Assignment 1 Writeup

Andrew Tackett

9/26/21

CS 315

Section 002

Problem Statement

We were given the task of sorting all Pokémon based on the Stats (Power). We are supposed to sort using Insertion Sort, Merge Sort, and Quick Sort.

Implementation

I just used the readFile.cpp that was given for my main program and template. I implemented my Insertion Sort as follows:

void insertionSort(vector<int>& data, int n, int& count)

I used the parameters as a vector since the read is in a vector. I also just used n to look at the size of the array in the main function (power.size()). For this data I used the for loop to iterate until i is less than n. In the while loop we look at the vector of “data” and make a new variable to house the data with a temporary value. We subtract one from j every single time and up the count when this occurs.

Merge Sort –

For the Merge Sort we will make the parameters as given:

void MergeSortedIntervals(vector<int>& v, int s, int a, int b, int& count)

we make a temporary vector being “temp” and make separate variables to iterate. We make a while loop and push the intervals back in the temporary vector to house new values, while also upping the count. We also have a count in the else loop that will push as well when i is less than a and j is less than b. We have two separate while loops that check to see if they are less than a and b. if they are they will use the temporary vector respectively and add a count. After we have a for loop that will check if I is equal to s, after all conditions are met, the v index will become equal to the temporary value [i – s]. We then make a separate function called MergeSort to incorporate the MergeSortedIntervals.

Quick Sort –

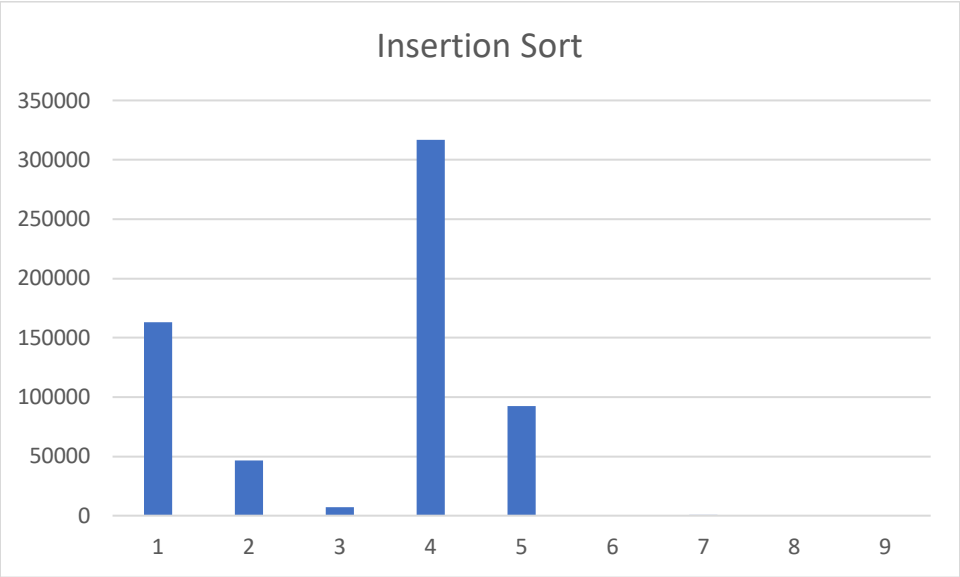
For the Quick Sort we make a partition function and a Quicksort Function. We have a vector again with a start, end, and a count. We set the pivot to the end and an integer to the start. We have multiple loops to iterate through the array and then swap them at the end.

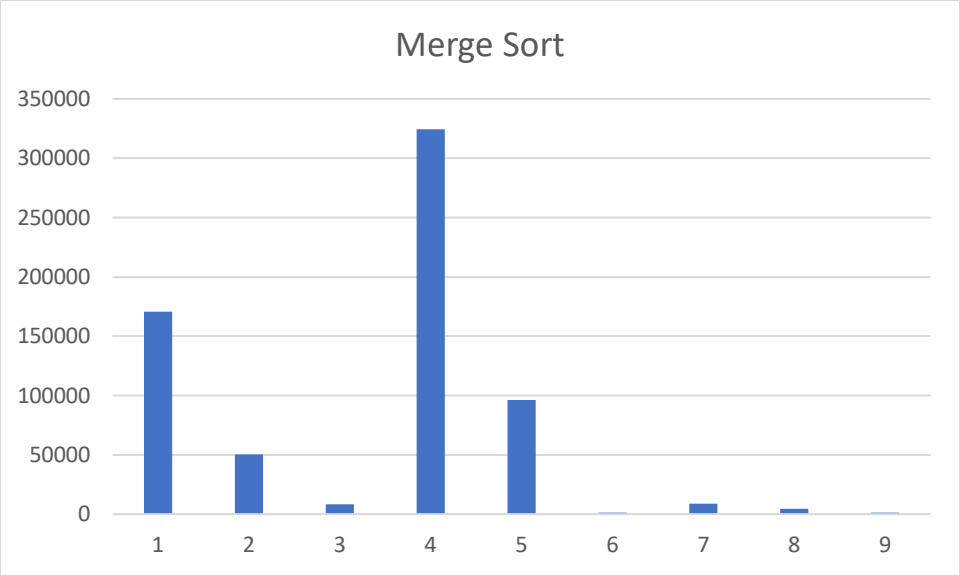
After that we go into the main to look at how everything sorted.

