A Data Sorting and Searching Program

CS 401 Project Documentation

Dawei Wang

Nov 2019

Requirements

- ✧ Program could read data file
- ✧ Restore data file into an appropriate data structure
- ✧ Allow users to edit data, including add, delete and adjust
- ✧ Program have ability to deal with data type like string, integer and float
- ✧ Allow users to choose sort algorithm
- ✧ Could sort data list by using bubble sort, heap sort and merge sort
- ✧ Support linear search for unsorted list
- ✧ Convert sorted list to binary search tree and run search

Analysis

Software should satisfy the following functions:

- ✧ Open and read a data file that suits requirements by giving a file path
- ✧ Array List collection could fit in the editing functions, add more personalized functions into its interface
- ✧ Software should read every single data from the file and store it to an array list
- ✧ Design a CHI for user to choose editing functions, sorting algorithms and searching algorithms
- ✧ Following user's choice and run editing functions, sorting algorithms and searching algorithms
- ✧ Linear search should run with unsorted list
- ✧ Sorting algorithms should be able to read stored array list, sort it and store the sorted list as an array list collection
- ✧ BST searching algorithm should read the sorted array list and restore it into a balanced binary search tree and then run searching function
- ✧ For different data type, software should deal with each of them by their own data type corresponding storing, sorting and searching algorithms
- ✧ User could edit or search the data more than once
- ✧ Running time or sorting and searching should be showed as a message

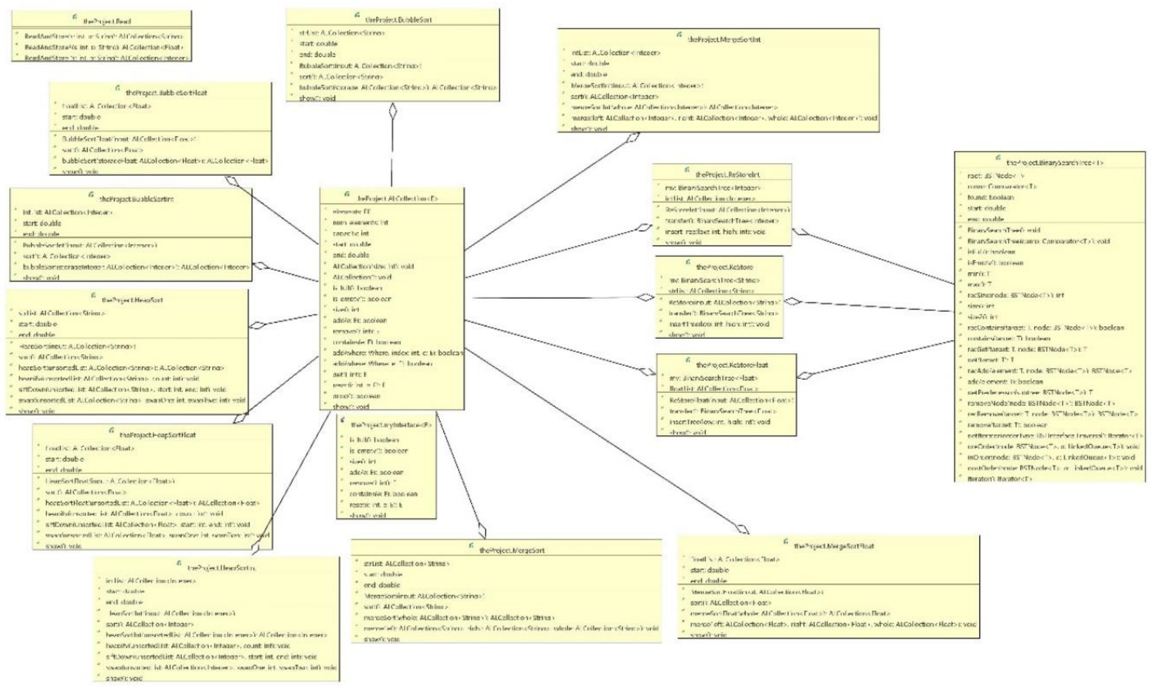
✧ The whole data list should be displayed every time after reading file in, editing data, and sorting process

✧ After a binary search tree created, the inorder traversal should be provided for users' inspection

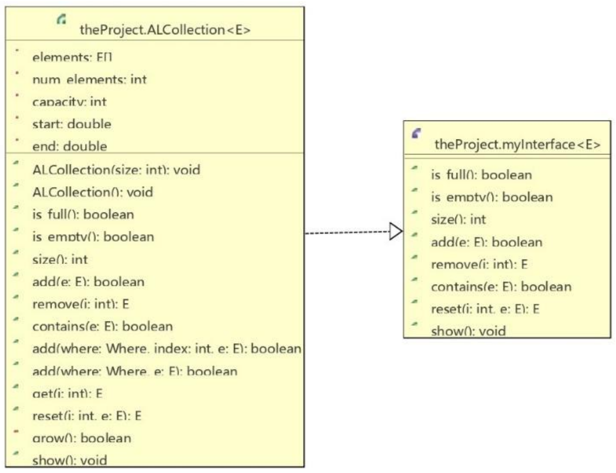
Software Design

Check appendix for the high-resolution figure.

