1. After implementation, the linked list would take more memory. The dynamic array's implementation is the size of the pointer type(int, double,...) plus the byteSize*n element.
On the other hand, the linked list needs the struct and the pointer for the head. The struct might need padding, the pointer will need its own memory setup, so we have more
parts in an linked-list compared to the Array already.

2. I would think the first element of both of them would take almost the exact amount of time. Maybe the array will be a tad bit faster, since memory will cache the cell faster and array does not need the head
to keep track of where it is.

3. I would expect the Linked-List(LL) to be slower. In a worst case scenario, Big-O notation, the LL needs to, step by step, go across n-1 elements until it reaches the n'th element. (So O(N)) The array
is very close to hardware (in terms of implementation) the memory will already cache certain cells in the array, and I would think it would take O(1) since it's a matter of simple pointer arithmetic. So,
the array would access the 9th element faster.

4. I would expect the linked list to be much faster. The value in the array must be individually pushed up, so that we can make space for the element that we are going to add in the begining. So in the
worst case scenario, it would be an O(N). While the LL is a matter of changing where the head points to first (while keeping track of the rest of the LL nodes so we don't lose them) so it's much easier.

5. I would expect array to be faster. Again, it's a matter of pointer arithmetic, and arrays are more natural to memory and Assembly than linked list are. The insertion at the end for the LL would be a matter
of O(N). In the worst case scenario, it has to go through _n_ nodes in order to add a new element at the end.