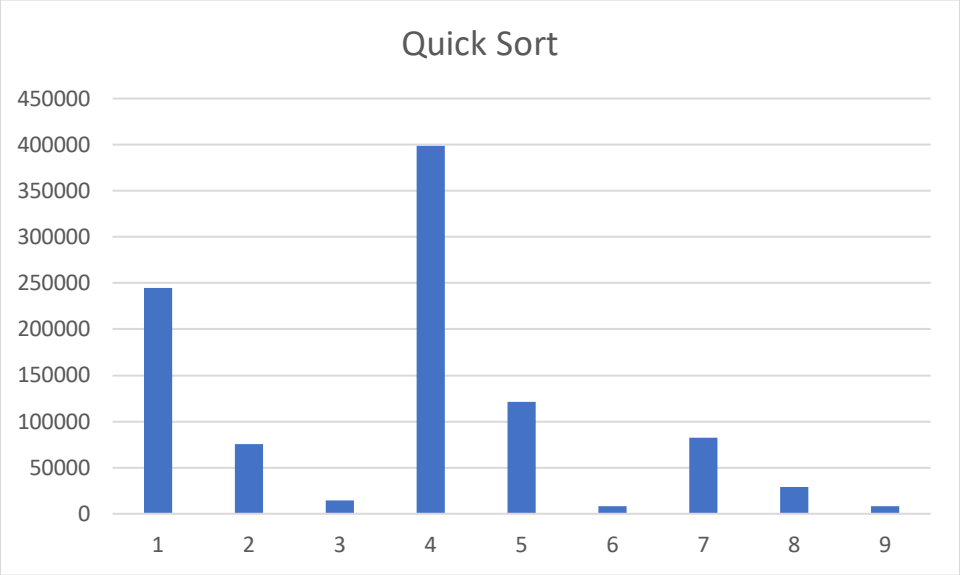
Results

I21

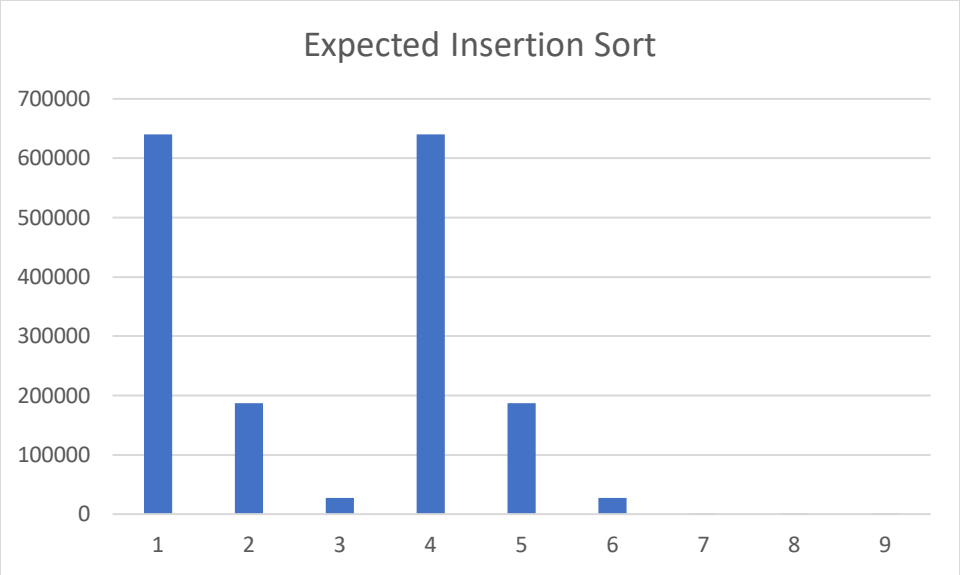
	C	D	E	F	G	H	I	J	K
1	RandomMedium	RandomSmall	ReverseSortedLarge	ReverseSortedMedium	ReverseSortedSmall	SortedLarge	SortedMedium	SortedSmall	
2	46589	6934	316730	92330	165	799	**431**	165	
3									
4	50397	8172	324506	96138	1403	8575	4239	1403	
5									
6	75229	14630	398513	120970	7861	82582	29071	7861	
7									
8	432	166	800	432	166	800	432	166	
9									
10	RandomMedium	RandomSmall	ReverseSortedLarge	ReverseSortedMedium	ReverseSortedSmall	SortedLarge	SortedMedium	SortedSmall	
11	186624	27556	640000	186624	27556	800	432	166	
12									
13	1138.528	368.538	640000	186624	27556	2322.472	1138.529	368.538	
14									
15	1138.528	368.538	2322.471	1138.529	368.538	2322.472	1138.529	368.538	
16									
17	432	166	800	432	166	800	432	166	
18									
19									

