Project 1

1 Introduction

In Project 1, we compared two sorting algorithms, selection sort and insertion sort. Insertion sort is an algorithm which takes a cell and inserts it in the corresponding placement according to order. For example, if we have an array A=[3,1,7,5,9], the algorithm would assume the first number as the smallest. In the next loop, it would take the second number and consider whether it was smaller or greater than the first. Because it is smaller, the number is sorted into the first position. This continues with all numbers in the array. Selection sort is an algorithm which takes a cell and switches it with the lowest number in the unsorted cells. Going back to array A, the algorithm would take the first number and search for the smallest number in the cells after it. In this case, the smallest is 1. Thus, the 1 and the 3 would switch places. In essence, insertion sorting is like cutting a line while selection sort is swapping places. In order to compare the two sorting algorithms, both will be restricted to a nested while loop in a for loop. We will be using the timeit module in order to find the timeframe for runs with cell size 1000,2500,5000,7500, and 10000 of increasing, decreasing, and randomly sorted arrays.

2 Selection Algorithm (Lines 4-17)

The selection algorithm is very simple in nature; we choose a cell to change in the array, find the minimum digit in the unsorted part of the array, and change the variables. Beginning with the first part of our algorithm, we know that we need to eventually go through every cell in the array to change it. Because there is repetition, we can use a for loop (line 6) that will go through the entire length of the array. Lines 7 and 8 establish aliases to index the array. In order to find the minimum digit in the sorting array, we need to go through every single number in the sorting array and establish whether it is smaller than our cell we want to change. Thus, we need another loop, a while loop! Lines 9-11 determine the minimum and create an alias in order for us to exchange variables later. Lastly, we need to exchange the minimum value in the unsorted array with the cell. This part of the code was very problematic for me, as I did not think it needed to be in the while loop. Upon further examination, I realized that not only would the python language not allow me to put the change of variables outside the while loop, but also that logically it had to go in the while loop. The exchange of variables is quite simple; one must first set up a variable that will hold one of the values. Then, we can change the variables by using that extra value. If we don't add an extra variable, python will overwrite one of the variables, thus destroying the previous relationship.

3 Insertion Algorithm (Lines 19-26)

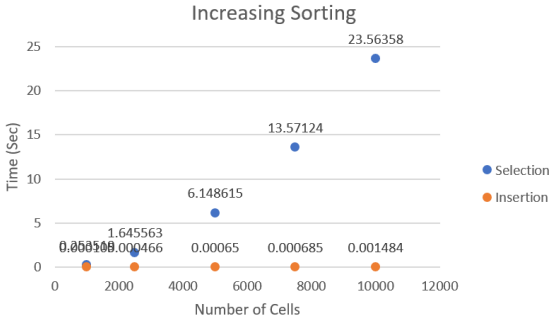
The insertion algorithm shares similar traits with the selection algorithm but differs greatly in how it functions. The insertion algorithm involves choosing a cell in the array, looking to its left neighboring cell, and deciding if it is smaller or not. Then, the variables are shifted. In order to apply this, the cell we want to move is first indexed with a for loop and establishment of aliases (lines 19-22). It is important to note that we must start with the second cell because the first cell has no neighbors to its left! The remainder of the code (lines 23-26) shows that while the digit to the left of the cell is smaller than the digit inside the cell, the two must switch values, causing the digit in the cell to "slide" to its position.

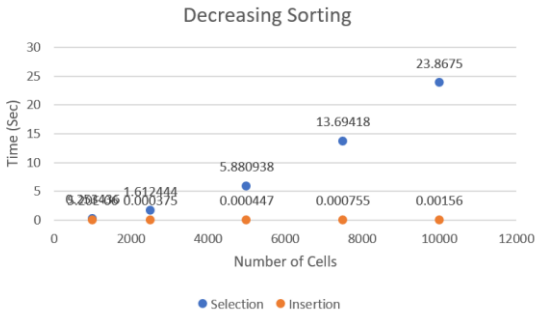
4 The Timeit Module (Lines 28-71)

In order to generate our code, we need to generate results. Starting with generating an increasing array, we can create a list (line 35) , add a random value to the list(line 36), and then add a random number greater than the last to the list for some length of the list. The list is then copied in order to be run on both insertion and selection sort. Because our project guidelines indicate we want to do this for five sizes, we can create a for loop to repeat this (line33 and 30). Decreasing arrays (lines 43-48) are done almost identically; the parameters change for line 47. The random array is done a little differently(lines 51-54). We create a list and then generate random numbers the size of the array. Once again, the list is copied. Both decreasing and random array generation algorithms are nested inside the for loop accounting for size.

