Question 2.

Linear Search Algorithm

```
public static int linearSearch(int[] array, int key){
   for (int i =0; i <= array.length-1; i++){       //O(n)
       if (array[i] == key){                        //O(1)
           return i;                                //O(1)
       }
   }
   return -1;                                       //O(1)
}
```

Run time complexity for linear search = O(n) + O(1) + O(1) + O(1) = O(n+3)

Best case run time complexity = O(1), happens when a key is found in the first element of search.
Average case run time complexity = O((n+1)/2), happens when a key is found between the first and last element.
Worst case run time complexity = O(n), happens when the key is found in the last element of search.

Interpolation Search Algorithm

```
public static int interpolationSearch(int[] array, int key){
   //sort algorithm
   Arrays.sort(array);                                          //O(n log n)
   int l = 0;                                                   //O(1)
   int r = array.length - 1;                                    //O(1)

   while (l < r){                                                       //O(n)
       int pos = (key - array[l]) / (array[r] -array[l]);              //O(1)
       int mid = l + (((r-l) * pos));                                   //O(1)
       if (key == array[mid]){                                          //O(1)
           return mid;                                                  //O(1)
       }
       if (key < array[mid]){                                           //O(1)
           r = pos -1;                                                  //O(1)
       }
       if (key > array[mid]){                                           //O(1)
           l = pos +1;                                                  //O(1)
       }
   }
   return -1;                                                   //O(1)
}
```

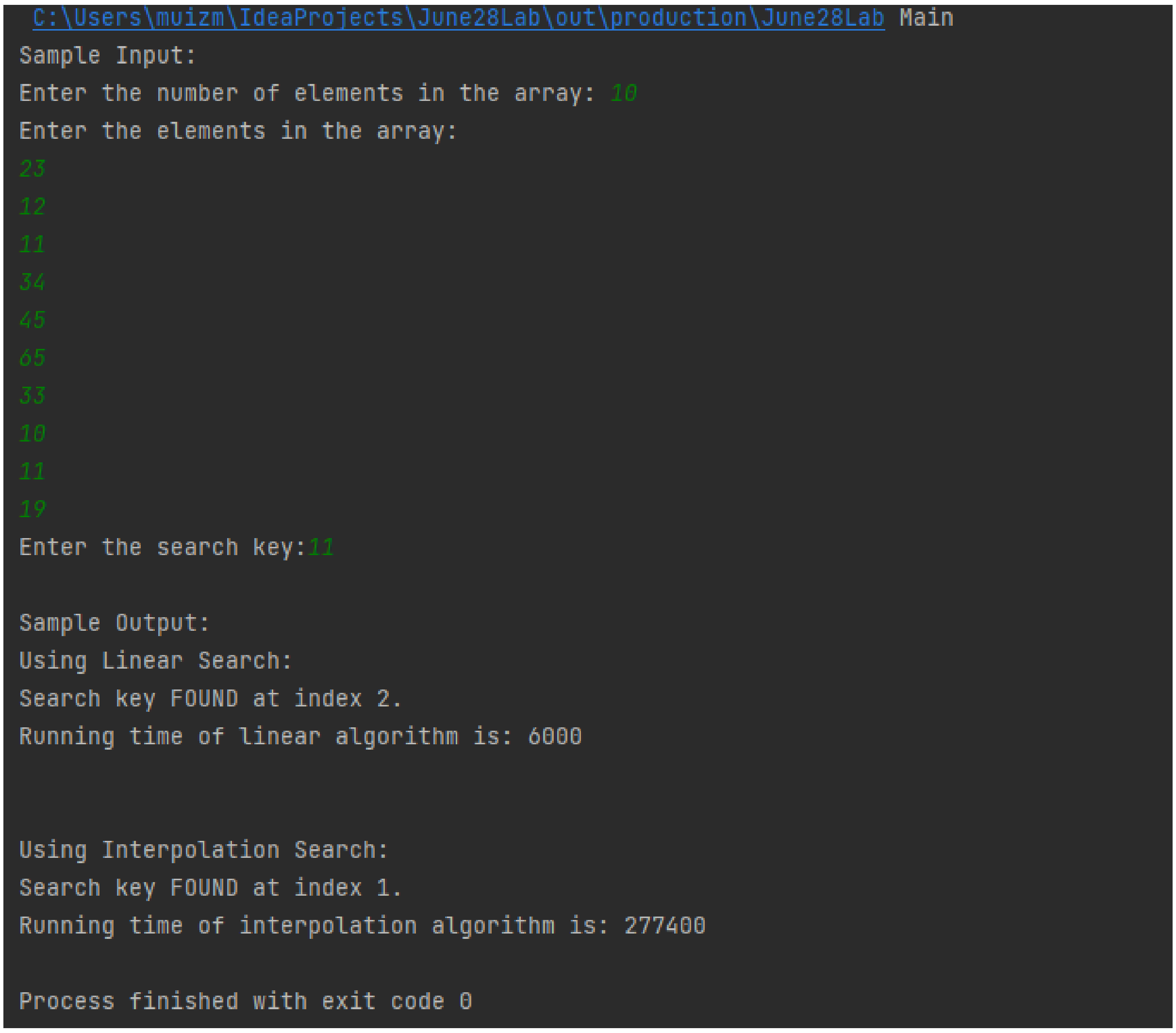
Run time complexity = O(n log n) + O(1) + O(1) + (O(n) + O(1)+ O(1)+ O(1)+ O(1)+ O(1)+ O(1)+ O(1)+ O(1)) + O(1) = O(n log n) + O(1) + O(1) +(O(n) + O(8)) + O(1) = O(n log n) + O(n) + O(11)