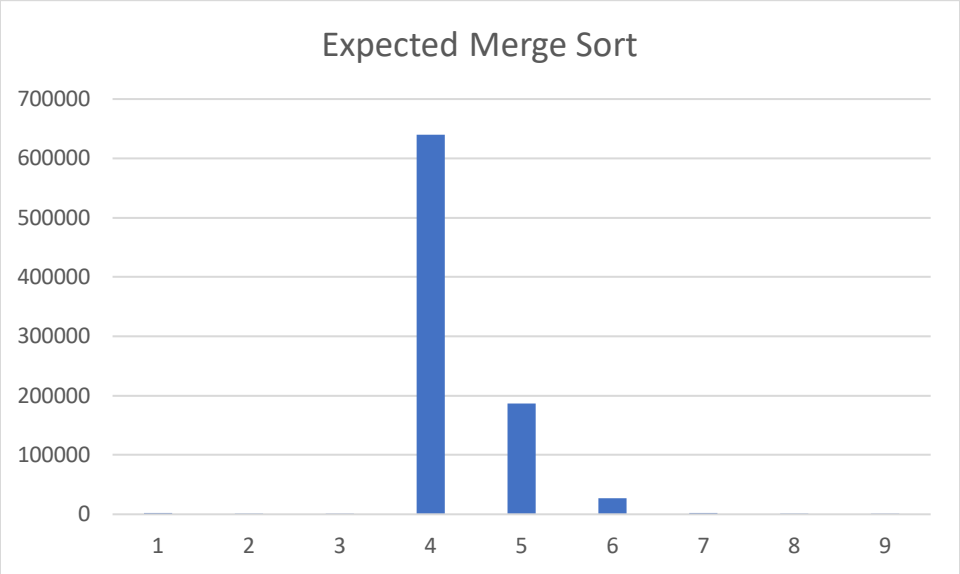


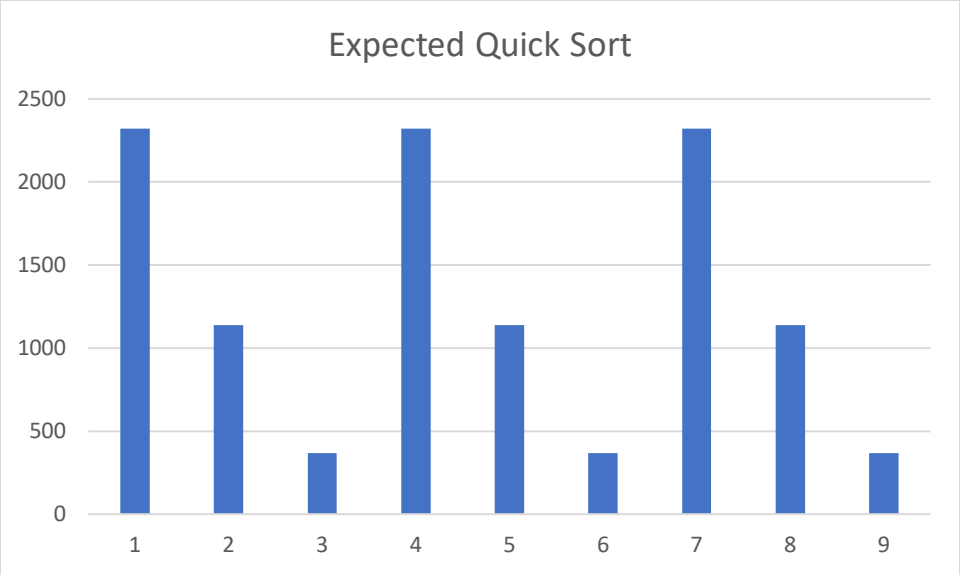




Here is the Expected Graphs:


Expected Insertion Sort
700000
600000
500000
400000
300000
200000
100000
0
1
2
3
4
5
6
7
8
9


Expected Merge Sort
700000
600000
500000
400000
300000
200000
100000
0
1
2
3
4
5
6
7
8
9



My results were somewhat close, with a couple anomalies and in the Merge Sorts, but most were within a certain margin of error. My results were also based on off all nine files of Pokemon. The order given by the excel spreadsheet is the order of the graphs shown above.