a)

Running Python (on a Mac):

1. Open terminal
2. 'cd' into the same ~~directiog~~ directory as your python (.py) file,
3. Type python3 (name of the file) and run
   eg. python3 main.py.

Running C (on a mac)

1. Open terminal
2. 'cd' into the same directory as your c executable (.c) file.
3. Type 'gcc main.c -o TEST' and hit enter
4. ./TEST and hit enter.

Make sure there is a folder named 'c' and 'Python'. ~~in~~ All the output files are written in these two folders.

b(i) Merge Sort

```
merge_sort (S) :
1.      n = length of list S
2.      if n<2 :
3.          return list S   // length 0 or 1 already sorted
4.      set mid = n//2
5.      set list S1 = S[0: mid]   // copy first half of S
6.      set list S2 = S[mid: n]   // copy second half of S
7.      call merge_sort (S1)   // recursive call 1 on S1
8.      Call merge_sort (S2)   // recursive call 2 on S2
9.      call merge (S1, S2, S)

merge (S1, S2, S) :
10.     set i=0 and j=0   // initialization
11      while i+j < length of list S
12          if j= length of S2 or (i < length of S1 and
            S1[i] < S2[j] then
13.         set S[i+j] = S1[i]
14.         increase i by 1
            else :
15.             set S[i+j] = S2[j]
16.             increase j by 1
            end if
        end while
```

b(ii) InsertionSort(alist):

```
For each index in the length of alist starting at
index 1

    set currentvalue = alist[index]
    set position = index

    while position > 0 and alist[position-1] > currentvalue:
        set alist[position] = alist[position-1]
        decrease position by 1
    End while
    update alist[position] = currentvalue
End for.
```

b(iii) Selection sort

```
For each fillslot in range( len(alist)-1, 0, -1 ):
    Initialize positionOfMax = 0
    for each location in range(1, fillslot +1)
        if alist[location] > alist[positionofmax]:
            set positionofMax= location
        end if
    end for
    ten
    Set temp = alist[fillslot]
    update alist[fillslot] = alist[positionOfmax]
    update alist[positionofmax] = temp.
end for.
```

MERGE SORT		
	Time Cost (microseconds)	
Input Size	Python	C
10	48	**2**
100	400	**12**
1000	5704	**155**
10000	76877	**1470**
100000	6177	**20709**
SELECTION SORT		
	Time Cost (microseconds)	
Input Size	Python	C
10	31	**3**
100	558	**17**
1000	49297	**1760**
10000	131783	**130346**
100000	493859	**13059003**
INSERTION SORT		
	Time Cost (microseconds)	
Input Size	Python	C
10	29	**2**
100	582	**11**
1000	73204	**740**
10000	379596	**75833**
100000	892531	**7100335**

In all the cases, the performance in C is better. This is an expected result. C, a compiled language, is faster than Python, an interpreted language. Python uses C in its implementation, so C is unsurprisingly faster than python.

d) Responsibility of each group member

Maniz

Code execution of insertion and merge sort function in python. Also, debugged C.

Nirdesh

- Code execution in C. Also, debugged python code.

Saujan

- Code execution of selection sort in python. Also, finished the write-up.

(e) • We faced problem/confusion when the rand() in C was generating the same set of numbers upon compilation.

How did we solve this? plan to solve?

After several searching on the internet, we still couldn't find a concrete answer.

This wasn't a major problem as the program sorted the numbers properly, but we wanted to see if we get a new set of random numbers each time.

We plan to try using srand() instead of rand() and modify the code a bit.

- It took a long time to run in Python.

  solution

  We only used file sizes with a maximum of 100000 elements. We were trying 1000000, so we removed that in the interest of time.

- Debugging C was tough because it's a compiled language.

  For this we took help of various internet resources.

f) This project helped us to put the theory learned in class about the efficiency of sorting algorithms into perspective by running them in code.

The efficiency of the sorting algorithms is easy to compare when programmed.

This project was also a good demonstration of the two programming languages used, namely python and C. We found C to be faster. Python itself is written in C so its implementation involves running C.
In addition to this, python is an interpreted language, while C is not. In Python, the line-by-line execution by the interpreter can involve a lot of unnecessary repeated translations.
C is compiled, meaning the whole source code is translated into machine code in one-go.

comments:

This was a useful project to finally program in this more of a theoretical class. It was nice to see the practicality of the theory of sorting algorithm.

Working in C was little tough because none of us had adequate experience with C, barring the things learned in class. Working with multiple executable files was confusing. So, it would be nice if we could go over C programming once more in class.