NameThatSort

Name: Taras Derewecki
PSU ID (Last 4 digits): 5672

Results:

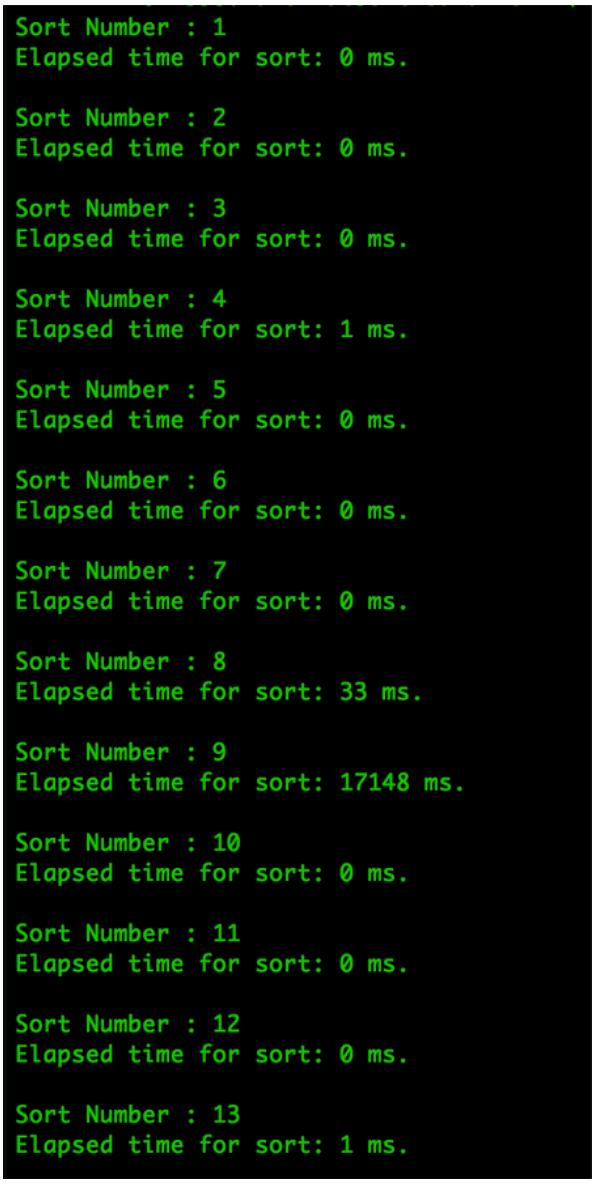
Sort No		Sort
1	-	Randomized Quick Sort
2	-	Quicksort
3	-	Cocktail Shaker Sort
4	-	Radix Sort
5	-	Selection Sort
6	-	Heap Sort
7	-	Insertion Sort
8	-	Bucket Sort (Extra Credit)
9	-	Bogo Sort
10	-	Bubble Sort
11	-	Merge Sort
12	-	Stooge Sort
13	-	Count Sort

Input 1:

Hypothesis:
Identifying Bogo Sort with a trivial input

Description:
Basic input containing 15 elements.

Results:



Conclusion:
Since every sorting algorithm apart from No 9 completed in less than one second, sorting algorithm 9 is Bogo sorting

Input 2:

Hypothesis:

Stooge sort should do worse than the other sorts for a large input and since it is sorted, we should be able to apply best case, worst case analysis for a few sorts such as bubble and insertion sort.

Description:

Numbers from 1 to 10000 which is already sorted

Results:

Conclusion:

Since the input size is relatively high and the array is already sorted, Stooge Sort should do worse than the other sorts.

This is clearly observed as Sort Number 12 is clearly more time consuming than the other sorts.

Input 3:

Hypothesis:

We should be able to filter out the basic sorts based on best-worst case analysis after testing with a large input which is reverse sorted.

Description:

Numbers from 1 to 10000 which is sorted in reverse

Results:

```
Sort Number : 1
Elapsed time for sort: 5 ms.

Sort Number : 2
Elapsed time for sort: 120 ms.

Sort Number : 3
Elapsed time for sort: 112 ms.

Sort Number : 4
Elapsed time for sort: 10 ms.

Sort Number : 5
Elapsed time for sort: 107 ms.

Sort Number : 6
Elapsed time for sort: 4 ms.

Sort Number : 7
Elapsed time for sort: 124 ms.

Sort Number : 8
Elapsed time for sort: 8 ms.

Sort Number : 10
Elapsed time for sort: 115 ms.

Sort Number : 11
Elapsed time for sort: 16 ms.

Sort Number : 13
Elapsed time for sort: 218 ms.
```

Conclusion:

Now, sorts 3, 7, 10 are performing very well for input 2, (Already sorted) and poorly when input is reverse sorted.

