Linked List and Array Analysis

Will Pringle - September 29 2019

Discussion

Arrays and Linked Lists are used extensively in computer applications. Each of the two data structures are implemented differently and are therefore more efficient for certain circumstances. The most notable differences are: that arrays are often fixed in size whereas linked lists are dynamic, accessing an element anywhere in an array is simple but doing so with a linked list requires traversing through the list from the start (or in the case of doubly linked list) from the end as well and adding a new element anywhere in a linked list is simple but requires shifting all the other elements in the case of arrays. From the research, I was able to conclude that swapping is more efficient with arrays, but increases in resources non-linearly (whereas linked list increases linearly). This means arrays are more efficient with smaller numbers of elements but may be less efficient with very large number of elements in comparison to linked lists.

Practical Examples of Arrays include use in the Radix LSD Sorting Algorithm, and an array of each month of the year. The Radix LSD sorting algorithm uses a fixed number of buckets, one for each digit (10 for base ten). Since during runtime, the amount of digits in the base ten number system won't change, there is no need to have the size dynamically change. Furthermore, since the amount of buckets does not change, the algorithm's efficiency won't be decreased by the negative effects of inserting/deleting an element in the middle of an array (causing the others to be shifted). There are only twelve months in the year and this amount will not change. This means if a program uses this, it will gain the benefits of ease of access of any element, month, (for instance in a calendar program) and will not face the negative impact of having to shift many elements over in the case of an insertion or delete.

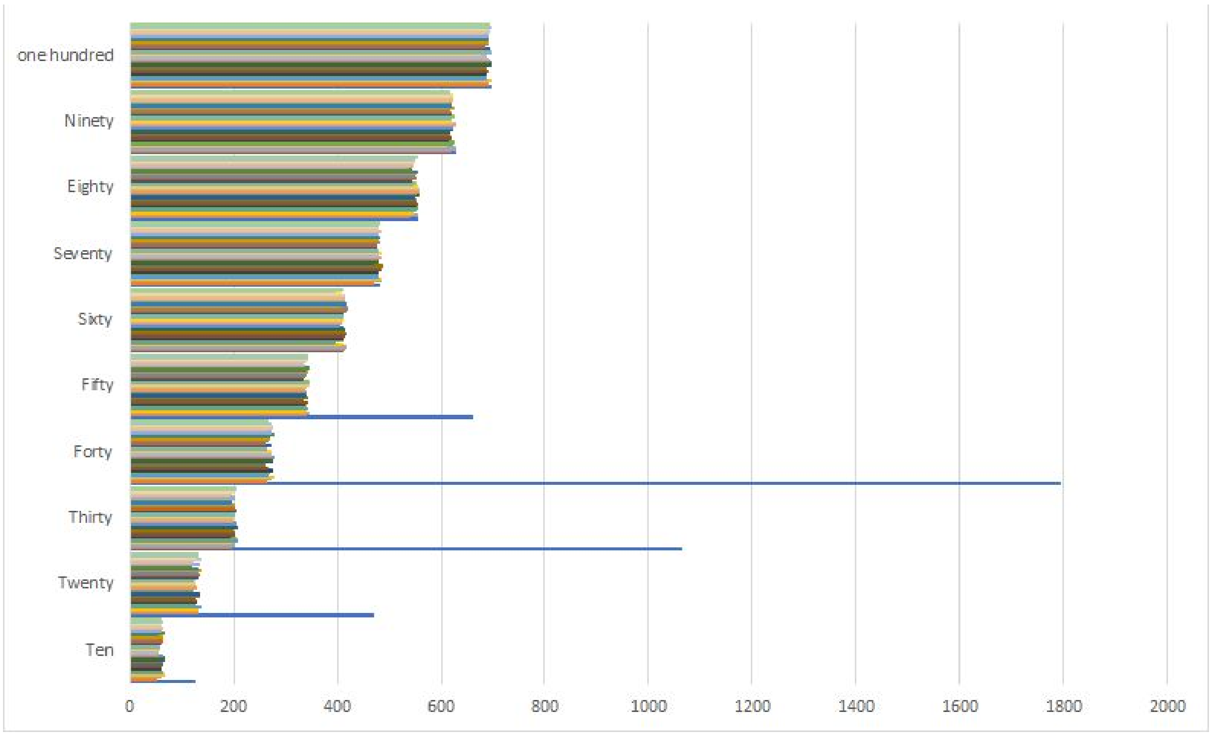
Practical examples of Linked Lists include the buckets in the Radix LSD Sorting Algorithm and more. Although the amount of buckets is fixed in the Radix sorting algorithm, the amount of elements in the buckets changes a lot. In fact after each iteration they get completely wiped. "Deleting" a linked list is as easy, apart from memory management, as pointing the first pointer to a non-value (such as NULL or in this assignment -1). Anything where inserting in the middle of the list, or ease of deletion, and little need to access a random element are all great use case scenarios for the linked list.