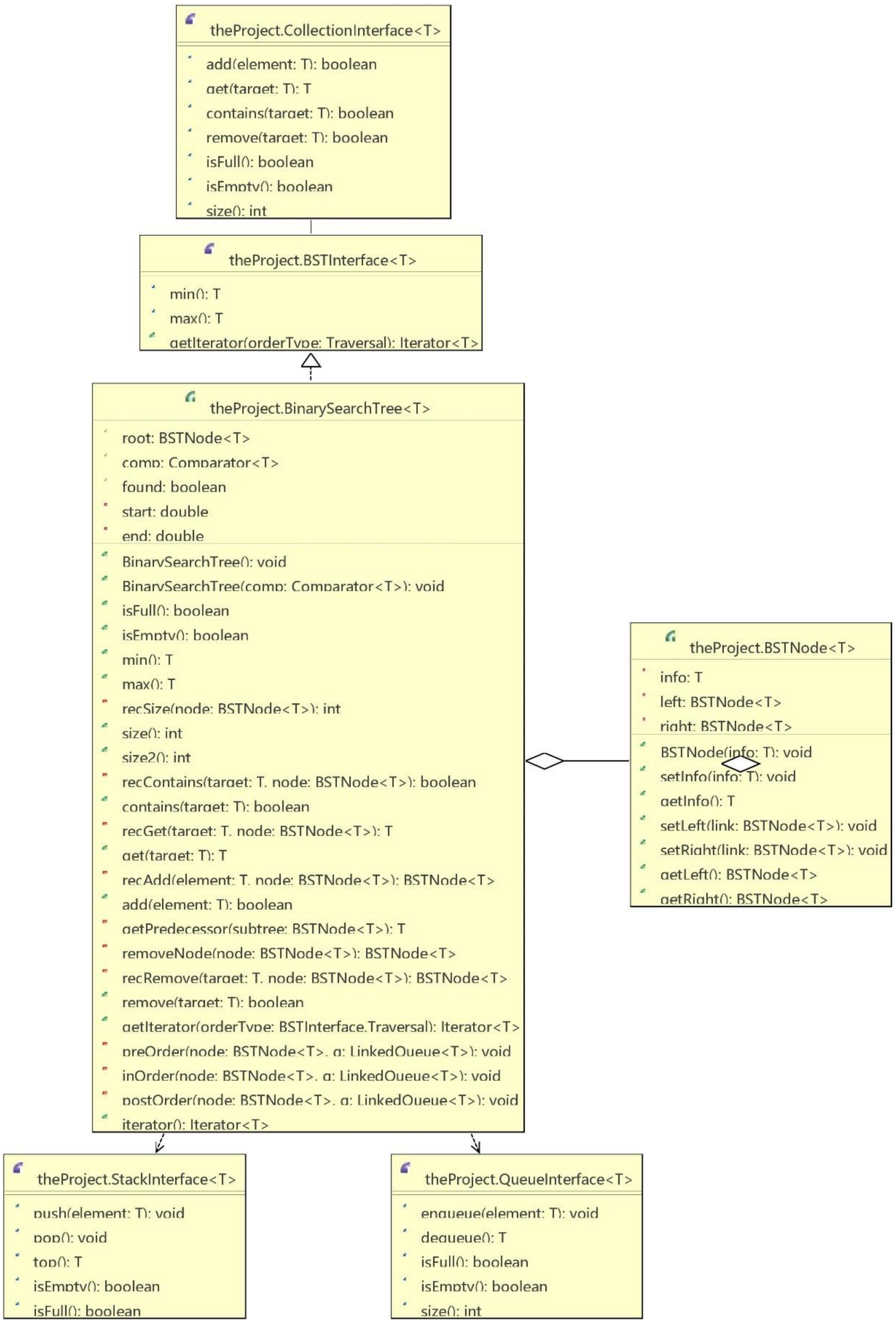
✧ Major class UML


theProject.ALCollection<E>
theProject.BubbleSort
theProject.MergeSortInt
theProject.BinarySearchTree<T>
theProject.HeapSort
theProject.MergeSort
theProject.MergeSortFloat
theProject.myInterface<E>