Thus, based on how the algorithms function, these must be

bubble sort, insertion sort and cocktail shaker sort (Best case, worst case analysis)

Input 4:

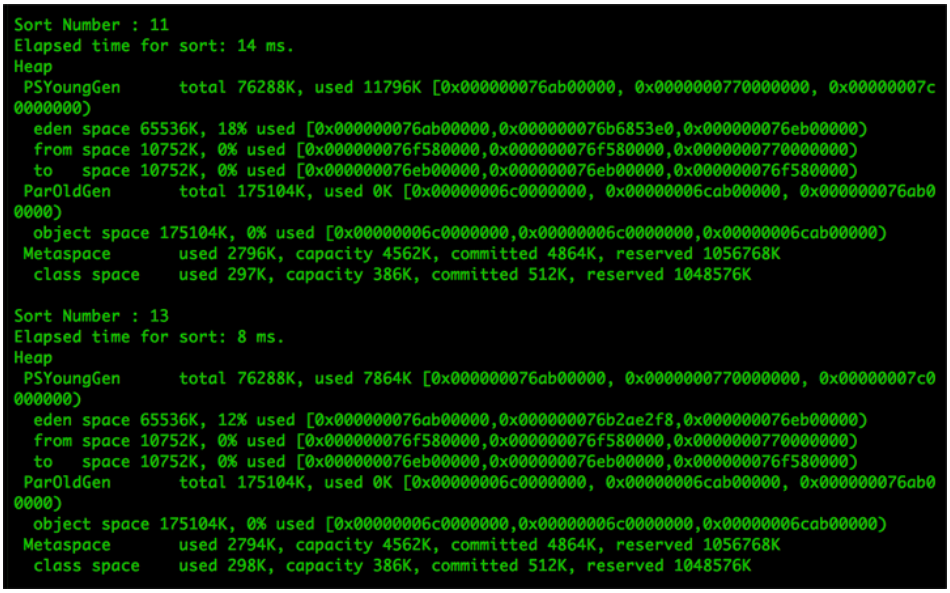
Hypothesis:

A randomised input of large size would help us with further tests and we also check the memory usage to identify merge sort.

Description:

Numbers from 1 to 10000 which are randomly generated

Results:



Conclusion:

Sort No 11 takes more memory (verbose:gc) and performs well in all cases. Thus, it is clearly Merge Sort

Input 5:

Hypothesis:

Providing an input with duplicates helps us figure out the sorts which handle stability. I.e, we should be able to filter out and analyse heap, quick and randomised quick sort.

Description:

Numbers with duplicates in reverse to filter out sorts which handle stability.

Results:

```
Sort Number : 1
Elapsed time for sort: 0 ms.
1 b
1 a
50 c
50 d
100 e
100 f

Sort Number : 2
Elapsed time for sort: 0 ms.
1 b
1 a
50 c
50 d
100 e
100 f

Sort Number : 3
Elapsed time for sort: 0 ms.
1 a
1 b
50 c
50 d
100 e
100 f

Sort Number : 4
Elapsed time for sort: 0 ms.
1 a
1 b
50 c
50 d
100 e
100 f

Sort Number : 5
Elapsed time for sort: 0 ms.
1 a
1 b
50 c
50 d
100 e
100 f

Sort Number : 6
Elapsed time for sort: 0 ms.
1 a
1 b
50 c
50 d
100 f
100 e

Sort Number : 7
Elapsed time for sort: 0 ms.
1 a
1 b
50 c
50 d
100 e
100 f
```

Conclusion:

Sorts 1, 2 and 6 are not stable sorts. Which means, these are Quick sort, Randomized quick sort and heap sort.

Now, from Input 2 and 3, we can clearly infer that Sort number 2 is quick sort. As the it performs poorly when compared to sorts 1 and 6.