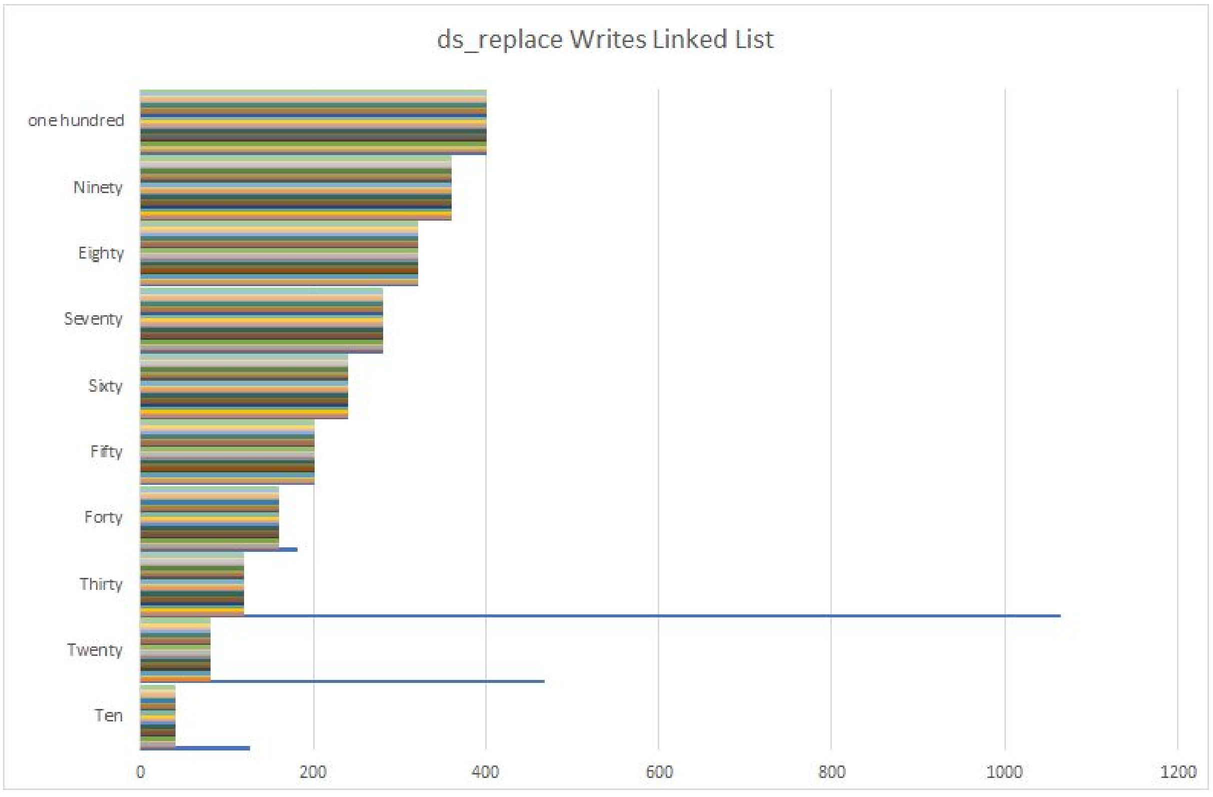
Note on the graphs

The data collection on the graphs is flawed. Most notably there is an incorrect number of reads and writes for the first entry into the ten, twenty, thirty, forty (and sometimes higher) bar.