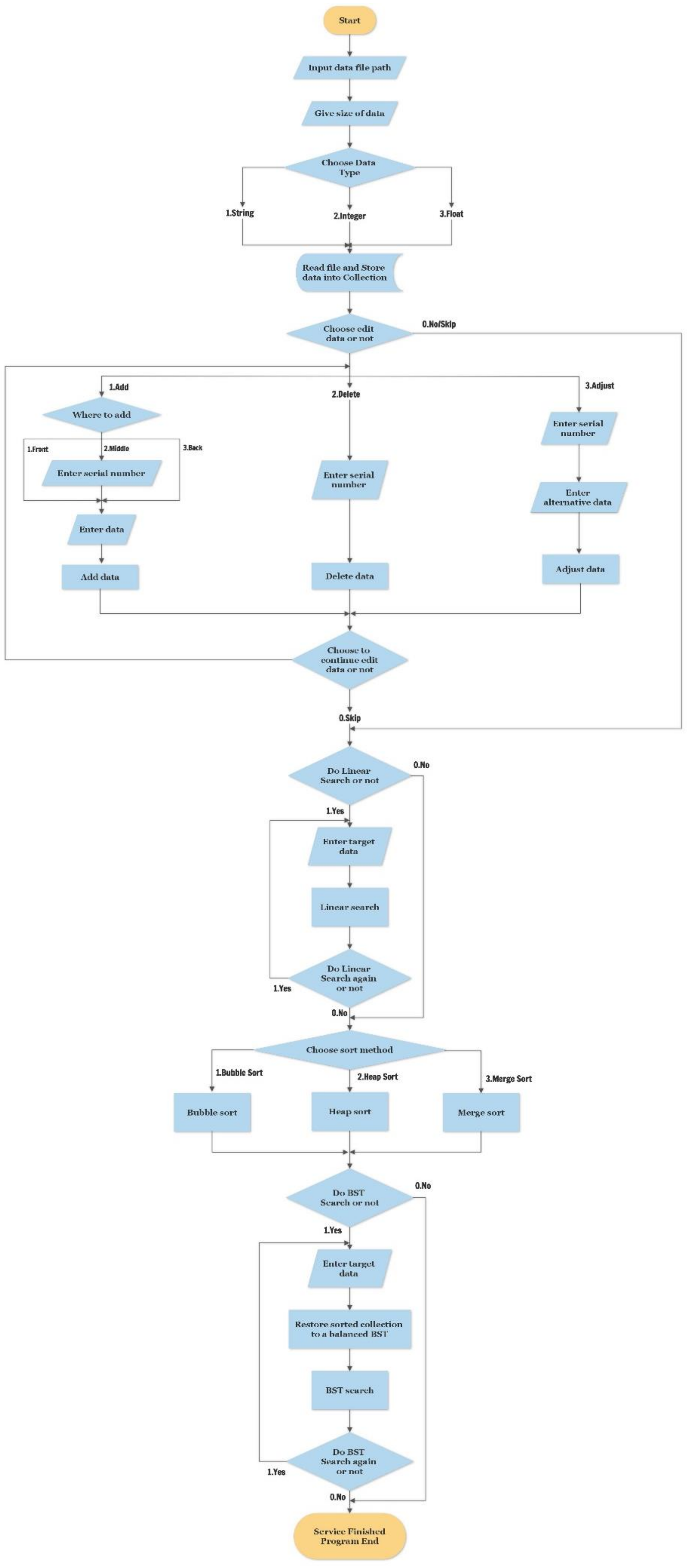
✧ ALCollection Interface UML



✧ BST Interface UML


theProject.CollectionInterface<T>
add(element: T): boolean
get(target: T): T
contains(target: T): boolean
remove(target: T): boolean
isFull(): boolean
isEmpty(): boolean
size(): int
theProject.BSTInterface<T>
min(): T
max(): T
getIterator(orderType: Traversal): Iterator<T>
theProject.BinarySearchTree<T>
root: BSTNode<T>
comp: Comparator<T>
found: boolean
start: double
end: double
BinarySearchTree(): void
BinarySearchTree(comp: Comparator<T>): void
isFull(): boolean
isEmpty(): boolean
min(): T
max(): T
recSize(node: BSTNode<T>): int
size(): int
size2(): int
recContains(target: T, node: BSTNode<T>): boolean
contains(target: T): boolean
recGet(target: T, node: BSTNode<T>): T
get(target: T): T
recAdd(element: T, node: BSTNode<T>): BSTNode<T>
add(element: T): boolean
getPredecessor(subtree: BSTNode<T>): T
removeNode(node: BSTNode<T>): BSTNode<T>
recRemove(target: T, node: BSTNode<T>): BSTNode<T>
remove(target: T): boolean
getIterator(orderType: BSTInterface.Traversal): Iterator<T>
preOrder(node: BSTNode<T>, q: LinkedQueue<T>): void
inOrder(node: BSTNode<T>, q: LinkedQueue<T>): void
postOrder(node: BSTNode<T>, q: LinkedQueue<T>): void
iterator(): Iterator<T>
theProject.BSTNode<T>
info: T
left: BSTNode<T>
right: BSTNode<T>
BSTNode(info: T): void
setInfo(info: T): void
getInfo(): T
setLeft(link: BSTNode<T>): void
setRight(link: BSTNode<T>): void
getLeft(): BSTNode<T>
getRight(): BSTNode<T>
theProject.StackInterface<T>
push(element: T): void
pop(): void
top(): T
isEmpty(): boolean
isFull(): boolean
theProject.QueueInterface<T>
enqueue(element: T): void
dequeue(): T
isFull(): boolean
isEmpty(): boolean
size(): int

✧ Flowchart of the usage



User's Manual

Introduction

This program is designed to help users to sort and search through there data file. User could give a certain txt data file as input and the program will read and store those data to a well-designed Array-List Collection. Editing is provided after data are stored. Using linear to search if a certain target is in the dataset. Afterwards users could also choose either bubble sort, heap sort or merge sort algorithm to sort their data. If the user wants to use BST search, they could also choose it after data being sorted. To ensure the best performance, the BST search will convert your date to a balanced binary search tree and then execute the searching process. For user's convenient, system will report the timing of sort or search process.

Requirements

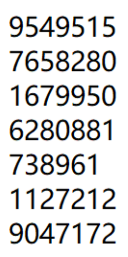
Computer requirements:

Computers with java compiler and java version 1.8.0 and up

Data file format:

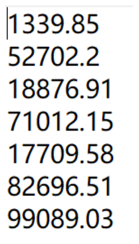
Your data should be stored in a txt file and every signal datum show be in a single line. String, integer or float are supported. Data format in a signal file should be the same type.

A typical data file format example (No file size limit):



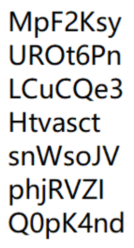
```
9549515
7658280
1679950
6280881
738961
1127212
9047172
```

Integer data file



```
1339.85
52702.2
18876.91
71012.15
17709.58
82696.51
99089.03
```

Float data file



```
MpF2Ksy
UROt6Pn
LCuCQe3
Htvasct
snWsoJV
phjRVZI
Q0pK4nd
```

String data file

Getting Started

The usage is simply following this Five steps:

<u>Entering data file → Editing the data → Linear search → Sort data → BST search</u>

You will allow to skip editing or any searching steps.

[Reminder] Serial numbers in this program start with 1. You may see system show some processing info use index number that different to your serial number, ignore it.

1. Entering data file

At the beginning of the program, you will be asked for entering datafile path that you want to deal with:

```
Please type datafile path:
D:\\TestCases/data_int_7digits_500.txt
```

After entering the path like above, hit the enter key.

