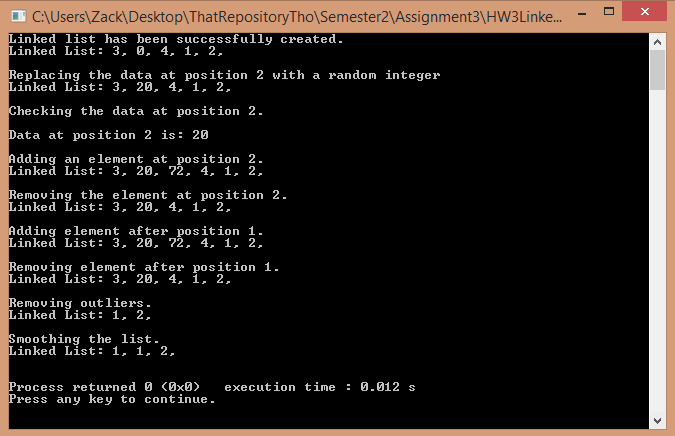
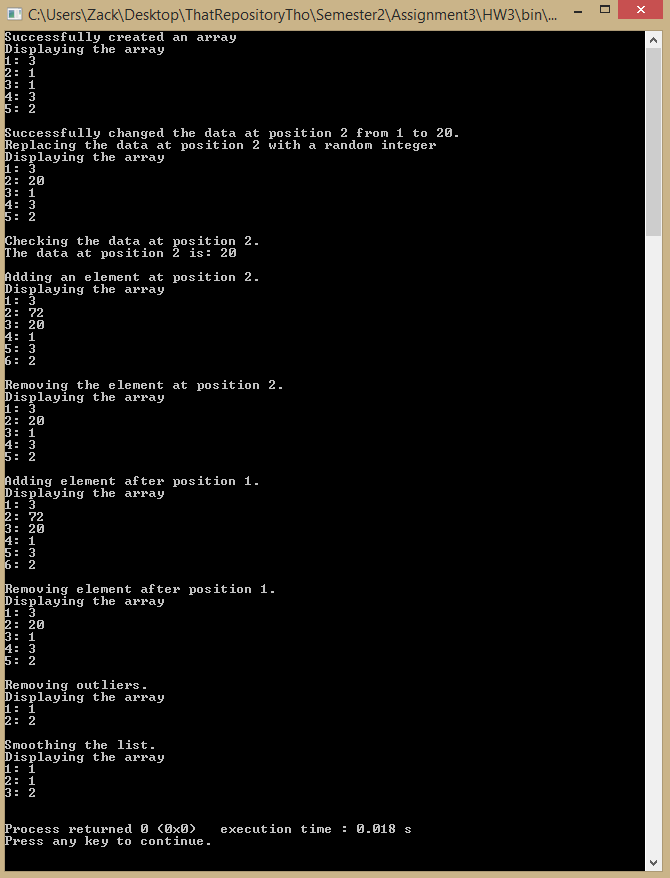
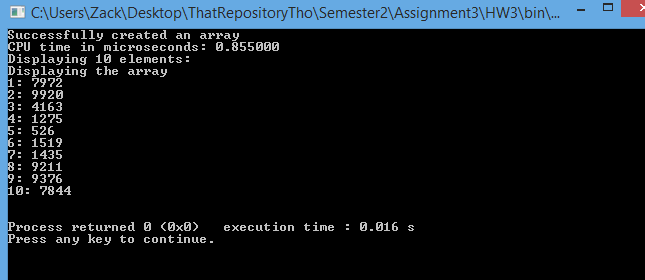
Test output with 5 elements(Linked list):



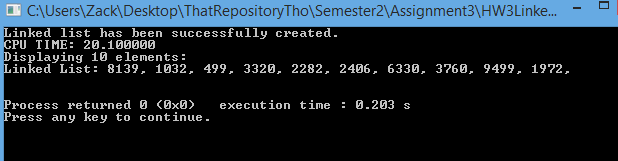
Test output with 5 elements(Array):



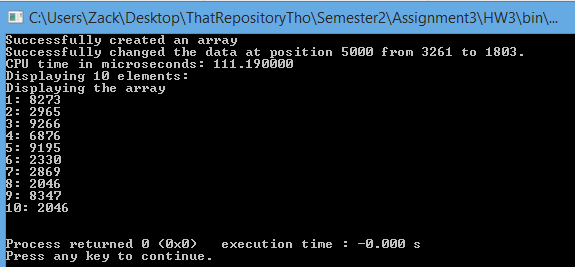
Testing CPU time for Array(checking element):



