Sort Manager

Created by Thomas Kirkwood

The sort manager project was assigned by Sparta Global trainer Astha, the minimum value product (MVP) of this project was for the implementation of 3 different sort methods according to the method signatures of given interfaces.

The tools and principles used in the design and implementation of the project are as follows:

OOP and SOLID Were used to guide the implementation of code.

TTD was used to ensure iteration of workable code that builds towards meeting the full MVP

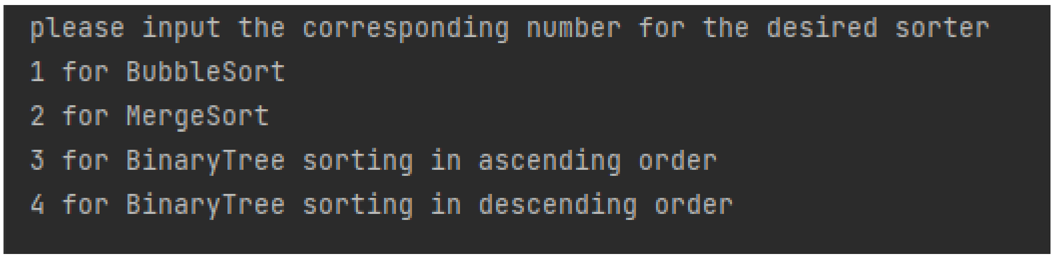
Jutil was used to test program functionality

Intellij was used as the IDE to write the code.

GitHub was used for version control and to backup code already done.

User guide to use

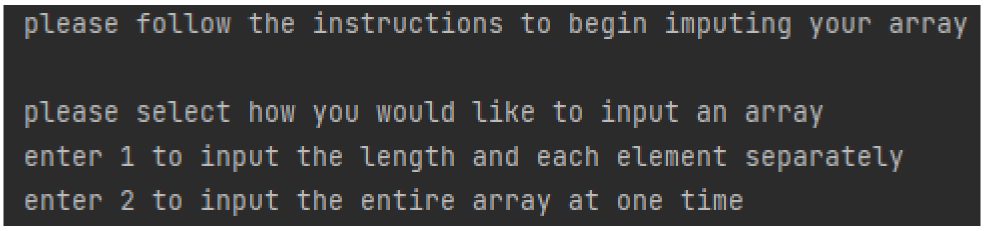
upon running the program, the console will output instructions, such as in the below picture, on what to do.



```
please input the corresponding number for the desired sorter
1 for BubbleSort
2 for MergeSort
3 for BinaryTree sorting in ascending order
4 for BinaryTree sorting in descending order
```

Once a choice is made between the 4 types of sorting that can be used the console will then output what sorter was selected.

The program will then ask how the user would like to enter in the numbers to be sorted and guide the user through the process.



```
please follow the instructions to begin imputing your array

please select how you would like to input an array
enter 1 to input the length and each element separately
enter 2 to input the entire array at one time
```

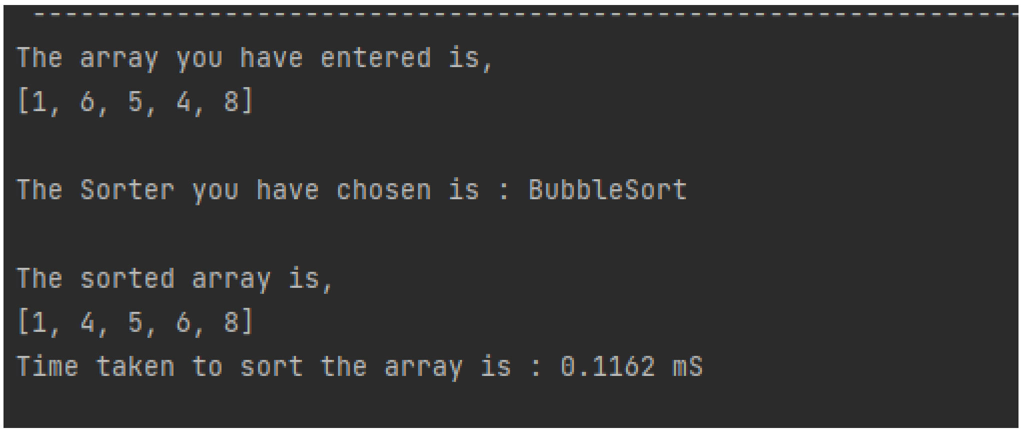
Once the user has made the selection the program will take one of two input methods possible.

if the user inputs something that is not a number or inputs a number that is not a 1 or 2 the program will then prompt the user to input a number within range.