Then enter corresponded number of the data type you are using. If your data is integer, then type 2 on your keyboard and press enter.

```
Please select data type: 1 for String, 2 for Integer, 3 for Float
Please type 1 or 2 or 3, hit enter to finish
2
```

And then check your data size and enter it. For data file contains 500 data, enter 500:

```
Please type data size: (Any integer number)
500
```

Then you should see the reminder that tells you your data have been stored and you can check your whole data list like this:

```
Data have been stored.
Check the data below:

Total number of the collection: 500
No.1 Element: 675531
No.2 Element: 7899021
No.3 Element: 341504
No.4 Element: 5362355
No.5 Element: 495333
No.6 Element: 492186
No.7 Element: 1948968
No.8 Element: 2552679
No.9 Element: 2144849
No.10 Element: 7309956
No.11 Element: 2237129
No.12 Element: 7624365
No.13 Element: 3224683
No.14 Element: 7976225
No.15 Element: 3204345
No.16 Element: 332776
```

……

```
No.496 Element: 8267379
No.497 Element: 9294837
No.498 Element: 2972012
No.499 Element: 7543783
No.500 Element: 4266311
```

You may notice there's a data type format reminder in some like (String Format). It's the same type as your input data.

2. Editing your data

If you want to make some edit after data checking, you will see the option right below your data list.

Add the data

```
Do you want to update your data: type 1 for add, type 2 for delete, type 3 for adjust value, type 0 to skip
1
```

To add your data, type 1 in the window and hit the enter.

This program provides you 3 different ways to add your data:

Add to the Front

Add it to the front and all data move forward for one place. To add your datum in the front, type 1 and hit enter at this window:

```
Where do you want to add: type 1 for Front, 2 for Middle, 3 for Back.
1
```

Then you will be asked to enter your datum. The data type must match the data stored in the list. You will have a format requirement reminder. If you want to add 1000, then input 1000 and hit the enter:

```
Please enter your datum (int Format):
1000
```

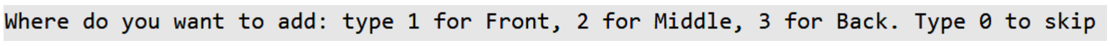
You may see some processing info and you can check your whole data list followed by: Current Data:

```
Capacity reached.  Increasing storage...
New capacity is 1000 elements
Inserting element at index 0
Compacting storage
Current Data:
No.1 Element: 1000
No.2 Element: 675531
No.3 Element: 7899021
No.4 Element: 341504
No.5 Element: 5362355
```

Scroll up and down to check your data list if necessary.

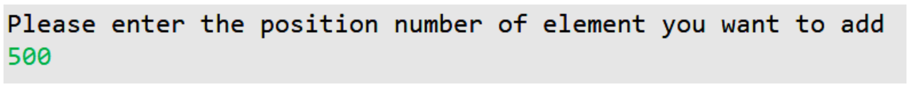
Add in the Middle

If you want to add it to the middle, type 2 when you see this:



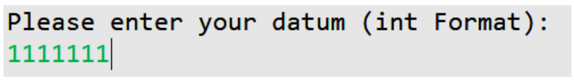
```
Where do you want to add: type 1 for Front, 2 for Middle, 3 for Back. Type 0 to skip
```

Then you will need to appoint the position to add it. All data starting from that position will move forward for one place.



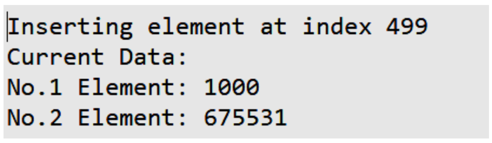
```
Please enter the position number of element you want to add
500
```

If you want to add your new datum to No. 500, input 500 above and enter your datum below:



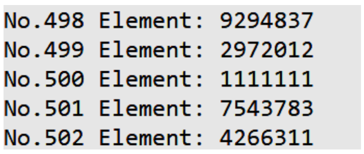
```
Please enter your datum (int Format):
1111111
```

After hit the enter key, you will see your current data list:



```
Inserting element at index 499
Current Data:
No.1 Element: 1000
No.2 Element: 675531
```

……



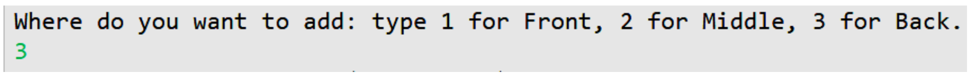
```
No.498 Element: 9294837
No.499 Element: 2972012
No.500 Element: 1111111
No.501 Element: 7543783
No.502 Element: 4266311
```

No.500 now is the datum you added, former No.500 moved to 501 and No.501 moved to 502.

Scroll up and down to check your data list if necessary.

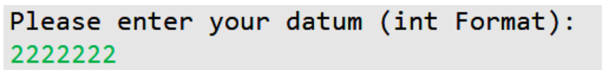
Add to the Back

For add your data to the back of the data list, type 3 and hit the enter in this window:



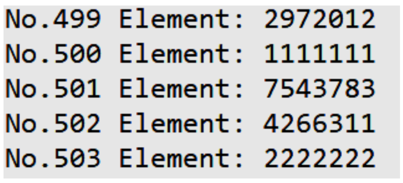
```
Where do you want to add: type 1 for Front, 2 for Middle, 3 for Back.
3
```

System will ask you to input your datum, and it will be added to the end of the data list without change any other data.



```
Please enter your datum (int Format):
2222222
```

You will see your whole current data list after processing:

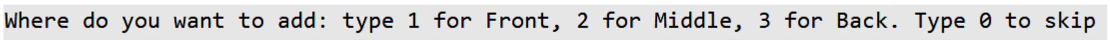


```
No.499 Element: 2972012
No.500 Element: 1111111
No.501 Element: 7543783
No.502 Element: 4266311
No.503 Element: 2222222
```

The datum 2222222 was successfully added to the end of your data list.