Also, based on the order in which the output elements are arranged,

Sort Number 6 does not preserve stability during the heapify process in the last two elements (min heap).

While randomized quick sort is analogous to quick sort in the order of stability.

Thus, Sort 1 is randomized quick sort and Sort 6 is heap sort.

Input 6:

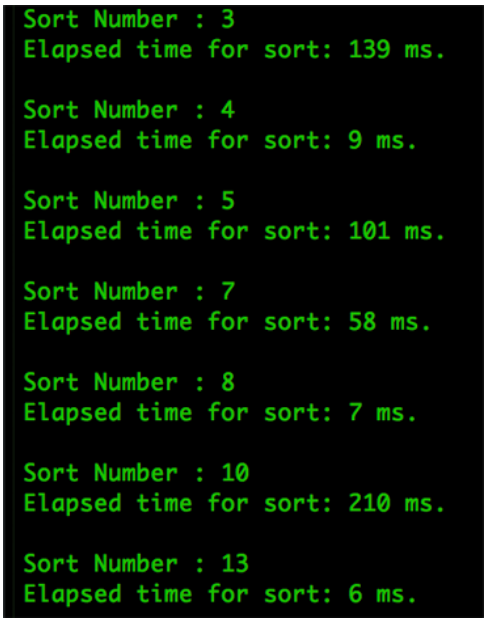
Hypothesis:

Providing a large number of inputs with a small span area will help us in figuring out count sort, as it would have lesser values to hash.

Description:

Numbers from 1-100 repeated 100 times each and shuffled.

Results:



Conclusion:

Sort number 13 clearly does better in this case when compared to the same input size when it is spread out (input 2).

Thus, sort 13 is Count sort (More values to hash)

Now, based on data collected from all these inputs, we are left with 6 sorting algorithms,
Sorts 3, 4, 5, 7, 8 and 10
We also know the following:
Sorts 3, 7, 10 are one of bubble sort, insertion sort and cocktail shaker sort each
Thus sorts 4, 5, 8 are one of radix sort, bucket sort (Extra credit) and selection sort
Now, sort 5 is clearly worse than 4 and 8 for all inputs, and is thus selection sort.

Input 7:

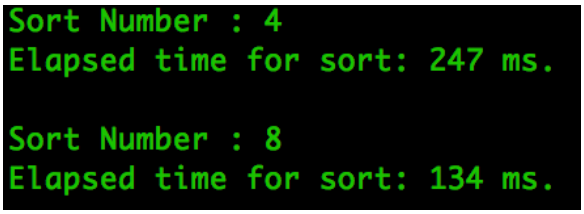
Hypothesis:

To differentiate between bucket sort and radix sort, we provide a large input with dense values. Bucket sort should

clearly perform better for this input.

Description:
Dense array (1-100) with large input size to differentiate between bucket and radix sort.

Results:



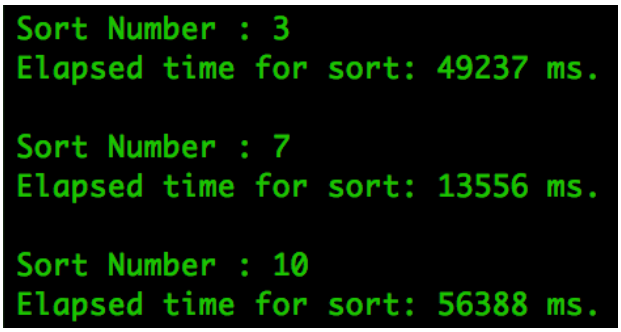
Conclusion:
We know that bucket sort does better than radix sort for dense inputs. To observe the difference, we took a large input and clearly, sort 8 does better than sort 4.
Thus, sort 8 is bucket sort, sort 4 is radix sort.

Input 8:

Hypothesis:
Finally, we run the remaining sorts on a large input so that we can map the three remaining sorts. Clearly for a random input, insertion sort > cocktail shaker sort > bubble sort.

Description:
A large input that is randomly arranged to differentiate between the three sorts remaining.

Results:



Conclusion:

The array is reverse sorted, thus we know that insertion sort should do better than cocktail shaker sort and bubble sort should perform the worst.

Thus, sort 3 is Cocktail Shaker sort, sort 7 is insertion sort and sort 10 is bubble sort.