When selecting 1 the user will then be prompted to input a number for the length of the array they wish to create to be sorted. After the length, the user will be prompted to enter each number individually. Once that is done the program then moves on to sort the array with the selected sort method.

When selecting 2 the user will be prompted to enter all the numbers they want sorted at one time separated by a space, the program will remove any non-number characters and sort the numbers into an array. Once that is done the program then moves on to sort the array with the selected sort method.

After the array has been created and the sort method selected the console will then output the array to be sorted, the sort method that will be used, the sorted array and the time taken to sort the array in milliseconds.



```
 ----------------------------------------------------------
The array you have entered is,
[1, 6, 5, 4, 8]

The Sorter you have chosen is : BubbleSort

The sorted array is,
[1, 4, 5, 6, 8]
Time taken to sort the array is : 0.1162 mS
```

Callable methods

The program has several callable methods these include the main methods that are required by the MVP.

Firstly the sorter interface has one method that has been implemented into each of the different sorters, the sortArray method that takes in a int array called arrayToSort.

This method is implemented in such a way that when it is called it will sort the array differently depending on the class it is in, i.e. using the bubble sort method in the BubbleSort class.

The BinaryTree interface has several methods that the binary tree sorter has to implement as well as the sorter interface.

The getRootElement method will simply return the value of the root element.

The getNumberOfElements method will return the number of elements currently being stored in the binary tree.

The addElement method will take in an int value and attempt to add it to the tree

The addElements method will take in an in array and run addElement for each value in the array.

The findElement method will take in an int value and will check the binary tree and return true if the value if in the array or else return false if it is not.

The getLeftChild method will take in an int and return the value of the left child of that int value or if there is no left child will throw the exception ChildNotFoundException.

The getRightChild method will take in an int and return the value of the right child of that int value or if there is no right child will throw the exception ChildNotFoundException.

The getSortedTreeAsc method will return an array of all the values stored in the tree in ascending order.

The getSortedTreeDesc method will return an array of all the values stored in the tree in descending order.

Testing guide

Testing was done using jutil to test methods and overall functionality. Some manual testing was required for methods that took input from a user or whose return type was void and there was no reasonable way to alter code for automatic testing.

Bubble Sort

Bubble sort only has one method to test, as such only 3 tests where ran.

1. functionalityBubbleTest, utilises the assert functionality of Jutil I was able to use a unsorted pre-determined array and the sorted version of that array to ensure that the bubble sort was sorting correctly.
2. duplicateBubbleTest, using a array with duplicate values ensures that duplicate values do not interfere with the normal workings of the sorter.
3. alreadySortedBubbleTest, checks that when a array that is already sorted is used as a input the sorter will not needlessly try to sort it and instead output that it was already sorted.

Merge sort

Merge sort has 2 testable methods, the first one being the method described by the interface and the second is one is private.

1. mergeMergeSortTest, for this test the normally private method had to be made public in order for the test to access it, once the method was public 2 pre-determined array where given to the merge method ad tested against the expected output, after successful testing the method had to be made private again
2. mergeSortTest, for this test, a single array was given to the sortArray method and tested against the expected sorted output.
3. DuplicateSortTest, tests the methods ability to deal with arrays with duplicate values.
4. preSortedMergeSortTest, tests the response to a pre sorted array being used as a input, will output a message in console when it detects a pre-sorted array.

Binary Tree

The binary tree implements 2 interfaces and has many more methods than the other sorts as such required more testing, further TTD was used in the implementation of binary tree.

