Lecture 6: Problem Solving - Sorting

September 6, 2023

Revision

Algorithms

Selection Sort

⇒ In every iteration pick the `i'th` smallest number and place that on the `i'th` position.

```
function selection_sort(array){
    for i in 0...N {
        min_ele = find_min(array, i, N)
        exchange min_ele with i'th index in array
    }
}
```

Insertion Sort

⇒ In every iteration, choose `i'th` element and find its correct position in array[0:i-1]

```
function insertion_sort(array){
    for i in 0...N {
        pos = find_correct_poision(array, 0, i-1)
        insert i'th element at pos in array
    }
}
```

Merge Sort

⇒ Divide the current array into two equal parts and recursively sort them. After both parts are sorted, merge them to get the whole array sorted.

```
function merge_sort(array){
    if len(array) == 0:
        return;

    merge_sort(array[0   : n/2]);
    merge_sort(array[n/2 : n]);

    merge(array, 0, n/2, n);
}
```

Quick Sort

⇒ Pick an element, divide the array in two sets such that one contains all elements less than the picked element and other one contains larger elements. And recursively do this on those two sets.

```
function quick_sort(array){
      pivot_index = partition(array);

      quick_sort(array[0             : pivot_index])
      quick_sort(array[pivot_index+1 : n])
}
```

Count Sort

⇒ Count the frequency of all elements and store them in an array. Using the linear property of an array, we can traverse the array in increasing order and output numbers based on their frequencies.

```
function count_sort(array, valueEnd){
      create an array called Counts of size valueEnd.
      calculate the frequency of each element in the array
      print all elements linearly as per the Count array
}
```

Radix Sort

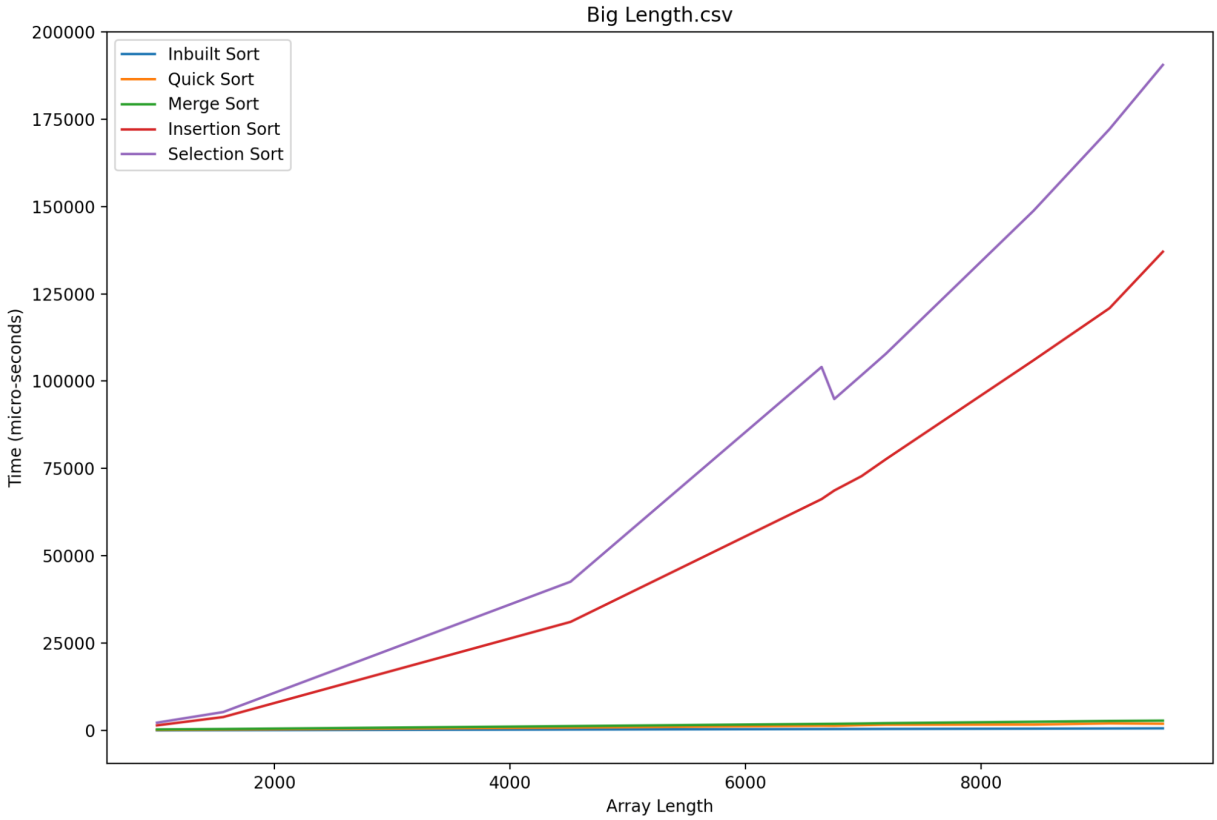
⇒ Sort all numbers digit by digits using a stable sorting algorithm.

