Project Report

Jon Abdulloev, Zeyang Gao, Victor Zuniga

a) How to run our programs:

For C: Commands:

```
gcc filename.c -o filename -lm -std=c99
./filename
```

For python:

```
python filename
```

b) Pseudo-code:

We used an array-based structure.

```
Insertion sort(array, length)
    I, j, key
    for loop through indices of array with j
        key = j in array
        for loop through indices to the left of key with I
            i+1 in array = I in array
        i+1 in array = key
Merge(subarray1, subarray2, array, sublen1, sublen2, arrlen)
    I, j = 0
    while i+j<arrlen
        if j==sublen2 or (i<sublen1 and I in subarray1<j in subarray 2)
            i+j in array = I in subarray1
            i++
        else
            i+j in array = j in subarray2
            j++
Merge sort(array, length)
    if array is trivially sorted return
    find midpoint of array
    create 2 subarrays
    for loop through left half of array with I
        I in subarray1 = I in array
```

```
        for loop through right half of array with j
            j-midpoint in subarray2 = j in array
        merge sort(subarray1, midpoint)
        merge sort(subarray2, midpoint)
        merge(subarray1, subarray2, array, midpoint, midpoint, length)
Partition(array, left, right)
        pivot, I, j, t
        pivot = left in array
        I = left
        j = right+1
        while true
            do ++i while I in array<=pivot and I<=right
            do - -j while j in array>pivot
            if i>=j
                break
            t = I in array
            I in array = j in array
            j in array = t
        t = left in array
        left in array = j in array
        j in array = t
        return j
Quick sort(array, left, right)
        j
        if left<right
            j = partition(array, left, right)
            quick sort(array, left, j-1)
            quick sort(array, j+1, right)
```

c) Running time comparison:

C:

Input size (N): (# of numbers)	Sorting algorithm:	Time cost (sec):
10	Insertion Sort	0.000002
10	Merge Sort	0.000006
10	Quick Sort	0.000002

100	Insertion Sort	0.000012
100	Merge Sort	0.000021
100	Quick Sort	0.000009
1000	Insertion Sort	0.000957
1000	Merge Sort	0.000217
1000	Quick Sort	0.000101
10000	Insertion Sort	0.095074
10000	Merge Sort	0.002620
10000	Quick Sort	0.001200
100000	Insertion Sort	9.418665
100000	Merge Sort	0.030271
100000	Quick Sort	0.014708

Python:

Input size	quicksort	merge_sort	insertion_sort
10	3.409385681152344e-05	3.814697265625e-05	1.3828277587890625e-05
100	0.0007104873657226562	0.00042319297790527344	0.0006475448608398438
1000	0.03674507141113281	0.0050585269927978516	0.06700468063354492
10000	0.6023180484771729	0.06680798530578613	7.780491590499878
100000	0.595299482345581	0.941087007522583	750.9759948253632

d) Responsibility for each group member:

Jon Abdulloev: Quicksort in C and python

Zeyang Gao: Insertion sort and Merge sort in python

Victor Zuniga: Insertion sort and Merge sort in C

Report: together

Python running program: Jon Abdulloev and Zeyang Gao

C running program: Jon Abdulloev and Victor Zuniga

Random number generator in python: Jon Abdulloev

Random number generator in C: Victor Zuniga

e) Questions:

f) Things learned during the project:

Learned about challenges in implementing the sorting algorithms in different languages

Learned that in C it takes less time than in python for very large input sizes.

Learned about collaboration on programming projects