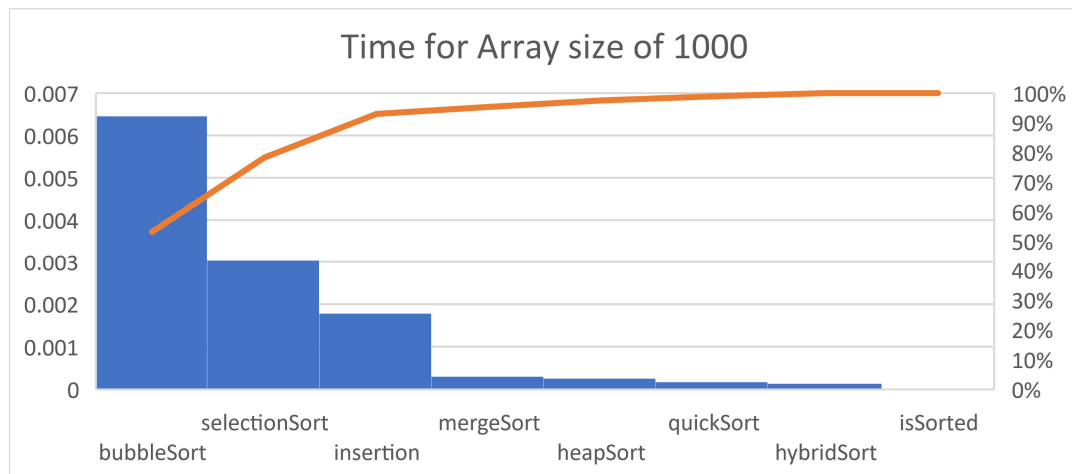## PA4

Nicholas Soliman

UIN: 326009195

1.)

 a. O(n), worst O(n^2). The isSorted() function was created by iterating through the array and making sure that the next term in the array is greater than the current. As long as this is true, then the whole array should be sorted properly
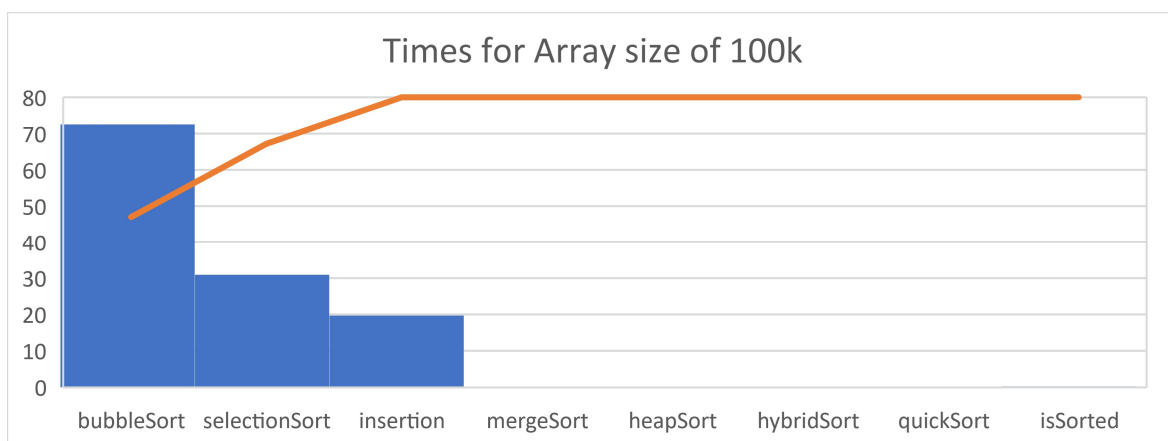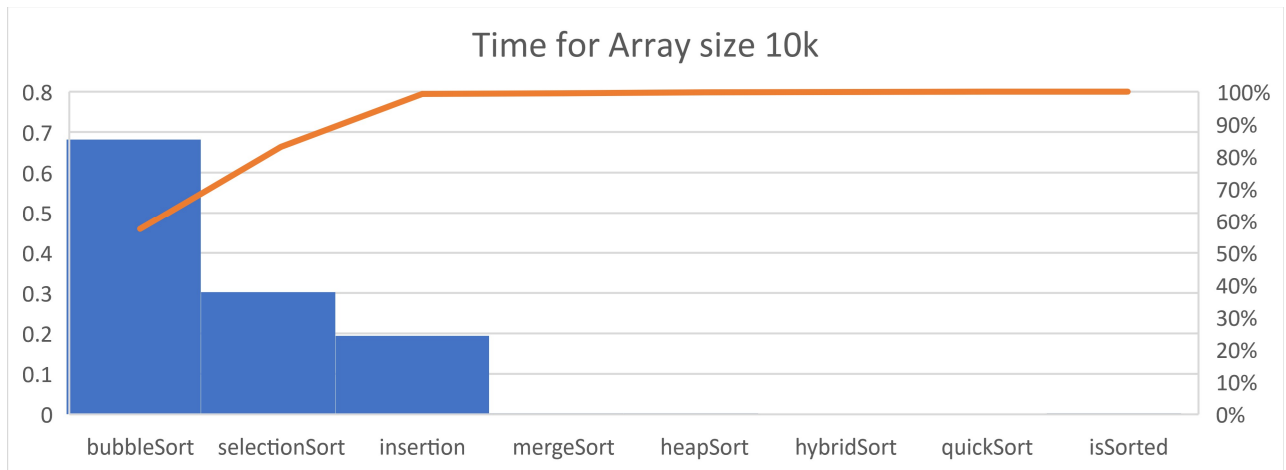
 b. O(n^2), best O(n). The bubble sort was implemented by using a nested for loop that goes through and compares every two values and swaps them if the first is greater than the second. It goes through the array multiple times until each of them are sorted properly.