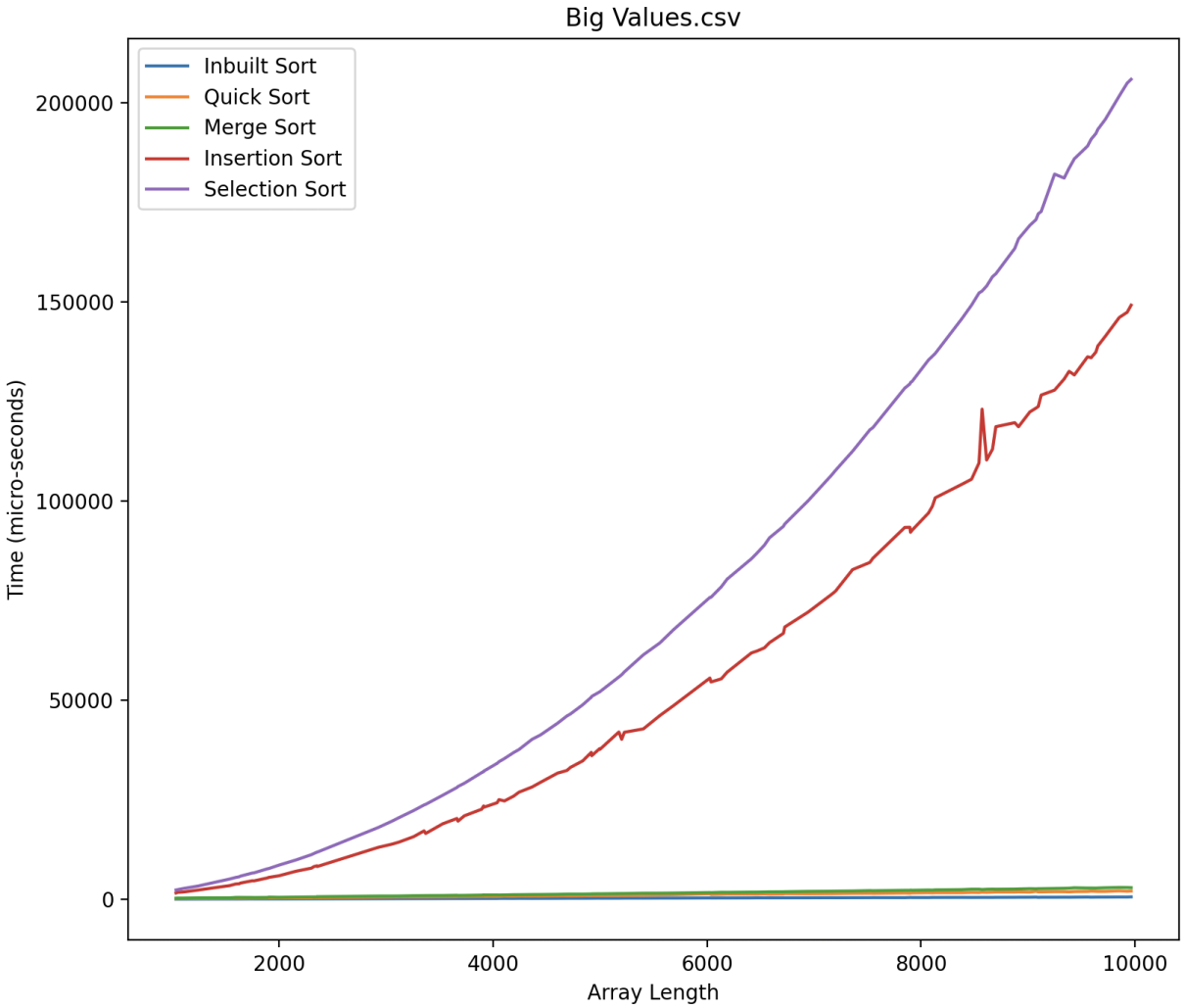
```
function radix_sort(array){
      for d in 0...total_digits:
            count_sort based on digit d
}
```

