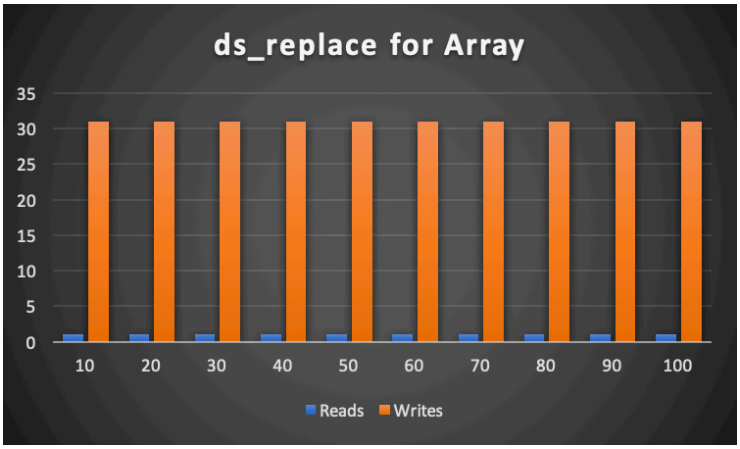
Adam Sheeres-Paulicpulle
1036569
asheeres@uoguelph.ca

Replacing at an Index

From the graphs, the use of an array when replacing nubers is more efficient. This is beause the array does not have to traverse the list to replace at a certain index. The Linked List, however, must read each element as it traverses through the list. This increases the number of reads and writes. Even if the list only had 10 elements in it, the array is still superior as it only writes one more than the desired number of replaces.



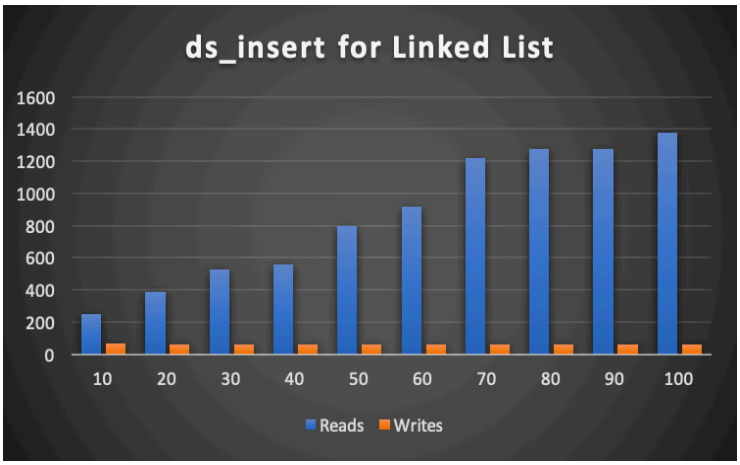
The linked list has to read each element and write to it only once (to replace that number), which is why the number of reads on the list is greater. The list must also read the head from the beginning of the file, whereas the list has a global variable of the number of elements, so it will not have to read the number of elements. Only when initializing the array will it have to read the elements and put it in the global variabel.

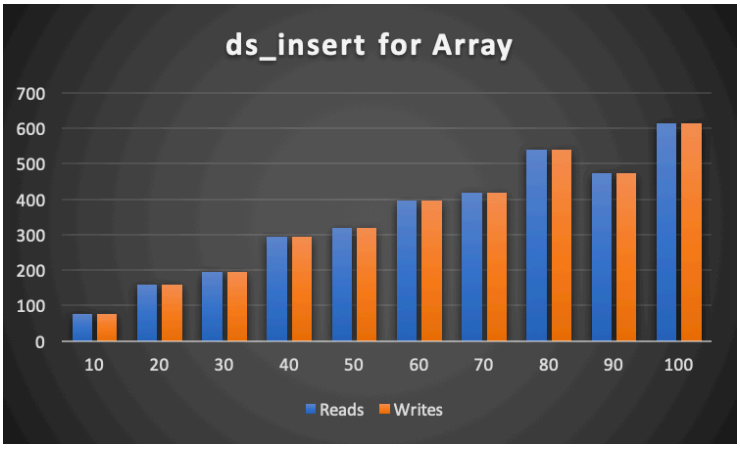
It is important to note that the number of writes in the Linked List is always below 30, whereas the number of writes for the Array is a constant 31.

In conclusion, because the number of reads for the Linked List far exceeds the combined reads and writes for the Array, the Array would be more efficient for a smaller or larger number of elements.

Adam Sheeres-Paulicpulle
1036569
asheeres@uoguelph.ca

Inserting at an Index





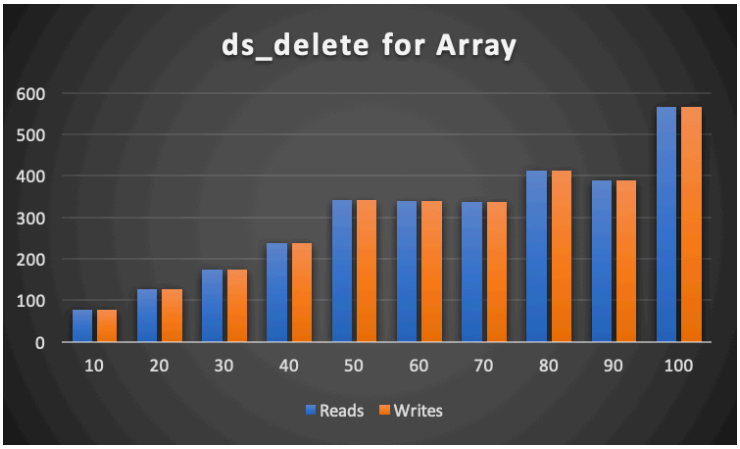
As shown in the graphs, the Array is once again more efficient than the Linked List. This is because when traversing through the list to find the proper index, the list must read every element before the index. For example, if there were one hundred elements in the list and the user wanted to insert an element at index 99, the list would have to traverse 98 times (read 98 times) until it gets to where it needs to be. However, with the Array, the program would only have to jump to the index and then re-write all subsequent elements after it. On the contrary, if the user wanted to insert something at element 0 of the Array, it would have to rearrange itself by re-writing all the elements one index above what they were. Whereas the linked list would simply allocate space for a new element and change the head to equal the element. The list would only have to read and write a couple of times, whereas the Array would have to read and write many times.