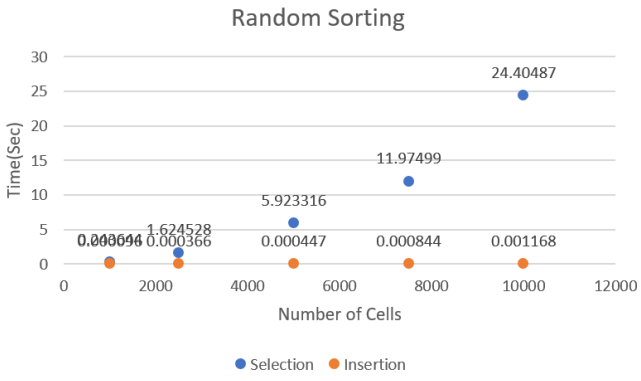
Continuing with the code, we then must apply the selection_sort and insertion_sort algorithms to our lists (lines 56-61). Lastly, we must print the timings(lines 63-71). In order to print all the timings for each list, two for loops are created containing the selection and insertion values. This entire block of code (lines 56-71) is contained inside the for loop accounting for size, and will be repeated for each size. Thus, the output will be three lines of selection sorting (increasing, decreasing, random) and three lines of insertion sorting (increasing, decreasing, random), all which repeat over different number of cells for a total of thirty results.

5 The Results





For this project, we included three distributions of arrays: an array in increasing order, an array ordered in decreasing order, and one in random order. From the graphs, it can be seen that insertion sort performed the best in all three circumstances. Selection sort grows exponentially in each distribution, while insertion sort increases minimally. We can note that as the number of cells increase, both sorting algorithms take longer to process. Selection sort ranged from ~0.24 seconds to ~24 seconds, while insertion sort ranged from ~9 x 10^{-5} seconds to ~0.0015 seconds. The fastest sorting algorithm occurred for decreasing insertion at 1000 cells. From the averages, we can note that insertion sort had the fastest results with random distributions across all number of cells, while selection sort struggled the most with increasing distributions. In order to confirm this, many more trials and number of cells need to be run, as the numbers for each distribution tend to be quite similar. Why does one algorithm outperform the other?



In essence, one algorithm searches through all the necessary sorted cells (insertion) while the other searches through all the unsorted cells. Because insertion stops once it has found the position for placement, it streamlines the time necessary for sorting.

Data Table

Sorting Type	Number of Cells	Distributi	TRIAL 1	TRIAL 2	TRIAL 3	TRIAL 4	TRIAL 5	AVG
Selection Sort	1000	increasin	0.220469	0.332338	0.238985	0.242105	0.233696	0.253519
		decreasin	0.253242	0.346334	0.250989	0.205346	0.211269	0.253436
		random	0.264745	0.313891	0.223598	0.198833	0.217152	0.243644
Insertion Sort		increasin	0.000235	0.000045	0.000058	0.000094	0.000094	0.000105
		decreasin	0.000091	0.000143	0.000055	0.000093	0.000094	9.52E-05
		random	0.00009	0.000149	0.000053	0.000095	0.000093	0.000096
Selection Sort	2500	increasin	1.509174	2.181956	1.507875	1.473074	1.555738	1.645563
		decreasin	1.431366	2.286012	1.43355	1.424964	1.486326	1.612444
		random	1.514231	2.186694	1.474853	1.490427	1.456433	1.624528
Insertion Sort		increasin	0.000227	0.00092	0.000235	0.000721	0.000229	0.000466
		decreasin	0.000224	0.000908	0.000263	0.00025	0.000229	0.000375
		random	0.000223	0.000911	0.000226	0.000244	0.000224	0.000366
Selection Sort	5000	increasin	6.04778	7.37257	5.836792	5.584655	5.901279	6.148615
		decreasin	6.204131	5.629882	5.870906	5.831808	5.867962	5.880938
		random	6.088466	6.246306	5.771216	5.893476	5.617117	5.923316
Insertion Sort		increasin	0.00117	0.000454	0.000458	0.00071	0.000458	0.00065
		decreasin	0.000449	0.000449	0.000446	0.000444	0.000448	0.000447
		random	0.000446	0.000447	0.000448	0.000443	0.000449	0.000447
Selection Sort	7500	increasin	14.77628	13.70886	12.72446	13.28845	13.35812	13.57124
		decreasin	13.97599	13.61644	13.55124	13.55022	13.777	13.69418
		random	6.959678	12.86704	13.6085	12.89093	13.54883	11.97499
Insertion Sort		increasin	0.000682	0.000687	0.00068	0.000689	0.000686	0.000685
		decreasin	0.000673	0.000665	0.001099	0.000668	0.00067	0.000755
		random	0.000973	0.000951	0.000669	0.000669	0.000959	0.000844
Selection Sort	10000	increasin	22.9533	23.09961	24.55821	23.75591	23.45088	23.56358
		decreasin	23.64783	24.11399	23.90686	24.06171	23.60714	23.8675
		random	25.78012	24.71355	24.40187	23.71915	23.40965	24.40487
Insertion Sort		increasin	0.001147	0.00217	0.000957	0.001842	0.001304	0.001484
		decreasin	0.000889	0.000887	0.00267	0.002445	0.00091	0.00156
		random	0.001032	0.000892	0.00214	0.000886	0.000889	0.001168