1. getRootElementBinaryTreeTest, this test was for the method to get the root node element, a constructor will take in a value and set a node up for that value and use it as the root node, the method then calls whatever has been set as root and checks to see if it matches the value we inputted.
2. findElementTest, for this test a series of values is inputted into a binary tree and then this method is called using one of those values, it will return true if it finds the element and false if it doesn't .
3. getLeftChildExceptionTest, and getRightChildExceptionTest, these tests checks that the methods are throwing exceptions correctly when there is no left or right child nodes. Both tests setup a binary tree with one node and then attempt to get the left or right child of that node and should throw the MissingChildNode exception.
4. getLeftChildTest, and getRightChildTest, these tests will check the correct functionality when there is a left or right child node. Both tests setup a root node of 10 and enter in the same array of numbers putting some at the left and some at the right of the root node, the methods are then called and return the values of the left and right node respectively.
5. getSortedTreeAscTest, and getSortedTreeDescTest, both tests will create the same binary tree and will add the same elements, for the ascending test the output should be in ascending order and for the descending test the output will be in descending order.

Manual tests

As many of the methods in the util package take input from the user or simply print to the console or elsewhere.

The methods that required manual testing where all three methods from the CreateIntArray class, selectIntCreationMethod, createIntArray, differentCreateIntArray. For these I simply printed out the array just before the return statement and compared it to what I had entered, this also let me test the selectIntCreationMethod, as it selects one of the others to use.

the scan class has one method that required manual testing, nextInt, this method will only accept int values and will continually prompt the user to input a int whenever something other than an int is inputted. For this several int and non-int values where entered one at a time and the output checked against the expected output.

The last two methods I tested manually where the getSorterType and the sorterFactory, for both of these I had them print out the retuned variable to ensure that what was being returned was the sort type and sorter that would be chosen.

There are several methods I chose not to explicitly test as they where only there to provide an additional level of abstraction and only implemented a single methods, like the print method will only run the System.out.println method.

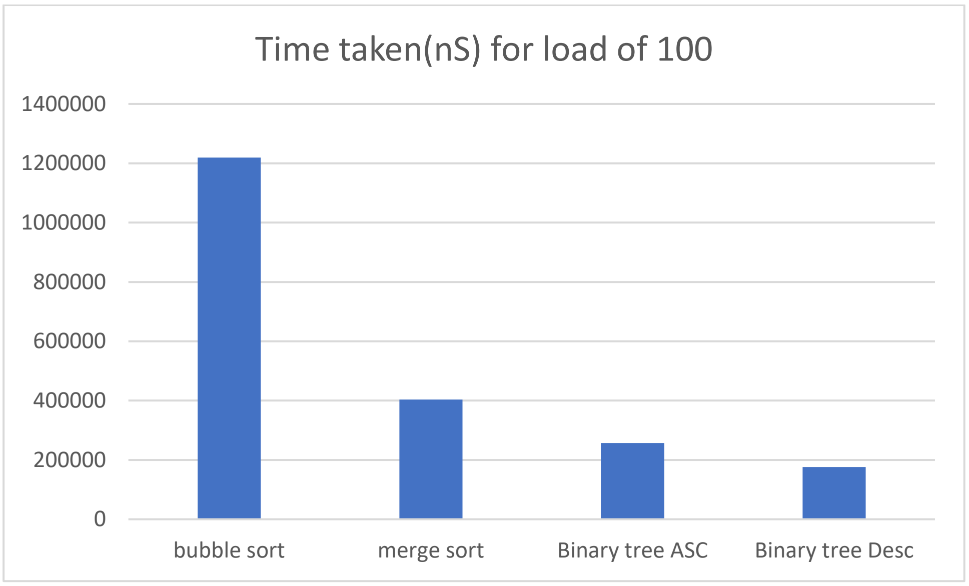
Performance testing

For the performance testing a randomly generated array is used of increasing length on all the sorters and the time taken for each recorded and averaged over 5 different random arrays to get a better idea of the general time taken for each sorter for a given load.