Scroll up and down to check your data list if necessary.

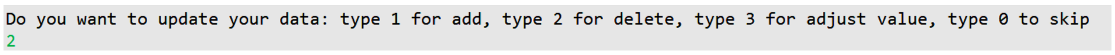
After each add operation, you will be asked again for where you want to add your next data. For adding more data, choose the place you want to add. You may enter 0 to skip the adding process:



```
Where do you want to add: type 1 for Front, 2 for Middle, 3 for Back. Type 0 to skip
```

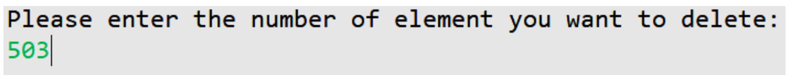
Delete the data

For delete a data, type 2 if you see this window:



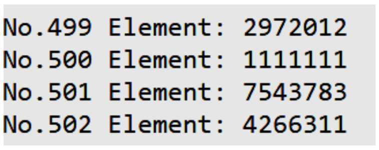
```
Do you want to update your data: type 1 for add, type 2 for delete, type 3 for adjust value, type 0 to skip
2
```

System will ask you which data you want to delete, enter its position number and hit the enter:



```
Please enter the number of element you want to delete:
503
```

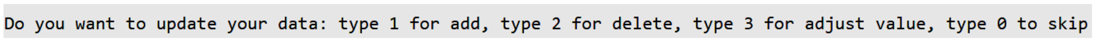
You will see your whole current data list after processing:



```
No.499 Element: 2972012
No.500 Element: 1111111
No.501 Element: 7543783
No.502 Element: 4266311
```

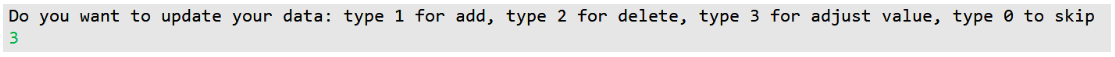
You assigned datum will be deleted and all the following data will move back for one place. If you have a data list contains 4 elements, and you deleted the second datum, the third one will be new second and forth one will be the new third.

After your deleting process, system will check if you want to edit another data, choose accordingly:



```
Do you want to update your data: type 1 for add, type 2 for delete, type 3 for adjust value, type 0 to skip
```

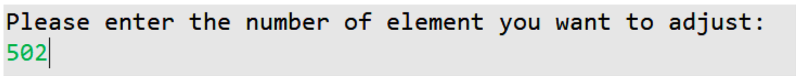
Adjust the data



```
Do you want to update your data: type 1 for add, type 2 for delete, type 3 for adjust value, type 0 to skip
3
```

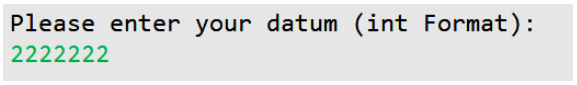
To adjust your data, type 3 in this window and hit the enter.

Then you will need to appoint the position to adjust your target data. All the other data will remain the same.



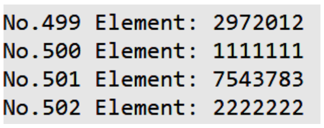
```
Please enter the number of element you want to adjust:
502
```

Then you need to type the new datum:



```
Please enter your datum (int Format):
2222222
```

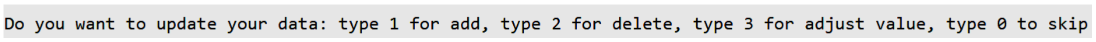
After hit the enter key, you will see your whole current data list:



```
No.499 Element: 2972012
No.500 Element: 1111111
No.501 Element: 7543783
No.502 Element: 2222222
```

The No.502 has been changed to 222222.

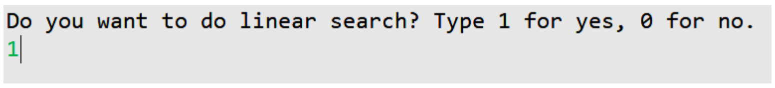
After your deleting process, system will check if you want to edit another data, choose accordingly. If you are done with editing, type 0 to skip editing.



```
Do you want to update your data: type 1 for add, type 2 for delete, type 3 for adjust value, type 0 to skip
```

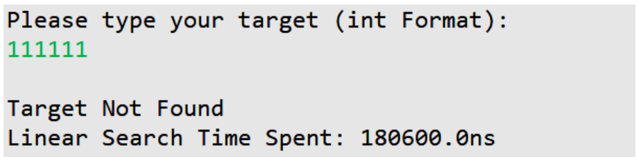
3. Linear Search

After data editing, you may do linear search with your current data list by enter 1 and press enter. You may also skip this part by input 0.



```
Do you want to do linear search? Type 1 for yes, 0 for no.
1
```

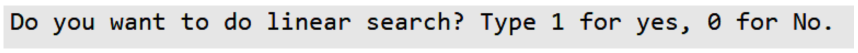
To search your target, enter in the following window and system will report if it’s existed in your data list or not to you. The searching time will also appear here.



```
Please type your target (int Format):
111111

Target Not Found
Linear Search Time Spent: 180600.0ns
```

You may search again by entering 1 or exit linear search by entering 0 in this window:



```
Do you want to do linear search? Type 1 for yes, 0 for No.
```

4. Sorting Data

For data sorting part, this program provides three different algorithms for you to choose. Type 1 for Bubble Sort, 2 for Heap Sort and 3 for Merge Sort.

```
Please select sort method: 1 for Bubble sort, 2 for Heap sort, 3 for Merge sort
1
```

After you hit the enter, the sorting process will start. When it's done, you will see "Bubble sort done." The time this sorting process spent and your data list after sorting.

```
Bubble sort done.

Time Spent: 1.39535E7ns
Data After Sorting:
No.1 Element: 1000
No.2 Element: 24525
No.3 Element: 38819
No.4 Element: 42534
No.5 Element: 47811
```

Same operation for heap sort and merge sort.

