Experiment-

My experiment aims to notice the difference between time taken to traverse an unrolled linked list, doubly linked list and simple array.

OSULL-

To experiment my OSULL I create an OSULL named osull and I find the last element which traverses each and every node and consecutively each and every array within the node.

OSLL-

To experiment my OSLL I create an OSLL named osll and I find the last element which traverses each and every node and goes until back.

ARRAY-

To experiment my array, I create a simple dynamic array and I find the last element. Doing this forces a traversal till the last element.

OBERSERVATION-

After rigorously running my experiment, I observe that my OSULL is faster than OSLL. There are some reasons for this. I observed that when I take my nodeCapacity to be higher, I get more time difference. This can be because of cache misses. When I have a larger nodeCapacity then more elements are being stored in my OSULL and when I have to traverse, the next element is already in the cache and because of that there will be no cache miss. If we compare this with traditional linked list, it is not the same case. When using linked list, only that particular node and the data value inside the node are inside the cache and hence

when it searches for the next node, there will be a cache miss and it would potentially take more time.

Some data as to how much time it took to run different programs-

(all programs were run on SFU CSIL LINUX MACHINES WITH INTEL i9 13900 processor with a cache size of 36 mb)

PARAMETERS	ARRAY	OSULL	OSLL
SIZE = 100,000,000 NODECAPACITY=2,000	47.2725 ms	65.5095 ms	200.655 ms
SIZE = 20,000,000 NODECAPACITY=2,000	9.85654 ms	14.8807 ms	43.3514 ms
Size = 40,000,000 NODECAPACITY=4,000	17.7812 ms	24.259 ms	85.8562 ms
Size = 100,000 NODECAPACITY=10	0.071 ms	0.11 ms	0.35 ms

The cache size is 36mb hence after constructing a 9 million size list when we traverse, we don't get a cache miss as the entire list fits into the cache. Hence to achieve the goal of the experiment i.e. to measure the time taken to traverse the list when not significant part of the list is in cache and to do this, I constructed lists which are very big in size and do not fit in the cache entirely.

The last case is when the list does fit entirely in the cache and there less cache misses hence traversal is very fast in that case.

INTERESTING OBSERVATIONS-

While designing the remove function in the OSULL class, I came across some conditions where after repeatedly removing items from the front node, after a point once it reaches size == 4, the front dummy node points to second node. This happens because we always check the size of the previous array to either steal or merge. To overcome this case, I designed two more conditions. When we are at first node and the size of first node becomes less than, then it will either steal from the second node if the size is greater than half and merges if the size of next node is less than or equal to half. This way I am able to have an optimum running code where all the nodes are at least at half the capacity.