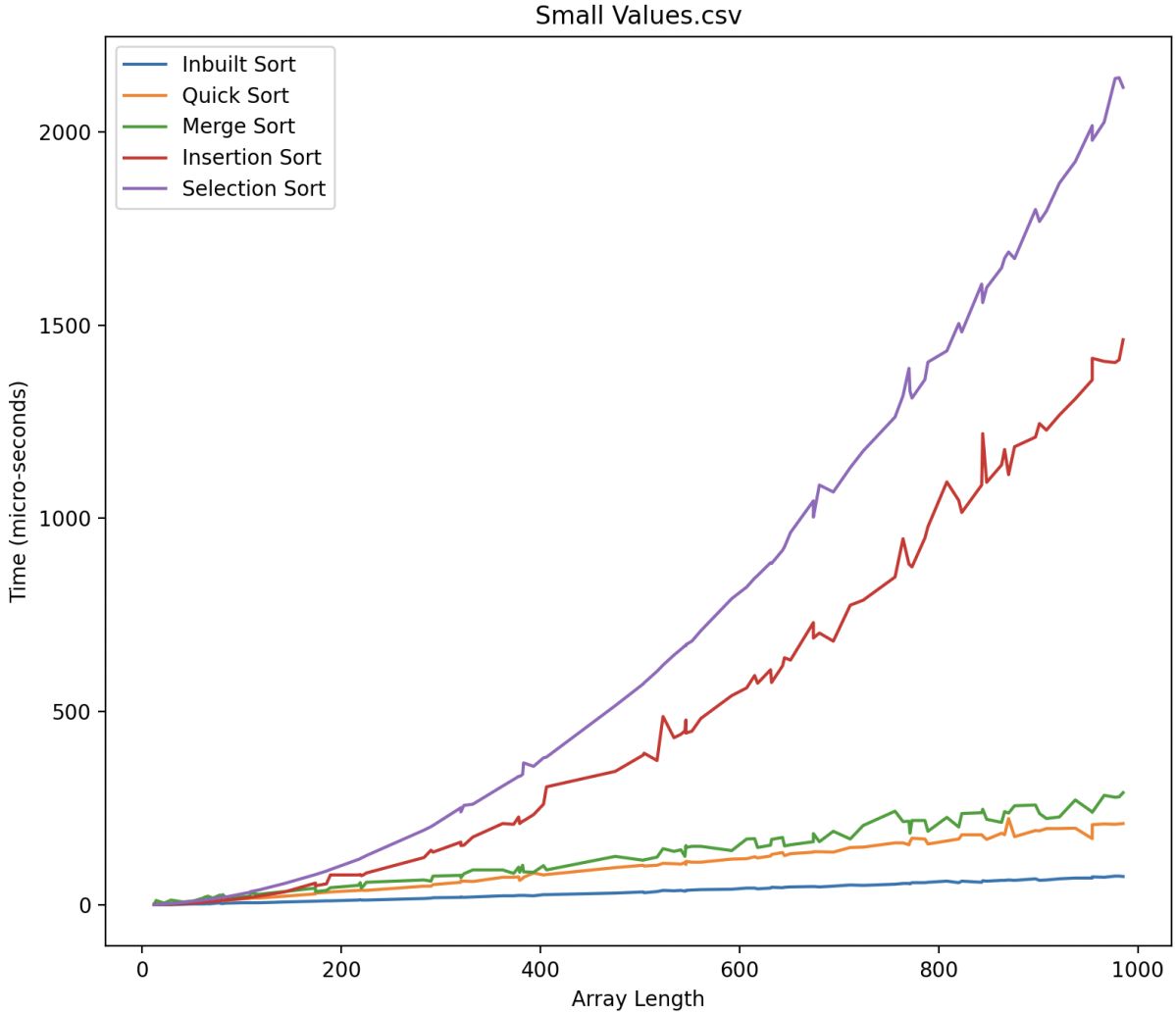
Comparison

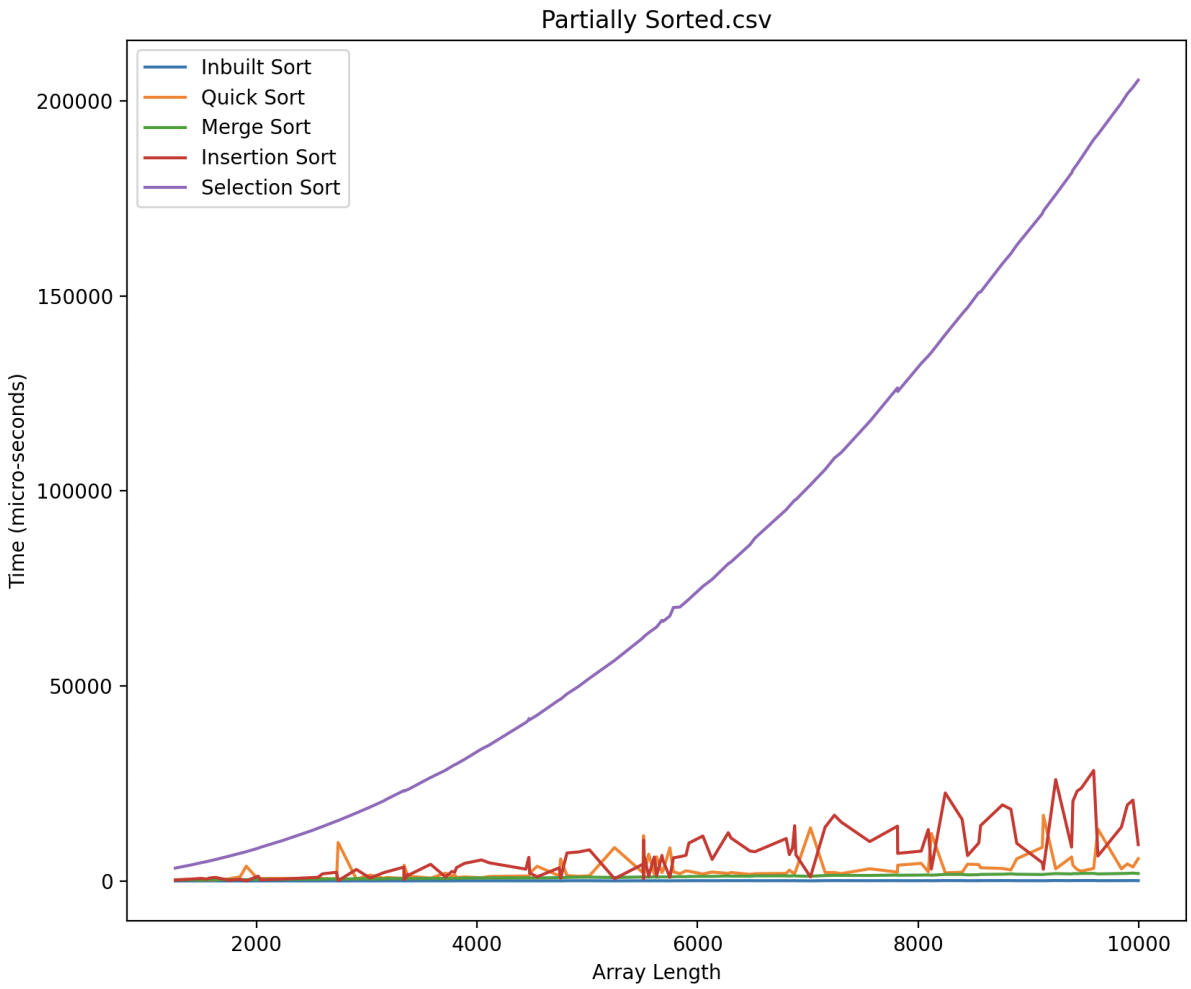
Algorithms	Time Complexity		Space Complexity
	Best Case	Worst Case	
Selection Sort	$\theta(n^2)$	$\theta(n^2)$	$\theta(1)$
Insertion Sort	$\theta(n)$	$\theta(n^2)$	$\theta(1)$
Merge Sort	$\theta(n * \log(n))$	$\theta(n * \log(n))$	$\theta(n)$
Quick Sort	$\theta(n * \log(n))$	$\theta(n^2)$	$\theta(\log(n))$
Count Sort	$\theta(n + max\_ele)$	$\theta(n + max\_ele)$	$\theta(max\_ele)$
Radix Sort	$\theta(n * digits)$	$\theta(n * digits)$	$\theta(n)$