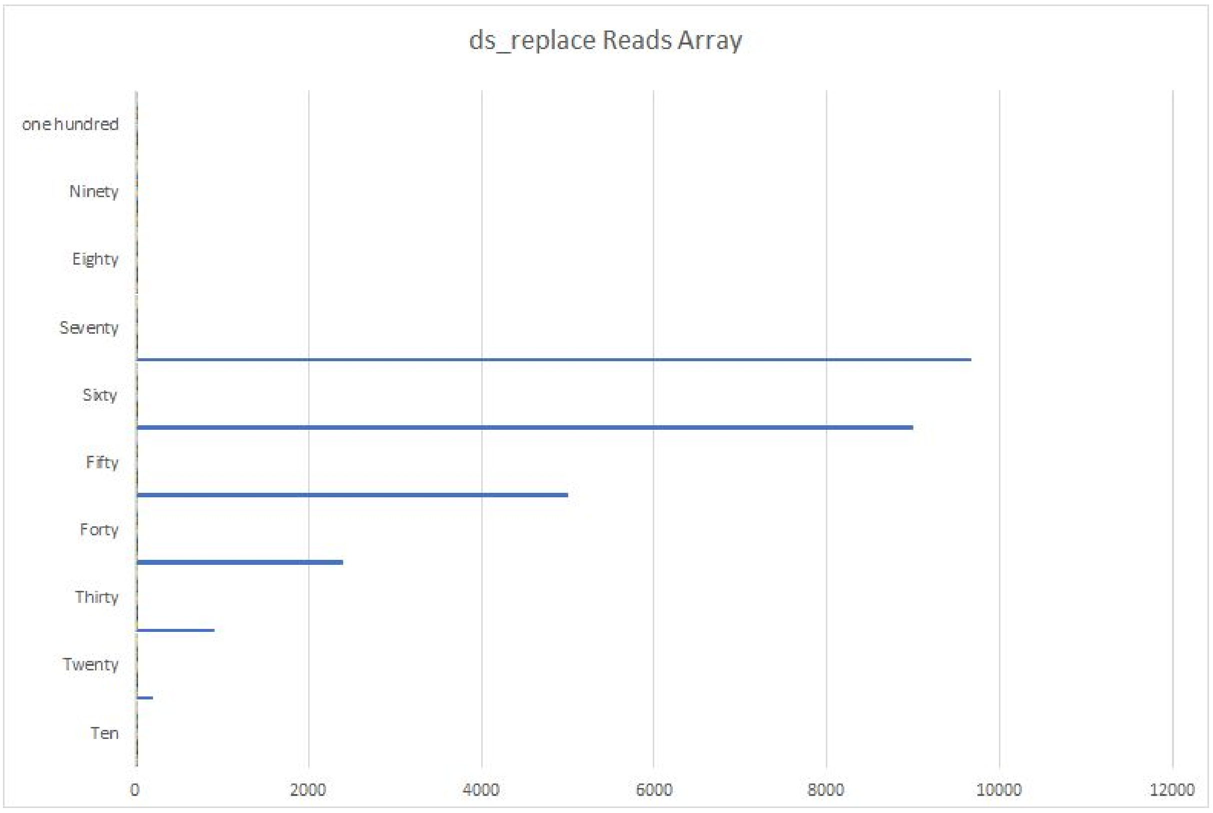
ds_replace Linked List

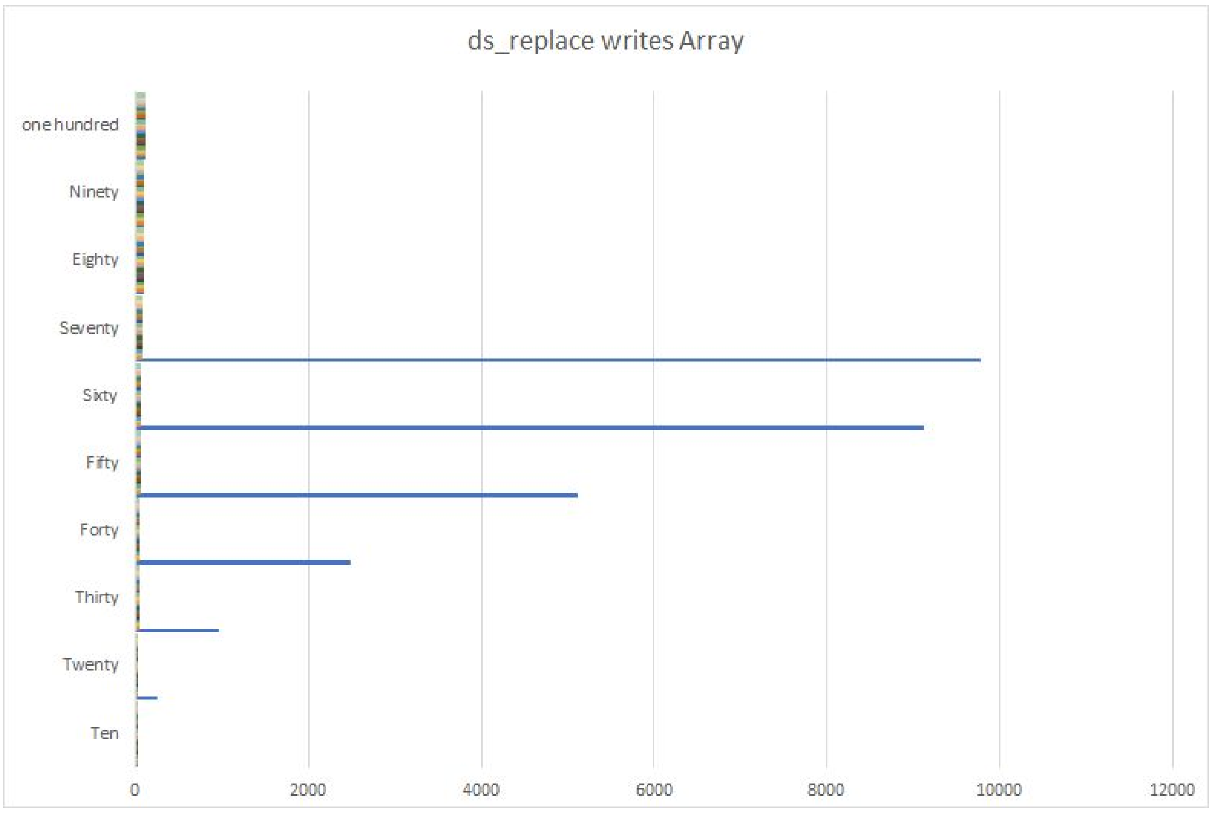
Ds_replace Linked List - reads



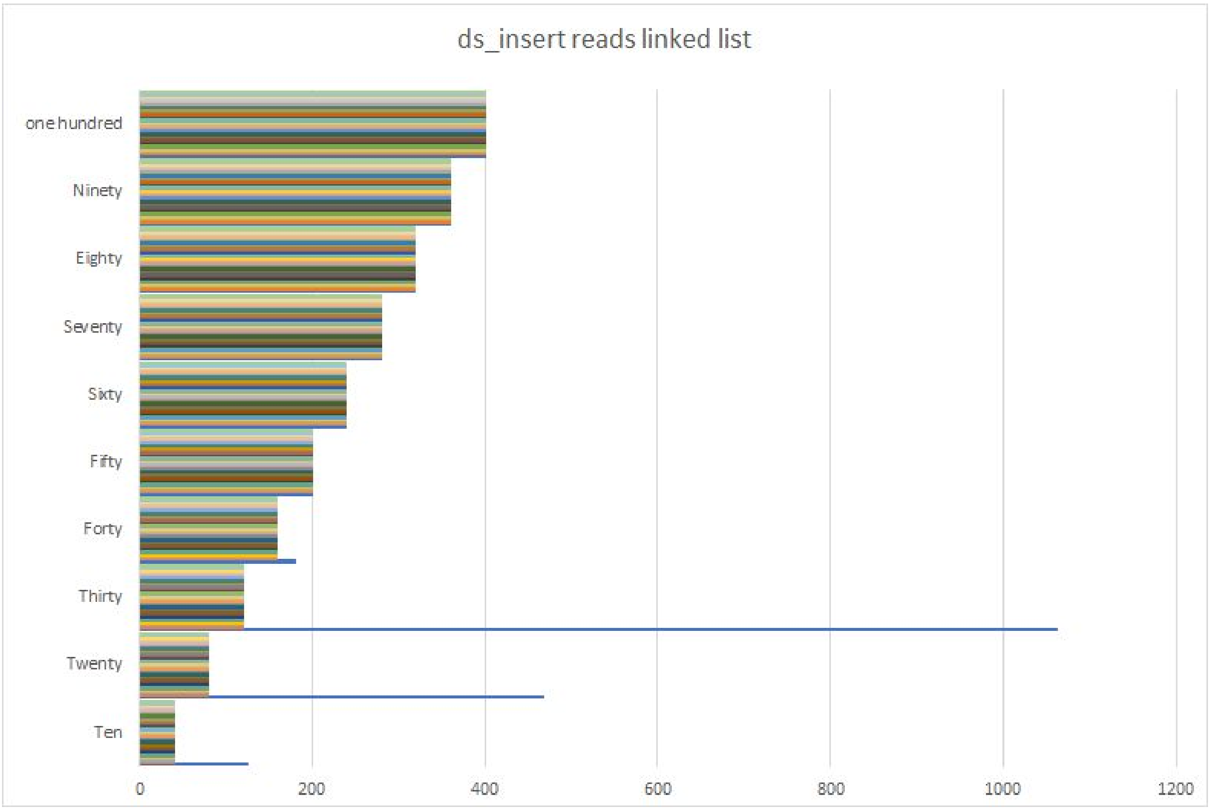


