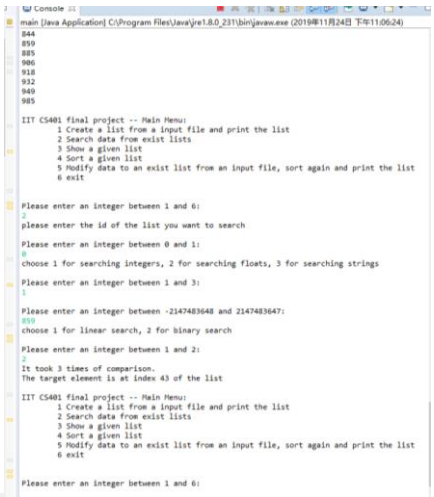
Run the main function under default package. It should give you 5 options:

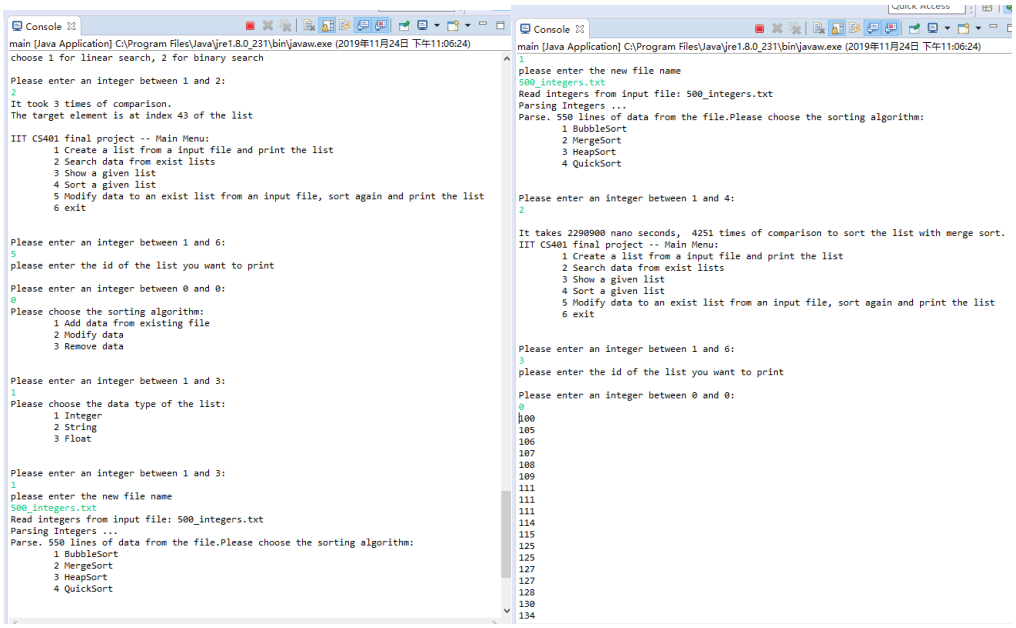
1) read a list of data from file into data map and print it out,
2) search data from an existing list in the data map
3) show an existing list in the data map
4) sort an existing list in the data map (bubble sort, merge sort, heap sort and quick sort are available for selection. The performance statistics of sorting algorithm is printed at the end).
5) modify an existing list in the data map (add more data from file, remove or replace data at a given index)
6) exit the searching and sorting application.

Note that for modifying an existing list, it is recommended that you print out the list to see the data type of the list, so you don't add different type of data into the list.


Quick Access
Console
main [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\javaw.exe (201
IIT CS401 final project -- Main Menu:
1 Create a list from a input file and print the list
2 Search data from exist lists
3 Show a given list
4 Sort a given list
5 Modify data to an exist list from an input file, so
6 exit
Please enter an integer between 1 and 6:
1
Please place the file under project directory and enter its n
50_integers.txt
Please choose the data type of the list:
1 Integer
2 String
3 Float
Please enter an integer between 1 and 3:
1
Read integers from input file: 50_integers.txt
Parsing Integers ...
Parse. 50 lines of data from the file.The data list from inpu
505
696
503
663
292
932
537
384
949
985
288
157
439
311
231
111
885
503
462
796
665
650
918
274
main [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\javaw.exe (201
IIT CS401 final project -- Main Menu:
1 Create a list from a input file and print the list
2 Search data from exist lists
3 Show a given list
4 Sort a given list
5 Modify data to an exist list from an input file, so
6 exit
Please enter an integer between 1 and 6:
3
please enter the id of the list you want to print
Please enter an integer between 0 and 0:
0
505
696
503
663
292
932
537
384
949
985
288
157
439
311
231
111
885
503
462
796
665
650
918
274
455
844
599
712
147
829
206
611
559


Console
main [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\javaw.exe (2019年11月24日 下午11:06:24)
IIT CS401 final project -- Main Menu:
1 Create a list from a input file and print the list
2 Search data from exist lists
3 Show a given list
4 Sort a given list
5 Modify data to an exist list from an input file, sort again and print the list
6 exit
Please enter an integer between 1 and 6:
4
please enter the id of the list you want to print
Please enter an integer between 0 and 0:
0
Please choose the sorting algorithm:
1 BubbleSort
2 MergeSort
3 HeapSort
4 QuickSort
Please enter an integer between 1 and 4:
3
It takes 189500 nano seconds, 261 times of comparison to sort the list with heap sort.
105
107
111
127
142
147
148
157
206
231
256
274
288
292
311
325
384
439
448
455
462
503
503





f. Testing document:

The sample input files (txt format) are in the file directory. N (50, 500 or 1000) integers, floating point numbers or strings in the text files.

g. **Future improvement:**

Specify the type of data stored in a list, so we can prevent adding different types of data into one single list.

Add GUI for the application.

Implement additional sorting algorithm for the application.

h. **Project schedule:**

11.03 ~ 11.10: high level design and pseudo code. Separate the tasks into different components.

11.11 ~ 11.17: low level implementation of sorting and searching algorithm

11.18 ~ 11.24: low level implementation of user I/O and menu. Complete software documentation and add more comments.