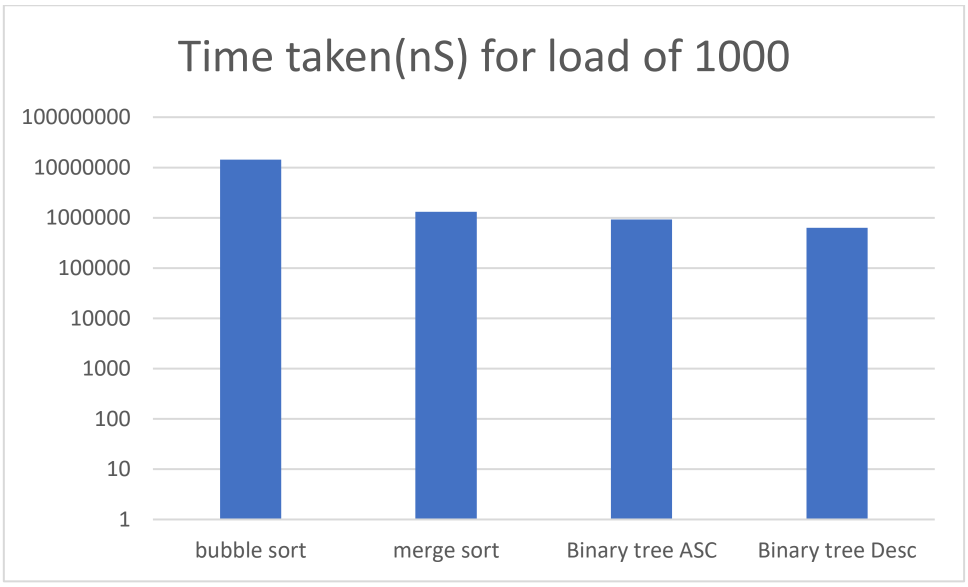
The below table give the results of the performance test for the given load values. The table has the time in nano seconds to better display the differences in time taken.

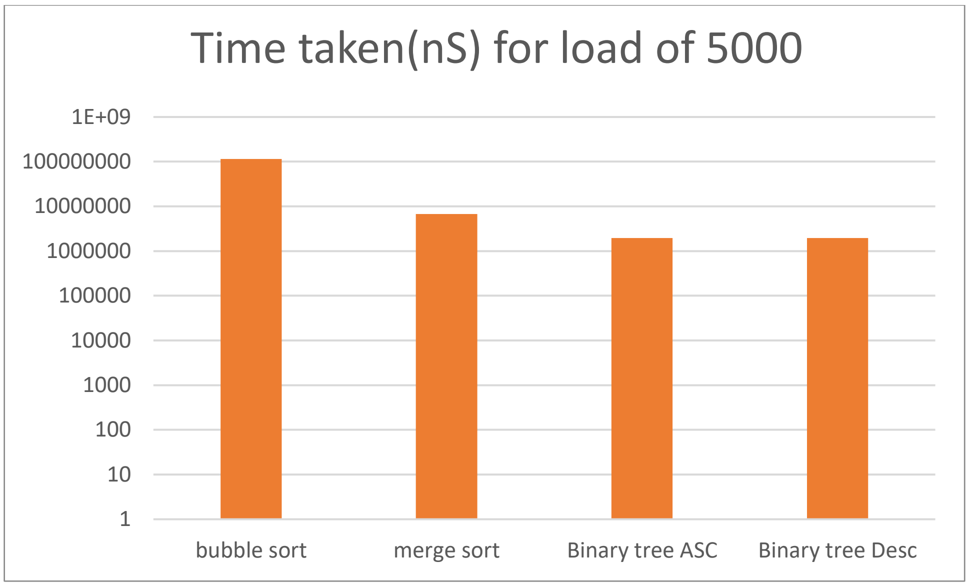
	average time in nanoseconds to complete each sorter operation			
Merge sort				
load	bubble sort	merge sort	Binary tree ASC	Binary tree Desc
100	1219120	403960	257180	176460
1000	14232960	1312860	930000	631660
5000	114484880	6786720	1943440	1951020
10000	433836280	4935760	2890020	2152980
50000	7860128900	41406100	6378280	5626000

The below tables offer a visual representation for the time taken for each sorter for each load value.