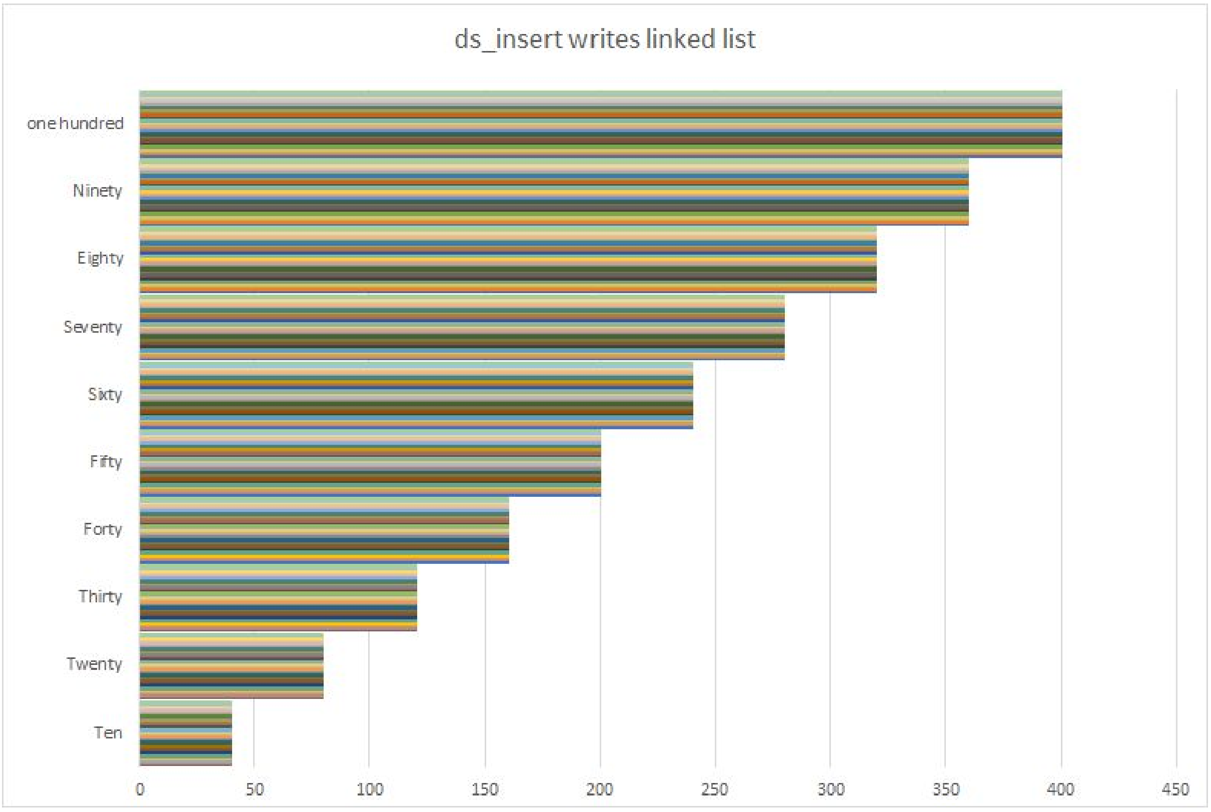
ds_replace Array



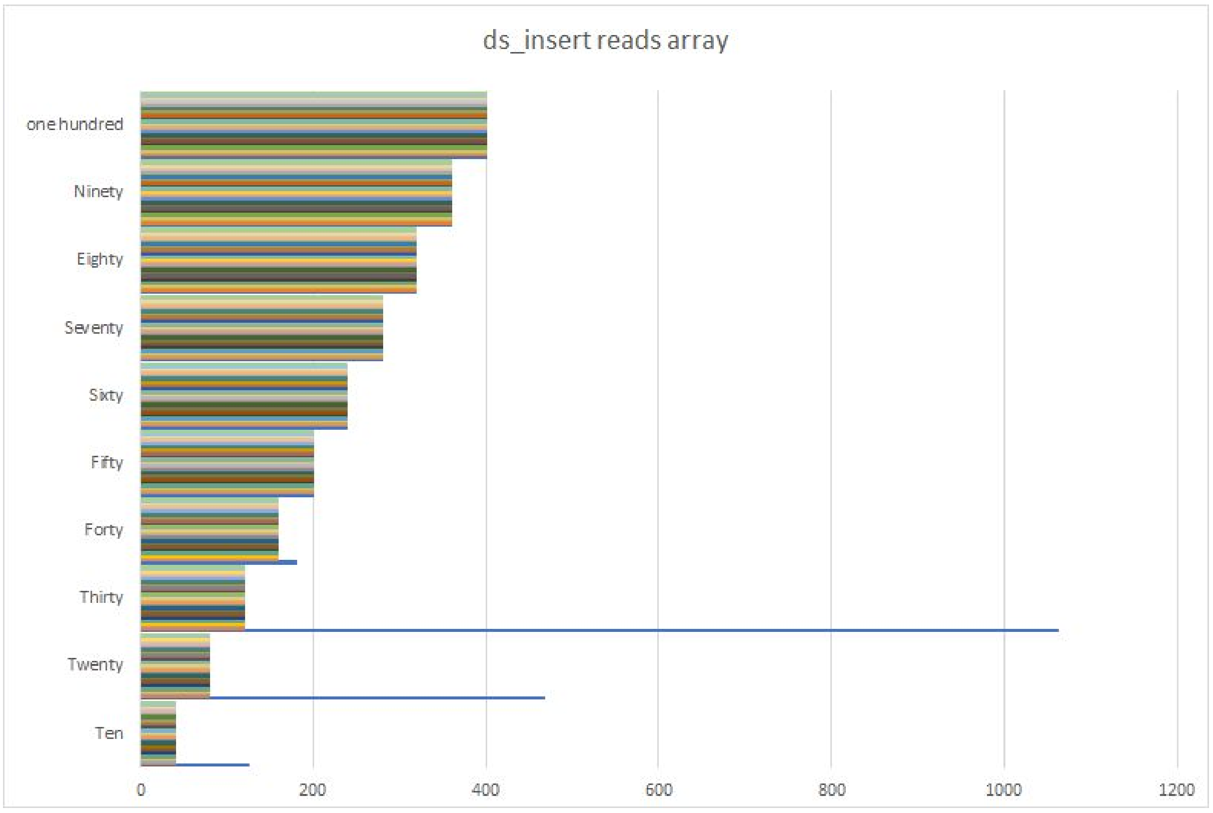


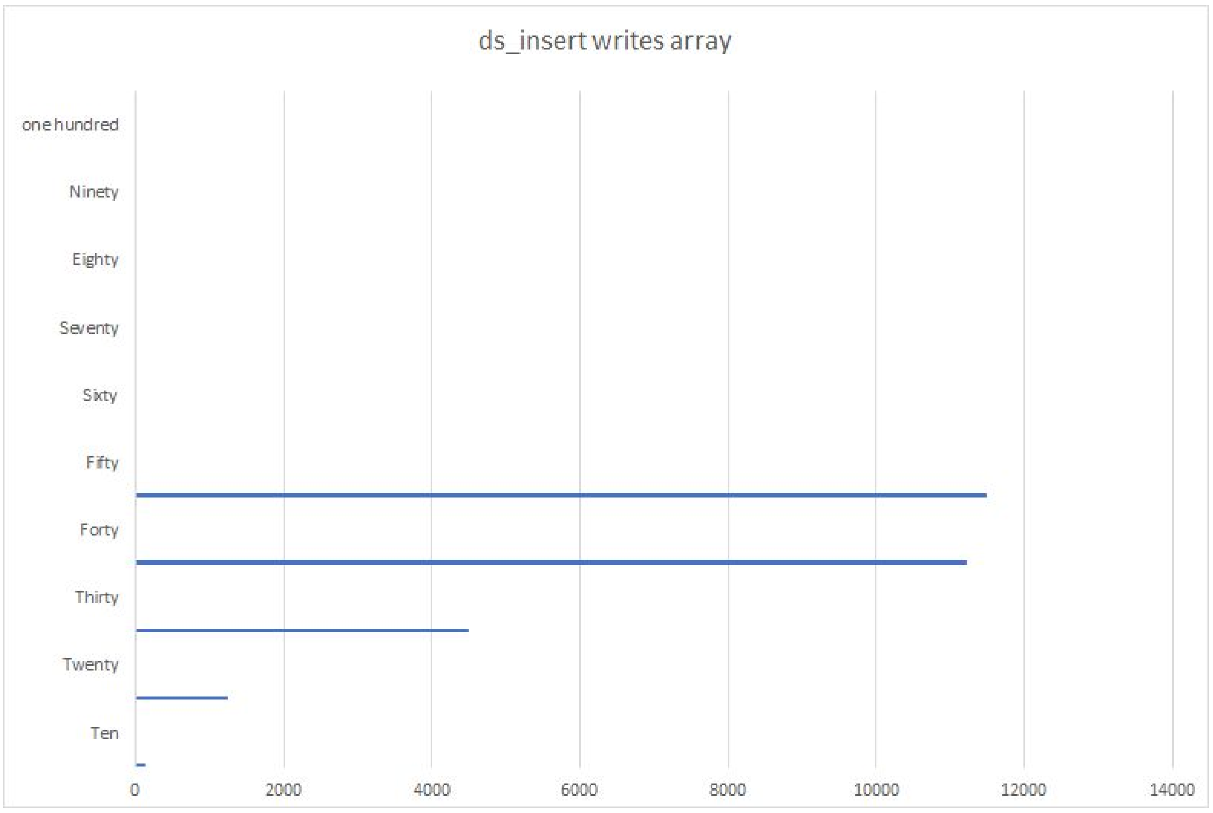
ds_insert Linked List