 c. O(n^2). The selection sort method finds the minimum element in the array and then places it at the beginning of the array. Then it repeats this process but with one less array index.

 d. O(nlog(n)). For the hybrid sort function, I split up the array into four different parts and sorted each separately using the quicksort function. Then I merged all four of these sorted array together using the merge function.

 e.

Time for Array size of 1000

Time for Array size 10k

A bar and line (Pareto) chart. The left vertical axis is labeled from 0 to 0.8 in increments of 0.1. The right vertical axis is labeled in percentages from 0% to 100% in increments of 10%. The horizontal axis categories are: bubbleSort, selectionSort, insertion, mergeSort, heapSort, hybridSort, quickSort, isSorted. Bars appear for bubbleSort (~0.69), selectionSort (~0.31), and insertion (~0.2), with an orange cumulative line rising to 100%.

Times for Array size of 100k

A bar and line (Pareto) chart. The left vertical axis is labeled from 0 to 80 in increments of 10. The horizontal axis categories are: bubbleSort, selectionSort, insertion, mergeSort, heapSort, hybridSort, quickSort, isSorted. Bars appear for bubbleSort (~73), selectionSort (~31), and insertion (~20), with an orange cumulative line rising to 80.

1 million took too long to compile

2.)

   a. For this part we created 4 separate functions:

          insertKey(): This function inserted a new key 'k' into the array by finding the best location in the array where the key will fit. After finding the proper location for it, it inserts that element into the array.

          extractMin(): This function takes out the smallest element in the array which will always be arr[0] as long as the array is sorted. Then it replaces that element and removes it from the heap.

          decreaseKey(): This function decreased the value of key at the index to a different value, then moved it to where it belongs in the tree. Since the new element wont always fit in the same spot.

          deleteKey(): This function simply removed one of the key elements. This was implemented by setting the value to 0 at that index, then extracting the minimum value.

b.

```
                    1
            3               2
        6       7       5       4
      15 14   12  9   10  11  13  8
```
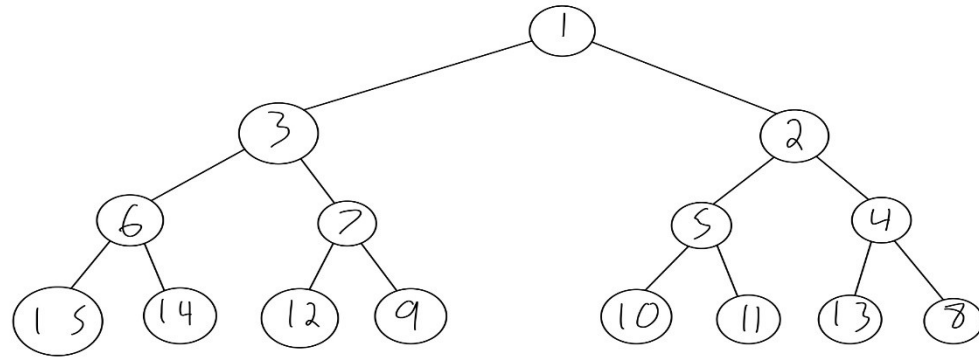
c. The kth smallest element function was implement by first creating a heap with all of the values in the array. Then after its been sorted, it will extract the minimum value multiple times (k − 1 times) then the next smallest value is the answer. So I use the getMin() function to return the answer.