In conclusion, If the user only every desired to add elements at index 0, the Linked List would be more efficient. However, if the user wished to input elements near the end of the elements, an Array would be more efficient.

Adam Sheeres-Paulicpulle
1036569
asheeres@uoguelph.ca

Deleting at an index

Deleting elements in an array is quite easy, the program must go to the index and then delete the cell. However, it must also re-write the following numbers to the index before. For smaller numbers, the total read and writes for a Linked List is lower than an Array. The Linked List is more efficient than the Array if the number is small. In my findings, this occurred when there were only 10 elements.



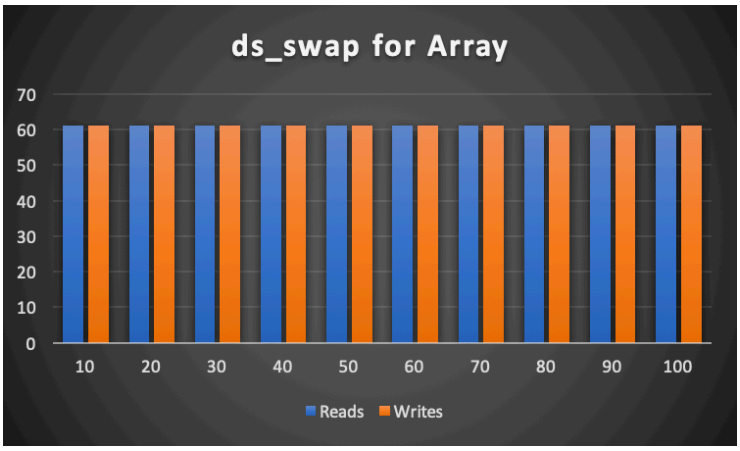
With larger numbers, the Array is more efficient than the Linked List. The number of reads and writes is the same for the Array because the program must read the old value in the index above it, and write it to the correct index. The Linked List, once again, must traverse the list until it is at the desired element.

If the user only wished to delete elements from index 0, the Linked list would be more efficient. This is because the Linked List would only have to re-write the head value and free the memory of where the element was stored. The Array would have to re-write the entire list of elements to the index just below them.

In conclusion, the Array is more efficient when dealing with a larger number of elements. However, the Linked List is more efficient if the user only wanted to delete elements at index 0, or if the number of elements is small.

Adam Sheeres-Paulicpulle
1036569
asheeres@uoguelph.ca

Swapping at an Index



While in an array, swapping elements is fairly easy as you only have to read and write twice each time you want to swap. However, for the Linked List, the program must first traverse the list to find the correct index and then read and write to the file. Since the Array does not need to increase or decrease the overall length of itself, the number of reads and writes will always be the same.

It is important to note, however, that the number of writes is very small, and the number of reads goes up in a fairly linear fashion for a Linked List. Moreover, while the number of writes is always 61 for the Array, the number of writes for the Linked List is almost always less than 60. When dealing with 10 numbers, it is only 30.

In conclusion, since the number of reads and writes is steady for the Array, the Linked List falls short in this scenario. If the user were to be dealing with a very large number of elements, the Array would be more efficient. If the user were to use only a few elements, the Array would still be more efficient unless the user wanted to store more data than just a single variable type.

Adam Sheeres-Paulicpulle
1036569
asheeres@uoguelph.ca

Conclusion

From the data, it is clear that the Array, in most cases, is more efficient than the Linked List. This is almost always because the Linked List must traverse itself before doing any reading or writing to the file. The most important aspect of this Linked List fault is to remember that a Linked List can store much more data than an Array. If the user only wanted a single variable type to be stored in each element, the Array, in most cases, would be more efficient. If the user wanted to store a multitude of data in few or many elements, the Linked List would be more efficient.

While inserting, deleting, and swapping, the number of reads and writes for the Array are the same. Since deleting is the opposite of inserting within the Array, this data makes sense. For swapping, all the program has to do is read at each index location and write at the opposite location. There will be two reads and two writes each time the user wishes to swap two elements.

In every test for the Linked List, the number of reads exceeds the number of writes by a large amount. This is due to the traversal of the list. However, it is important to note that the number of writes is almost always low. In some instances, such as the delete function, the number of writes for the Linked List is less than the number of writes for the Array. If the number of reads did not matter to the user, then a Linked List would be more efficient for all of the functions.

Due to the traversal of the Linked List, the Array seems to be more efficient in all cases. However, there are some key points in the data that are important to note:

- The number of writes is almost always lower in the Linked List
- Linked Lists can hold more data than a single element in an Array
- Most of the reads for the Linked List come from the traversal of the list

Lastly, for typical use, an Array is more efficient. For more complex use, a Linked List is better due to the amount of data it can hold. The one downside of using a Linked List is the number of reads is fairly large.