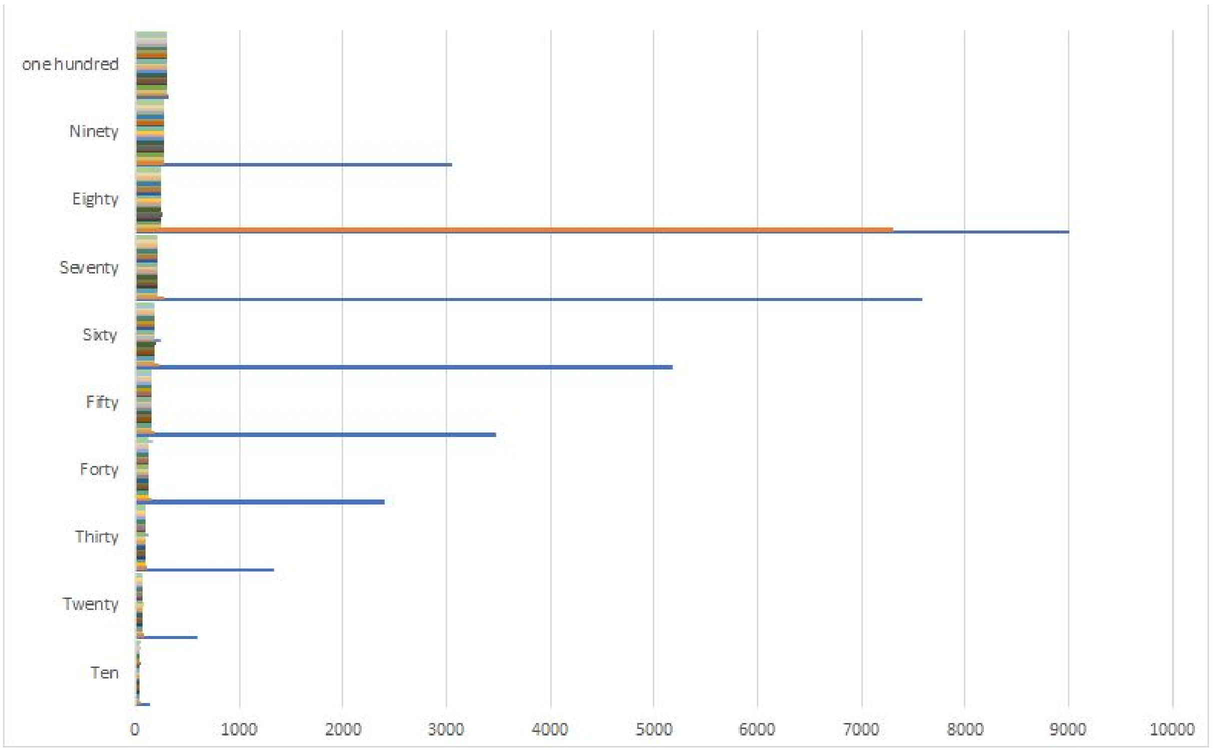
ds_insert Array



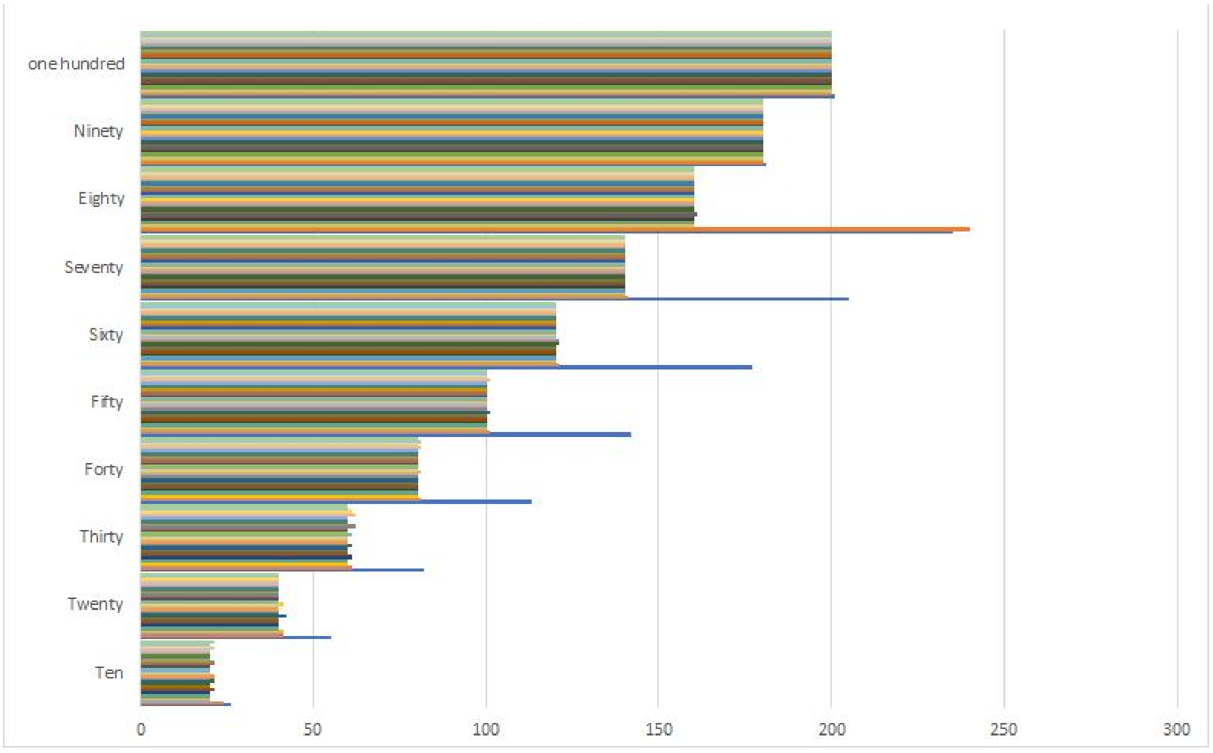


ds_delete Linked List

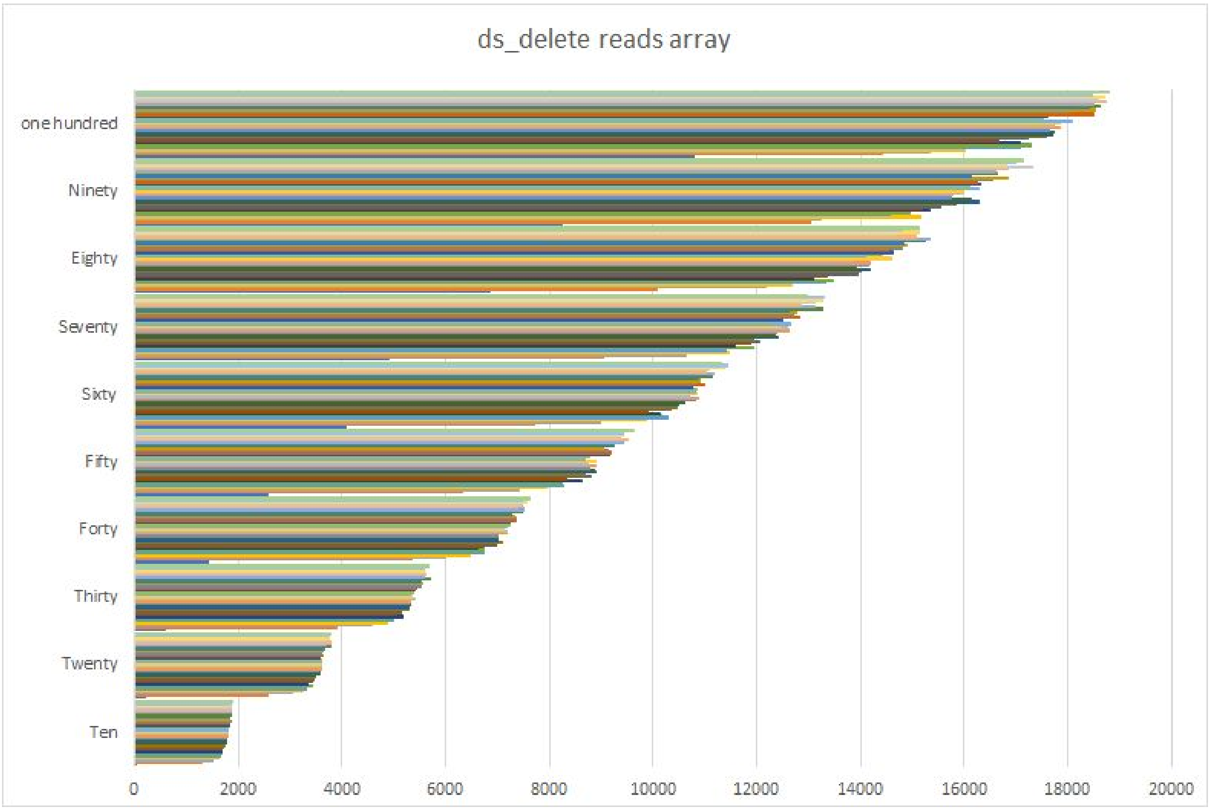