Best case run time complexity = O(1), happens when a key is found in the middle element of search.
Average case run time complexity = O(lg (lg n)), happens when a key is found within a few subdivisions of the searched array (excluding the last sub-division).
Worst case run time complexity = O(n), happens when the key is found in the last sub-division of the searched array.

In Theory
While both the linear search and interpolation search had the best runtime of O(1) and the worst runtime of O(n),their performance differs in average case runtime complexity.The linear search average case run time complexity is O((n+1)/2) and,the interpolation search average case run time complexity is O(lg (lg n))

This means, while they both have similar performance in best and worst case, the interpolation search algorithm is slightly faster than the linear search algorithm in the average case because O(lg (lg n)) grows more slowly than O(n+1/2).

In Practice
The following is a screenshot of a sample output and the time it takes for each algorithm to run in nano-seconds. (for the github repo, i have commented the time blocks out)



```
 C:\Users\muizm\IdeaProjects\June28Lab\out\production\June28Lab Main
Sample Input:
Enter the number of elements in the array: 10
Enter the elements in the array:
23
12
11
34
45
65
33
10
11
19
Enter the search key:11

Sample Output:
Using Linear Search:
Search key FOUND at index 2.
Running time of linear algorithm is: 6000


Using Interpolation Search:
Search key FOUND at index 1.
Running time of interpolation algorithm is: 277400

Process finished with exit code 0
```

Using the example provided in the lab sheet, the linear search algorithm finishes in 6000 nano-seconds and the interpolation algorithm takes 277400 nanoseconds. While it is true that the interpolation algorithm should have faster average case time complexity, in this example it is not the case. This is because the example in the screenshot, it uses a small array and for the linear algorithm and with the ordering of the array, it only takes 3 iterations to find the answer. This means for the linear algorithm, the answer is closer to being the best-case (first element being the answer) than the worst case (last element being the answer).

For the interpolation search algorithm, the algorithm needs to create multiple sub-divisions to find the answer. This is far from the best case for this algorithm and more so leaning towards the average to worst case. As such, our interpolation search algorithm performs poorly in this example, making our linear search algorithm faster than the interpolation search algorithm.

Question 3.

The improved running time of the linear search algorithm is reflected in the optimizedLinearSearch() method.

It uses the starting structure of a binary search to sort and find a middle point. If middle point of the array is the key, then we have our best case answer. If not, then the algorithm will partition the array into two halves and use the first half if the key is smaller than the middle point of the array or use the second half if the key is greater than the middle point of the array. Then on that new sub-divided array, we perform a linear search.

The reason this algorithm is more optimized than the original linear search algorithm is because it essentially divides the array in half and does linear search on the subdivision. So if the original array had to do 101 entry checks in worst case, this optimized algorithm will have to do 1 + 50 entry checks (mid point check + all entries in subdivided array).

In this case, the improvement in the algorithm would be (101-51)/101 *100 = 49.50%

However, like question 2, there are cases where the original linear algorithm would be faster, such being where the key is found closer to the beginning of the array rather than farther away from the beginning.