Graphs


Big Length.csv
Inbuilt Sort
Quick Sort
Merge Sort
Insertion Sort
Selection Sort
Time (micro-seconds)
200000
175000
150000
125000
100000
75000
50000
25000
0
2000
4000
6000
8000
Array Length


Big Values.csv
Inbuilt Sort
Quick Sort
Merge Sort
Insertion Sort
Selection Sort
200000
150000
100000
50000
0
Time (micro-seconds)
2000
4000
6000
8000
10000
Array Length


Small Values.csv
Inbuilt Sort
Quick Sort
Merge Sort
Insertion Sort
Selection Sort
Time (micro-seconds)
0
500
1000
1500
2000
0
200
400
600
800
1000
Array Length



Exercises

⇒ **Which method runs faster for an array with all keys identical, selection sort or insertion sort?** (Algorithms, 4e. problem 2.1.6)
Insertion sort, because it will stop at just the first comparison with the previous element. Whereas the selection sort will have to go through all elements in array to find minimum in every iteration

⇒ **Which method runs faster for an array in reverse order, selection sort or insertion sort?** (Algorithms, 4e. problem 2.1.7)
Both algorithm will run in quadratic time but selection sort will do less array exchanges which will be faster.

⇒ **Suppose that we use insertion sort on a randomly ordered array where elements have only one of three values. Is the running time linear, quadratic, or something in between?** (Algorithms, 4e. problem 2.1.8)
Quadratic, because it will be linear only when the array is already sorted.

⇒ **Give examples of best case / worst case of Insertion Sort**
Worst Case: [5, 4, 3, 2, 1]
Best Case : [1, 2, 3, 4, 5]

⇒ **Give examples of best case / worst case of Quick Sort (first pivot)**
Worst Case: [5, 4, 3, 2, 1] or [1, 2, 3, 4, 5]
Best Case : [5, 3, 1, 2, 7, 6, 8]

⇒ **Give examples of best case / worst case of Count Sort**
Worst Case: [100,1000,500,431,560]
Best Case : [0,1,2,1,1,1,0,1,0,1,2,2,0,1,0,2,2,1]

Problems

1. Missing Number
2. Find The Difference
3. Count Inversion
4. Given an array of positive and negative integers, segregate them in linear time and constant space. The output should print all negative numbers, followed by all positive numbers.

   Input: { 9, -3, 5, -2, -8, -6, 1, 3 }
   Output: { -3, -2, -8, -6, 5, 9, 1, 3 }

5. Find the largest number possible from a set of given numbers where the numbers append to each other in any order to form the largest number.

   Input: { 10, 68, 75, 7, 21, 12 }
   Output: 77568211210