CS401 - Final Project

Problem Specification

I. The project compares the complexities of sorting algorithms and searching algorithms.
The user is allowed to choose two sorting algorithms between one simple sorting algorithm (selection sort, insertion sort, bubble sort) and O(Nlog2N) sorts (Quick sort, Merge Sort, Heap sort). After the array is sorted, the count numbers of comparisons in the algorithm are printed so we can conclude which algorithm performs better with the total counts. The program's result is discussed with the theory of the algorithm you selected with the Big O.

II. linear search with the original list and BST with the sorted list is performed.
The hash function list is created with the same data and checks for the complexities to compare with linear searching, binary searching, and hash function searching.

Software specification

Methods provided within the ProjectMain class are :

i. selectionSort
ii. swap
iii. insertionSort
iv. insertElement
v. BubbleSort
vi. bubbleUp
vii. mergeSort
viii. Merge
ix. Quicksort
x. split
xi. HeapSort
xii. buildMaxHeap
xiii. heapify
xiv. linearSearch
xv. sortedtoBST
xvi. Binarysearch
xvii. hashSearch

Pseudo code

1. Selection sort

```
selectionSort(array)
 repeat (array size - 1) times
 set the first unsorted element as the minimum
 for each of the unsorted elements
 if element < current minimum
 set element as new minimum
 swap minimum with the first unsorted position
 end selectionSort
```

2. Insertion sort

```
insertionSort(array)
start from the second element
for each unsorted element
for every element(lastSortedIndex) down to 0
```

```
if current element <  current minus one
swap the elements
break the inner loop when all the left side of the array is sorted
break the last loop when it has traversed till the end of the array
end insertionSort
```

3. **Bubble sort**

```
BubbleSort(array)
for all elements of list
if array[i]>array[i+1]
swap the elements
end BubbleSort
```

4. **Merge sort**

```
mergeSort(array,)
declare array left right and middle value
perform merge function.
if left > right
return
mid= (left+right)/2
mergesort(array, left, mid)
mergesort(array, mid+1, right)
merge(array, left, mid, right)
end mergeSort
```

5. **Quick sort**

```
quickSort(array, start, end)
if (start < end)
splitpoint = split(arr,start, end)
quickSort(arr, start, splitpoint)
quickSort(arr, splitpoint + 1, end)
split(arr, start, end)
set end as splitpoint
Index = start - 1
for i = start to end-1
if arr[i] < splitValue
swap arr[i] and arr[pIndex]
Index++
swap splitValue and arr[pIndex+1]
return pIndex + 1
```

6. **Heap Sort**

```
Heapsort(arr)
building the max heap to fetch the maximum value
Maximum value will be at root value
for(int i=arrayRef.length-1;  i>0;  i--) {
swap(&arr[0], &arr[i]);
maxHeapify(arr,0);
}
```

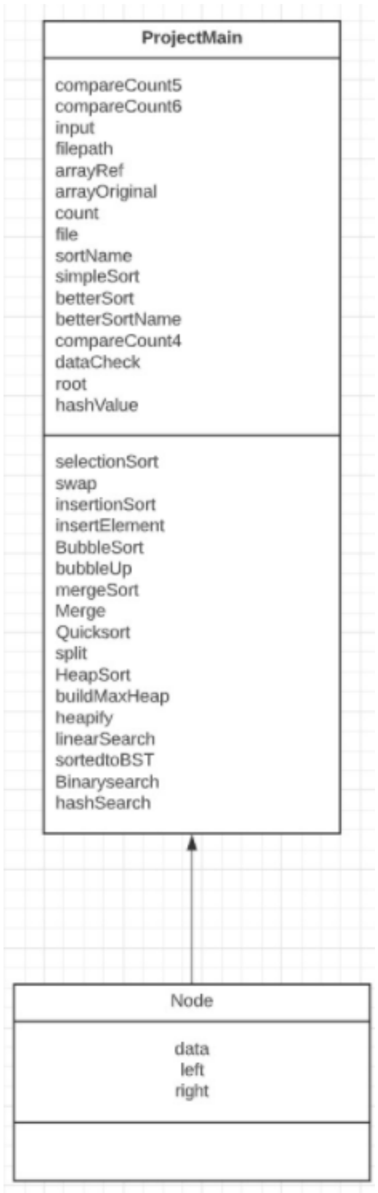
7. **Linear Search**

```
linearSearch(array, data)
for each value in the array
if match value == data
return the item's location
end if
end for
end linearSearch
```

8. Binary Search

```
binarySearchTree(array,target)
start with the root node
assign current value to root initially
if the current value is not null and current value is not target value
assign current value parent
if target < current data
change current value to left node
else change current value to right node
if current value is null then data is not present
if parent value is null then data is root node
if target is less than value then target is left node is the location
if target is more than value then target is right node is the location
end binarySearchTree
```

- **UML Diagram**



- **Testing document**
  data.txt file is provided along with this file which is the input data file

- **Project management/schedule**
  Utilized 2hrs daily for last entire week

- **Complexity analysis**
  We have taken the selection sort from the simple sorting approach and the merge sort from the binary sorting approach. Selection sorting has a complexity of O(N^2) and merge sorting has a complexity of O(nlogn).
  In the project, we have taken integer values in the data file which consists of 124 integer values.
  When the selection method runs for the 124 integer values, it makes a comparison 7626.
  Total Count of the selection sort=n(n-1)/2=(124*123)/2=7626.
  But when the same data is sorted using merge sort, the comparison count is 740.
  Therefore, we can conclude that merge sort is better than selection sort.
  Linear search has a complexity of O(N) and Binary search has a complexity of O(logN) and hash function has a complexity of O(N). Binary search tree approach is better than the other two search options