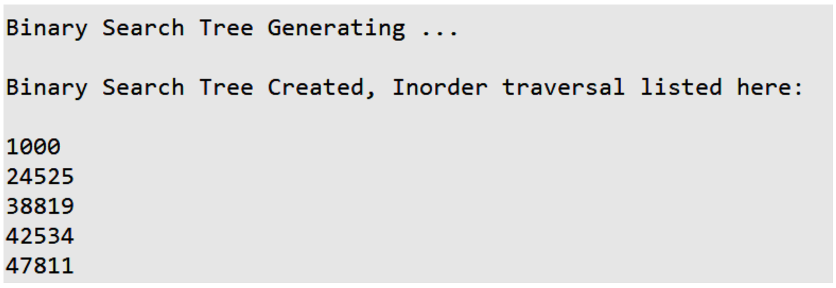
Notice, you may sort your data list only for one time. If you want to try another sorting method, you may start over the who program. You may NOT skip the sorting process.

5. BST Search

Once you get the sorted list, you may have choice to do BST search by enter 1.

```
Do you want to do BST search? Type 1 for yes, 2 for no.
1
```

For BST search, the program will automatically convert your sorted data list to a Balanced Binary Search Tree for the best searching performance. You will see the following remainder below. After the tree is created, system will print the inorder traversal of this tree. This traversal should be listed as same as your sorted list. To check your data, you can compare it with your sorted list.

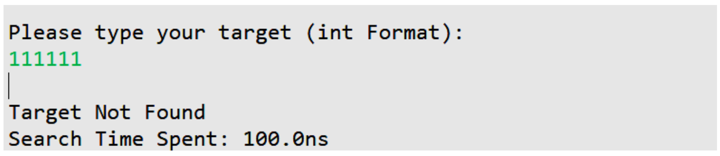


```
Binary Search Tree Generating ...

Binary Search Tree Created, Inorder traversal listed here:

1000
24525
38819
42534
47811
```

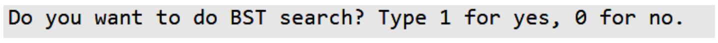
After this you will be asked to type the target in the window. Type it in and you will receive the message if it’s existed in your data list or not, as well as the time spent for this searching process.



```
Please type your target (int Format):
111111

Target Not Found
Search Time Spent: 100.0ns
```

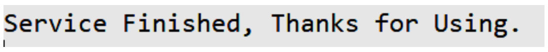
You will be asked for choosing if you want to search again or not. Choose accordingly.



```
Do you want to do BST search? Type 1 for yes, 0 for no.
```

6. End

After all the steps or part of them above, you will see this message:



```
Service Finished, Thanks for Using.
```

This marks the program has done all the job asked. If you want to use it again, you may start running this program again.

Testing Documents

To test the program, the Data Generation program was created to generate random data for certain size (see Data Generation folder in source codes). It contains three different algorithms to generate either string, integer or float data. The single datum length and the data file size could be set manually. The size was set to 50, 500, 1001 and 2000 for each data type. All datum length was set to up to 7 digits. 4 (different sizes) × 3 (data types) = 12 files was created (check TestCases folder in source code). They were used for unit testing and final testing. It shows that the final version of this program runs well.

Debug Notes

1. int del in main function is same as index of array list but different with serial number that users use. To fit the user's logic, adjust it by minus 1.
2. In resort class, new BST created with multiple same nodes. This pointed the algorithm is somehow wrong. After checking logic, it wasn't wrong. After rewrite the code and implement to a test case, and compared the code with the original code, it turns out an intermediate variable was defined outside the recursive function. Moved it to the middle of recursive part solved the issue.
3. The main function logic wasn't perfect. To allow user to do multiple editing or searching, another logical code was added, and it runs perfect afterwards.
4. The timing was using ms as unit and in the real testing process, it could not show the processing time since the actual running time is too fast. Change to System.nanotime() solved the problem and the new timing unit change to ns.