Ds_delete Linked list - reads

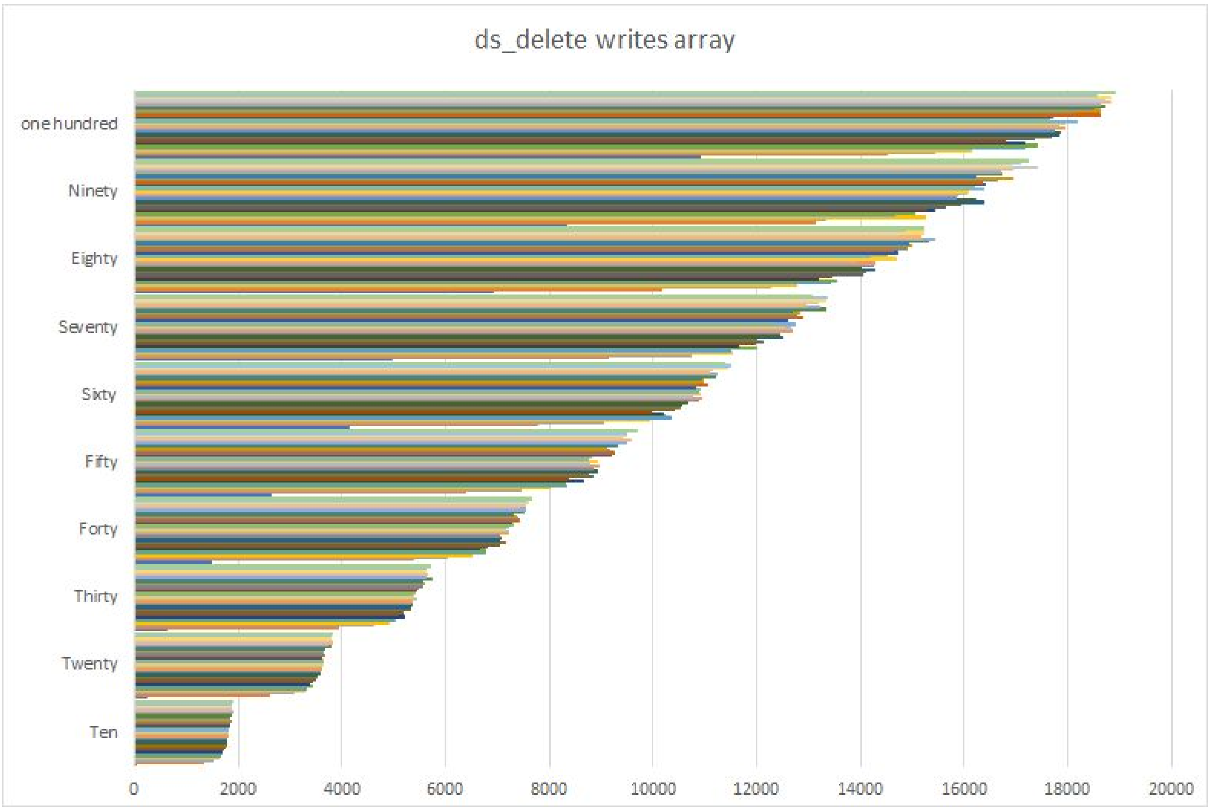


Ds_delete Linked list - writes

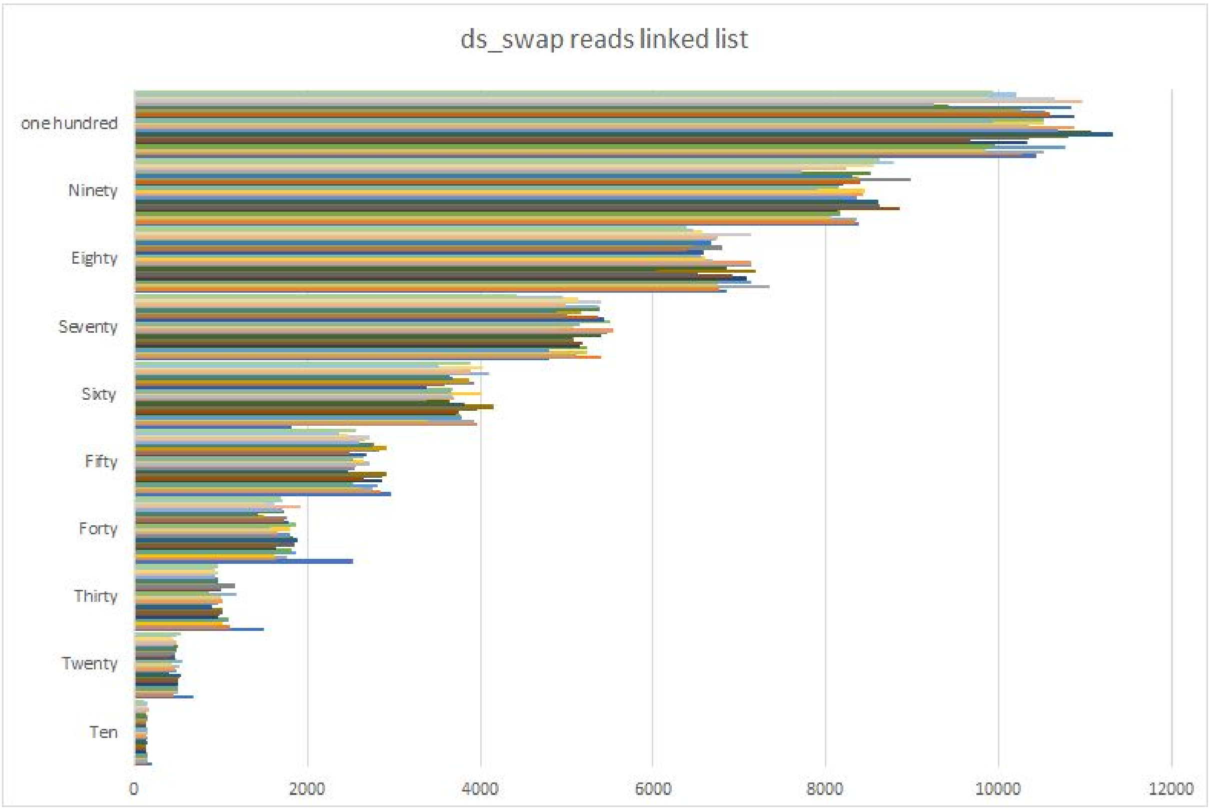


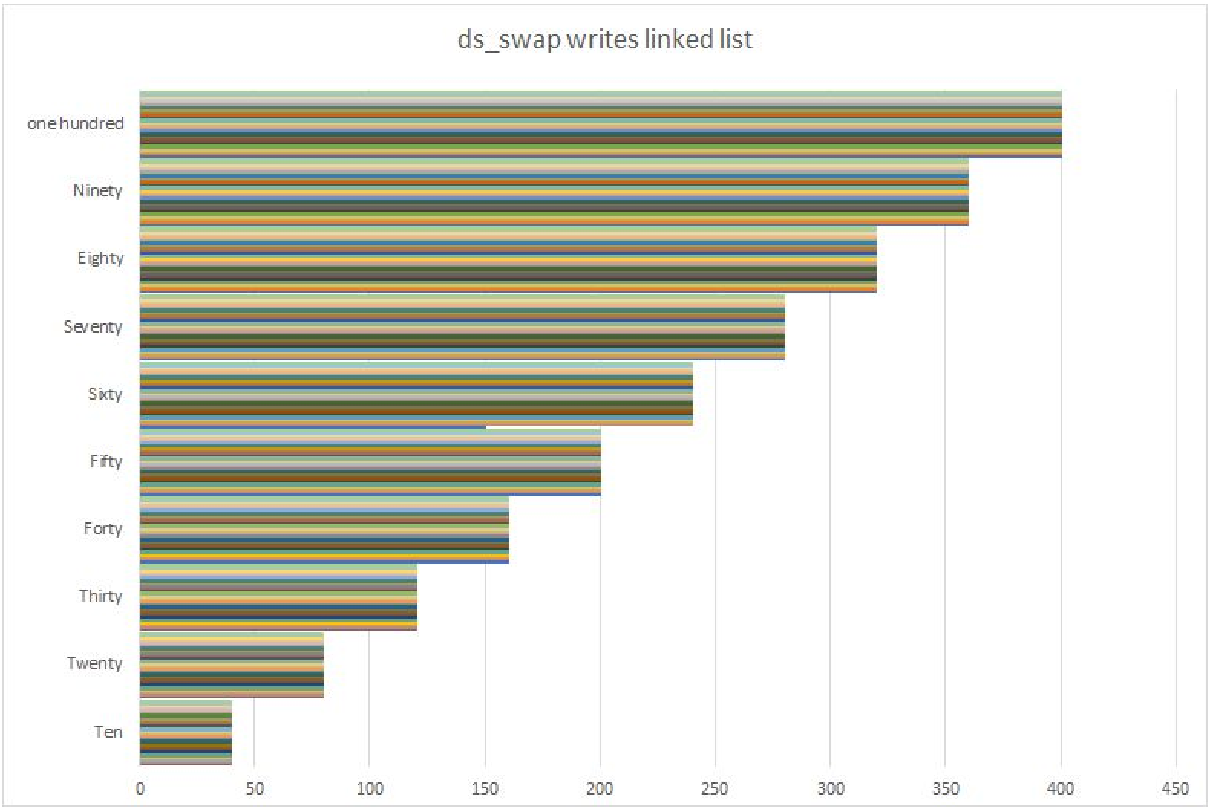
