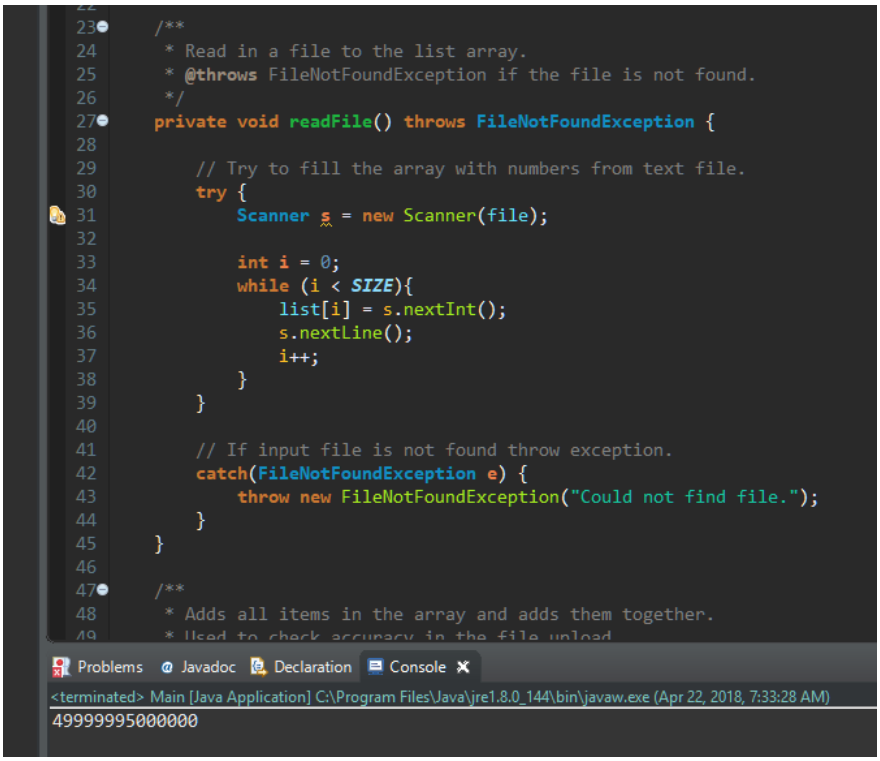
CS361 Algorithm Lab 1

What to do

1. As a refresher I looked over a post on stackoverflow (link superlong) on how to read text files since it has been a long time since I've done that. I set the while loop to use the SIZE value of the array instead of the scanner .hasNext function so that I could stop reading in lines just by adjusting the value in SIZE.



2. To get the mergesort working I used the pseudocode from the book in combination with the geeksforgeeks, Code n Learn, vogella, and code review. I struggled with getting this one working, I kept getting overflow errors and kept trying different examples until it outputted true so mostly just relying on that functioning correctly.

```java
/**
 * Main driver of the MergeSort method.
 * @param arr the array to be sorted.
 * @param l the left index.
 * @param r the right index.
 */
public void mergeSort(int arr[], int l, int r) {
    if(l < r) {
        // establish middle point.
        int m = (l+r)/2;

        // cut my life into pieces.
        mergeSort(arr, l, m);
        mergeSort(arr, m+1, r);

        // combine arrays.
        merge(arr, l, m, r);
    }
}
```

```java
/**
 * Merge the arrays back together, sorting them.
 * @param arr array to be merged
 * @param l starting index of array.
 * @param m middle index of array.
 * @param r end index of array.
 */
private void merge(int arr[], int l, int m, int r) {
    // find size of subarray's
    int li = m - l + 1;
    int ri = r - m;

    // create left and right arrays.
    int L[] = new int [li];
    int R[] = new int [ri];

    // copy data into new arrays.
    for (int i=0; i < li; ++i)
        L[i] = arr[l + i];
    for (int j=0; j < ri; ++j)
        R[j] = arr[m + 1 + j];

    // establish indexes
    int i = 0;
    int j = 0;
    int k = l;

    // begin merge
    while (i < li && j < ri)
    {
        if (L[i] <= R[j])
        {
            arr[k] = L[i];
            i++;
        }
        else
        {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    // copy any remaining elements of L[]
    while (i < li)
    {
        arr[k] = L[i];
        i++;
        k++;
    }

    // copy any remaining elements of R[]
    while (j < ri)
    {
        arr[k] = R[j];
        j++;
        k++;
    }
}
```

OUPUT (isSorted):

```
true
```

3. To get the quicksort working I used the pseudocode from the book in combination with the geeksforgeeks section on quicksort. Also used slideshows from here to figure out how to make the median work correctly. In my initial version the pivot would be incorrect and I would swap into the middle a larger number and it would not be in order.

```java
/**
 * Main driver of the quicksort.
 * @param arr the array to be sorted
 * @param p the first index location of an array to be sorted.
 * @param r the last index location of an array to be sorted.
 */
public void quickSort(int arr[], int p, int r){
    if (p < r){
        // Perform partitioning of array.
        int q = partition(arr, p, r);
        // Recursively run quicksort again.
        quickSort(arr, p, q-1);
        quickSort(arr, q+1, r);
    }
}
```

```
<terminated> Main [Java Application] C:\Program Fi
64
65
66
67
68
69
70
71
72
73
```

OUPUT (isSorted):

```
true
```

```java
private int partition(int arr[], int p, int r)
{
    // Find the middle of the array.
    int m = (p + r) / 2;

    //Swap the middle value with the last value.
    int value = arr[r];
    arr[r] = arr[m];
    arr[m] = value;

    // Initialize pivot with last value.
    int pivot = arr[r];

    int i = p-1;
    for (int j=p; j<r; j++)
    {
        // Test j against pivot
        if (arr[j] <= pivot)

        {
            i++;

            // Swap
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }

    // Swap pivot into "center"
    int temp = arr[i+1];
    arr[i+1] = arr[r];
    arr[r] = temp;

    // Return the "center" index number.
    return i+1;
}
```

4. Used this stackoverflow code as a reference to wrap my head around how to accomplish the check. Originally kept having a stack overflow error but after rereading the initial instructions I made the recursive call start from the halfway point instead of .length – 1.

```java
/**
 * Check if the array has been sorted.
 * @param arr the array to be checked.
 * @return true if the array is sorted, false if it is not.
 */
public boolean flgIsSorted(int arr[]){
    return checkSort(arr, arr.length);
}

/**
 * Recursively check the array to see if it is sorted.
 * @param arr the array to be checked.
 * @param x the starting point for checking.
 * @return
 */
public static boolean checkSort(int arr[], int n) {
      // If array is empty or length is less then 2 return true.
      if (arr == null || n < 2) {
        return true;
      }

      // Find the middle of the array.
      int m = n / 2;
      // Test to see if the "left" is greater than the "right", if yes return false.
      if ( arr[m-1] > arr[m+1] ) {
        return false;
      }

      // Otherwise recurse again.
      return checkSort(arr, m-1);
    }
```

5. Look into System.nanoTime()
    a. RAN OUT OF TIME MY BAD…