Feature Improvement

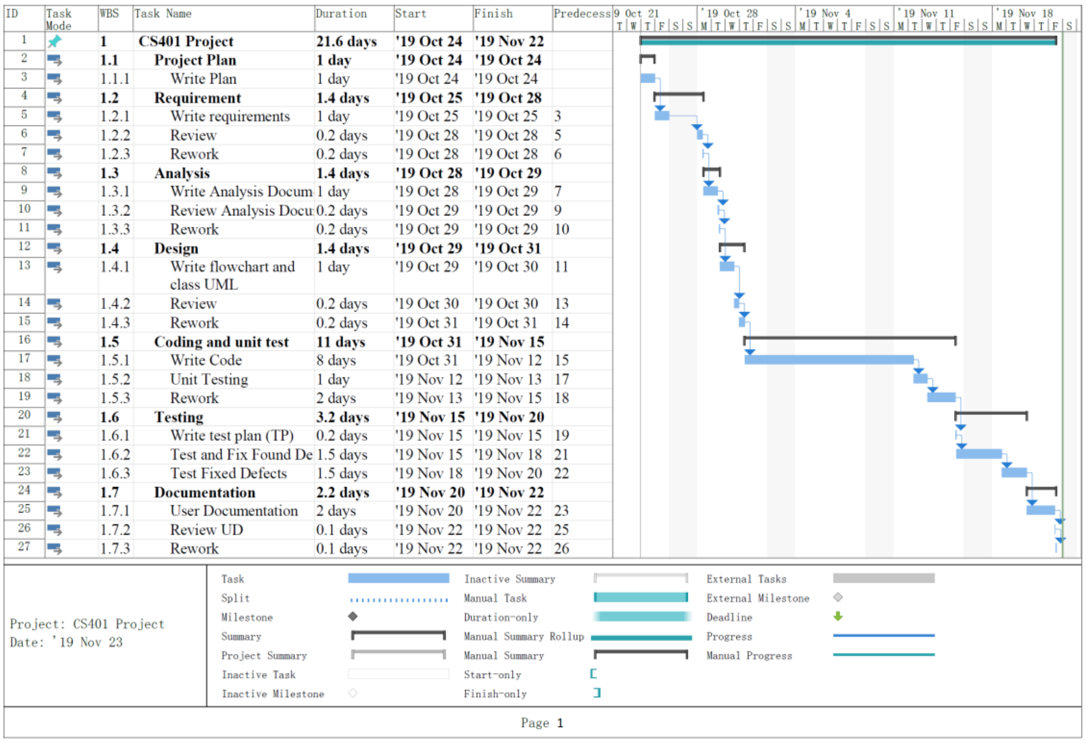
✧ The interface of ALCollection also supports get function and more functions are supported by BST Interface. To improve user experience, those function could be written in the program and provide more options for user to use.

- ✧ Graphic interface may be developed in the feature to make operations easier to read and operate.
- ✧ The invalid file path input may cause program termination and it will require user to restart the program. Even it's the first thing of running the program, it's still a better idea to allow user re-input the path instead of re-start the program.
- ✧ Auto recognition of data type and data size may add to enhance user experience in the feature.

Project Management

The project was planned at the time of project released. The whole plan based on software development life cycle and personal experience.

To balance the developer's other courses and tasks, ensure the best performance of finishing the task and leave enough slack to finish the project, the schedule was set as following chart:



ID	Task Mode	WBS	Task Name	Duration	Start	Finish	Predecess
1	📌	**1**	**CS401 Project**	**21.6 days**	**'19 Oct 24**	**'19 Nov 22**	
2	➡	**1.1**	**Project Plan**	**1 day**	**'19 Oct 24**	**'19 Oct 24**	
3	➡	1.1.1	Write Plan	1 day	'19 Oct 24	'19 Oct 24	
4	➡	**1.2**	**Requirement**	**1.4 days**	**'19 Oct 25**	**'19 Oct 28**	
5	➡	1.2.1	Write requirements	1 day	'19 Oct 25	'19 Oct 25	3
6	➡	1.2.2	Review	0.2 days	'19 Oct 28	'19 Oct 28	5
7	➡	1.2.3	Rework	0.2 days	'19 Oct 28	'19 Oct 28	6
8	➡	**1.3**	**Analysis**	**1.4 days**	**'19 Oct 28**	**'19 Oct 29**	
9	➡	1.3.1	Write Analysis Docum	1 day	'19 Oct 28	'19 Oct 29	7
10	➡	1.3.2	Review Analysis Docu	0.2 days	'19 Oct 29	'19 Oct 29	9
11	➡	1.3.3	Rework	0.2 days	'19 Oct 29	'19 Oct 29	10
12	➡	**1.4**	**Design**	**1.4 days**	**'19 Oct 29**	**'19 Oct 31**	
13	➡	1.4.1	Write flowchart and class UML	1 day	'19 Oct 29	'19 Oct 30	11
14	➡	1.4.2	Review	0.2 days	'19 Oct 30	'19 Oct 30	13
15	➡	1.4.3	Rework	0.2 days	'19 Oct 31	'19 Oct 31	14
16	➡	**1.5**	**Coding and unit test**	**11 days**	**'19 Oct 31**	**'19 Nov 15**	
17	➡	1.5.1	Write Code	8 days	'19 Oct 31	'19 Nov 12	15
18	➡	1.5.2	Unit Testing	1 day	'19 Nov 12	'19 Nov 13	17
19	➡	1.5.3	Rework	2 days	'19 Nov 13	'19 Nov 15	18
20	➡	**1.6**	**Testing**	**3.2 days**	**'19 Nov 15**	**'19 Nov 20**	
21	➡	1.6.1	Write test plan (TP)	0.2 days	'19 Nov 15	'19 Nov 15	19
22	➡	1.6.2	Test and Fix Found De	1.5 days	'19 Nov 15	'19 Nov 18	21
23	➡	1.6.3	Test Fixed Defects	1.5 days	'19 Nov 18	'19 Nov 20	22
24	➡	**1.7**	**Documentation**	**2.2 days**	**'19 Nov 20**	**'19 Nov 22**	
25	➡	1.7.1	User Documentation	2 days	'19 Nov 20	'19 Nov 22	23
26	➡	1.7.2	Review UD	0.1 days	'19 Nov 22	'19 Nov 22	25
27	➡	1.7.3	Rework	0.1 days	'19 Nov 22	'19 Nov 22	26

Project: CS401 Project
Date: '19 Nov 23

In this schedule, no rigid hourly plan was created. Since the developer is the only one who's working on the project, all tasks should get done one by one. It didn't put weekends to the plan and weekends counts on slack in practice. Check appendix for the high-resolution figure.