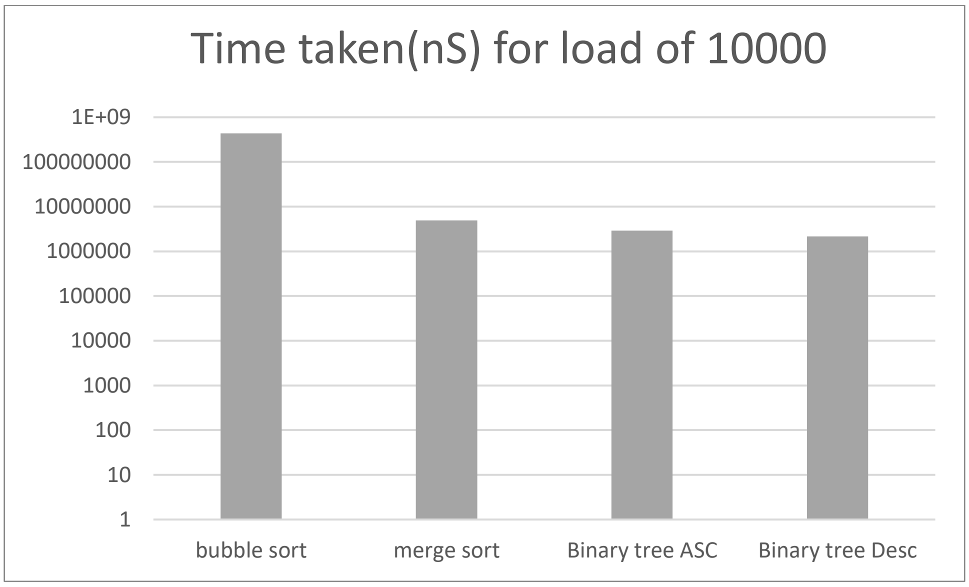


For the following charts, a logarithmic scale is used to get a better view of the smaller times taken

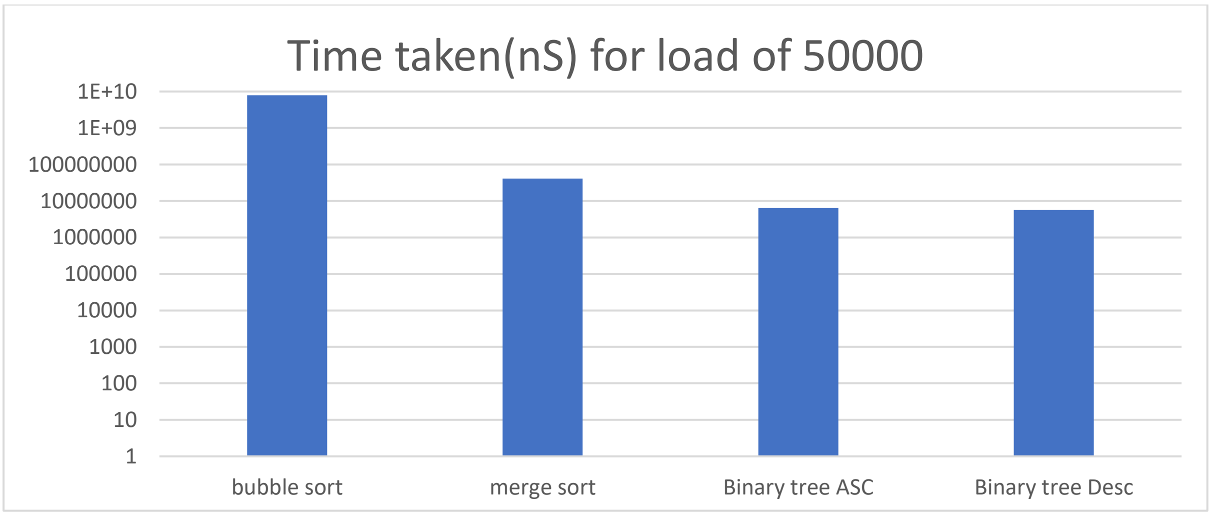




Time taken(nS) for load of 5000
1E+09
100000000
10000000
1000000
100000
10000
1000
100
10
1
bubble sort
merge sort
Binary tree ASC
Binary tree Desc



Time taken(nS) for load of 10000
1E+09
100000000
10000000
1000000
100000
10000
1000
100
10
1
bubble sort
merge sort
Binary tree ASC
Binary tree Desc



Time taken(nS) for load of 50000
1E+10
1E+09
100000000
10000000
1000000
100000
10000
1000
100
10
1
bubble sort
merge sort
Binary tree ASC
Binary tree Desc