ds_delete Array



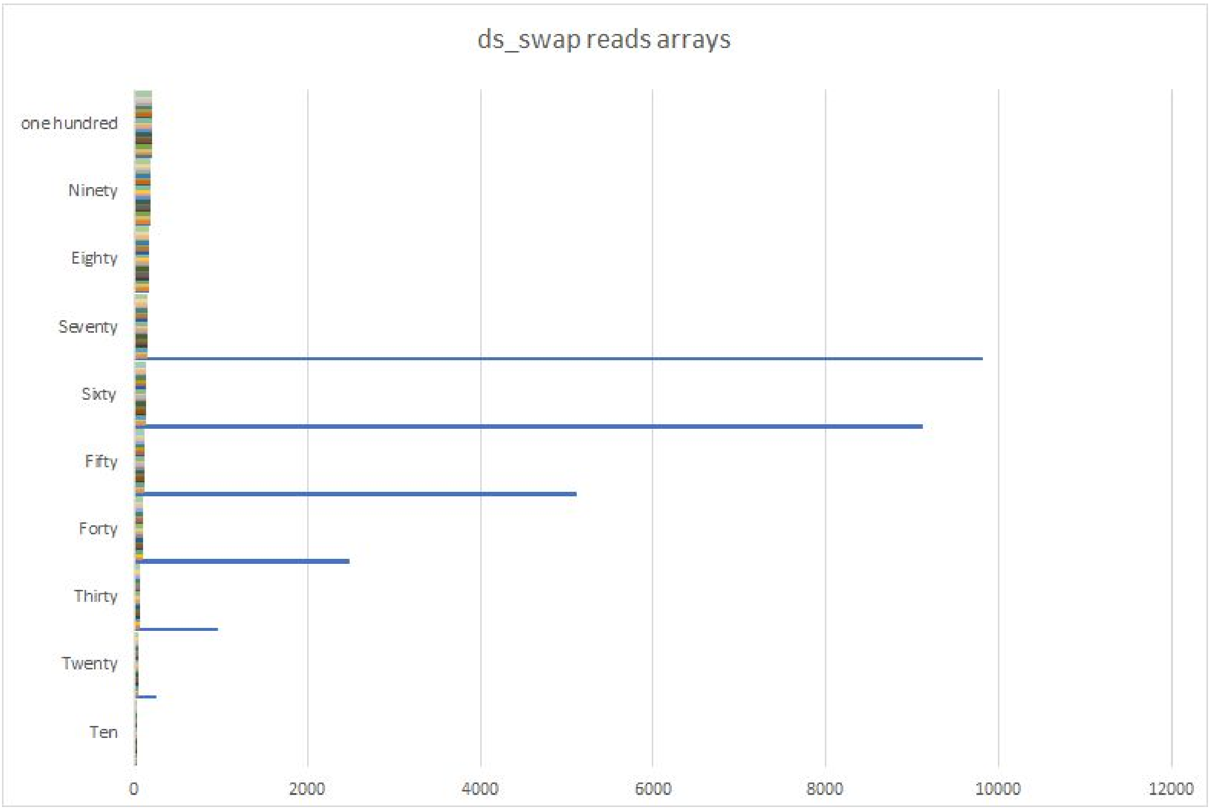


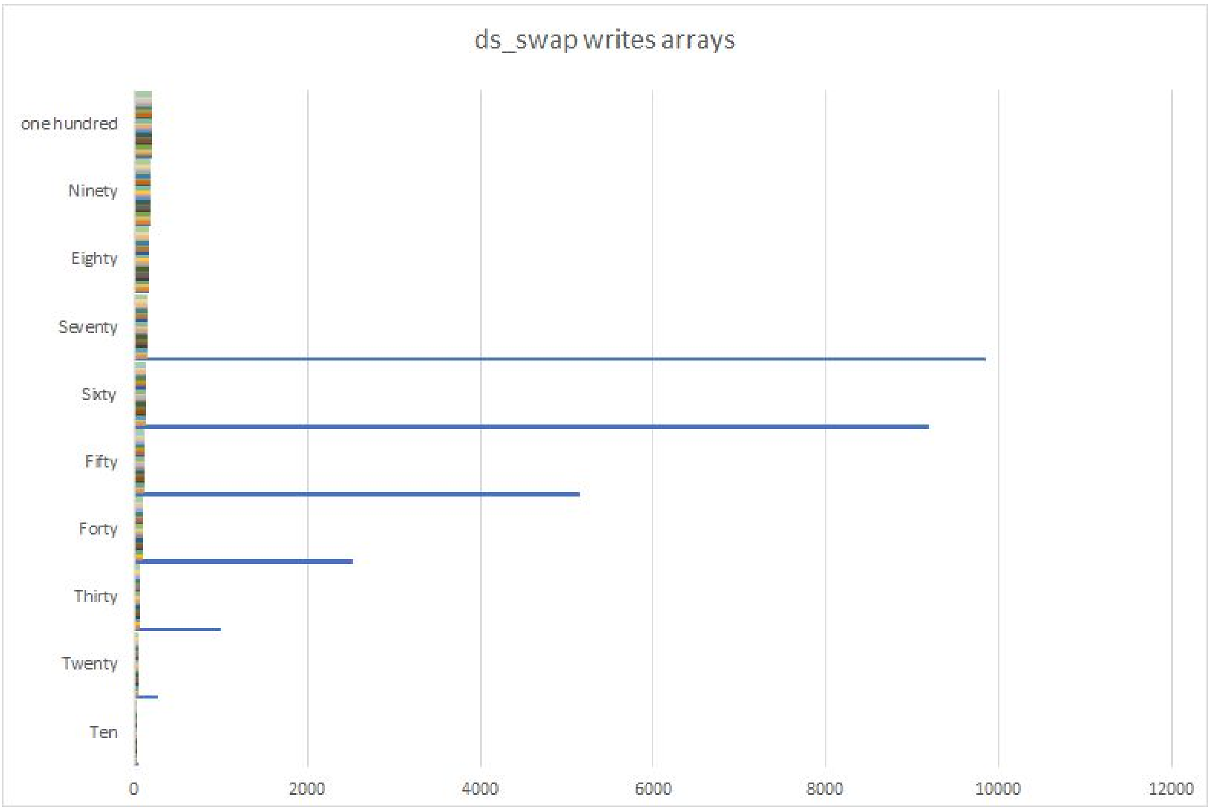
ds_swap Linked List





ds_swap Array


ds_swap reads arrays
one hundred
Ninety
Eighty
Seventy
Sixty
Fifty
Forty
Thirty
Twenty
Ten


ds_swap writes arrays
one hundred
Ninety
Eighty
Seventy
Sixty
Fifty
Forty
Thirty
Twenty
Ten