Complexity Analysis

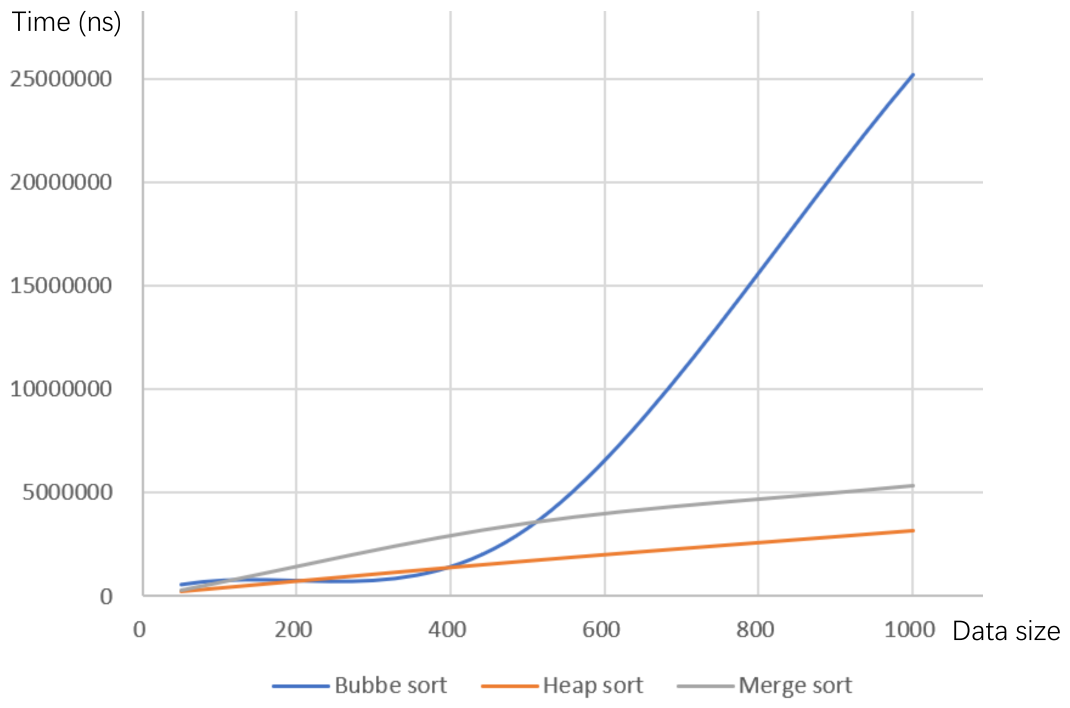
Base on the algorithm, the complexity shows in the table below:

	Best	Worst
Bubble Sort	O(n)	O(n^2)
Heap Sort	O($n\log_2 n$)	O($n\log_2 n$)
Merge Sort	O($n\log_2 n$)	O($n\log_2 n$)
Linear Search	O(1)	O(n)
BST Search	O(1)	O(log n)

The worst case is more meaningful in real life. The program was running for all cases and for both data types. Since all data types show similar tendency, here the integer data files are used for this table:

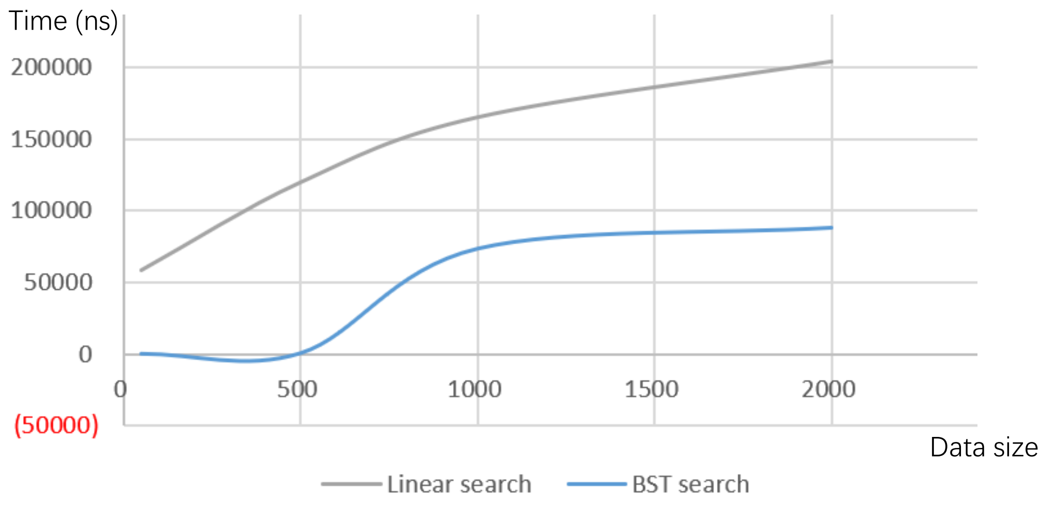
Data size (n) / Algorithm	50	500	1001	2000
Bubble sort	555000	3270500	25170500	50279500
Heap sort	263600	1723700	3174400	3056800
Merge sort	303700	3521200	5320300	7756100
Linear search	58500	119900	165400	204300
BST search	100	500	73600	88200

There are several cases need to be mentioned. To test the worst case for linear search, a data that does not exist was used in this case, and for BST search, a leaf node located in the middle was used. The system running time for exactly same situation are vary. It may be affected by a lot of different factors. For instance, the same search will be a lot faster than the same search for the first time. The time here was selected the first time running of each algorithm. To show the tendency, these data were made to two graphs below, one is for sorting comparison and the other is for searching comparison.



Sorting algorithms comparison

* To have a better view, n = 2000 wasn't selected in this graph



Searching algorithms comparison

These two graphs show basically the same tendency that complexity analysis in the beginning of this part. Since points used here are very limited, curves may not that accurate.