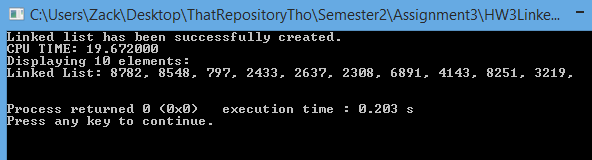
Testing CPU time for Linked list(checking element):



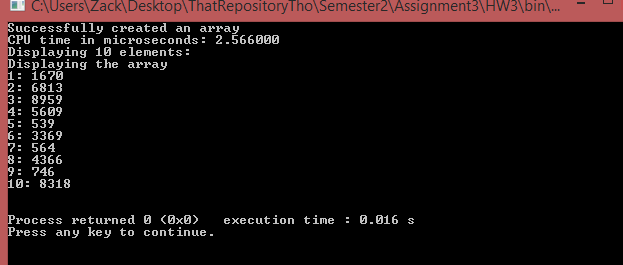
Testing CPU time for Array(replacing data):



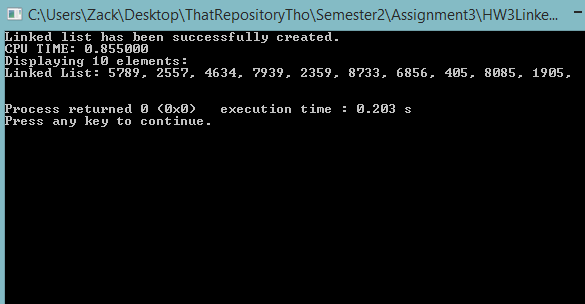
Testing CPU time for Linked list(replacing data):



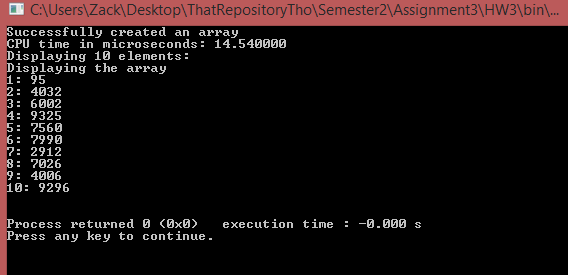
Testing CPU time for Array(adding at the beginning):



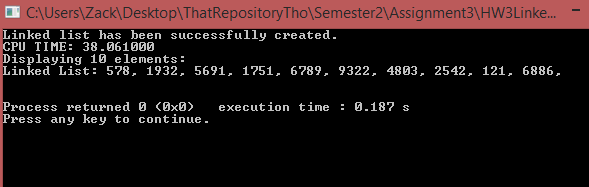
Testing CPU time for Linked list(adding at the beginning):



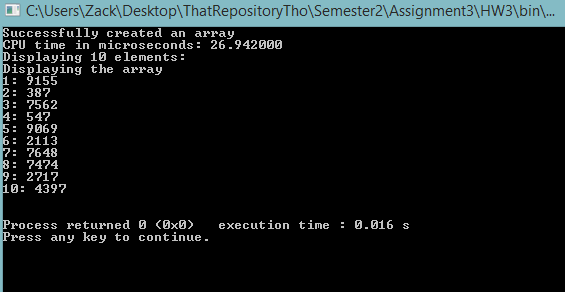
Testing CPU time for Array(adding at the middle):



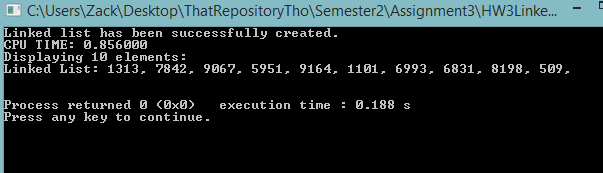
Testing CPU time for Linked list(adding at the middle):



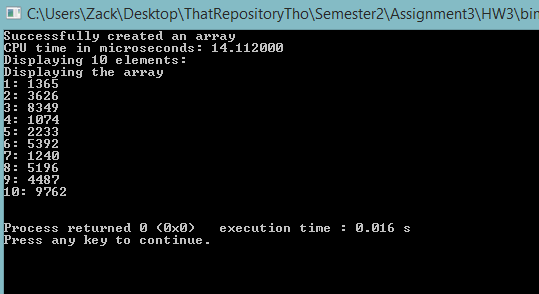
Testing CPU time for Array(removing at start):



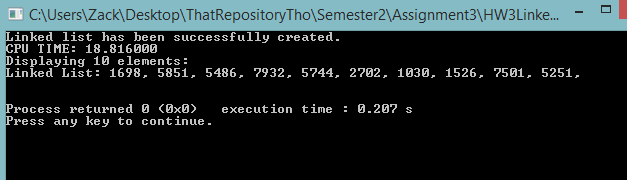
Testing CPU time for Linked List(removing at start):



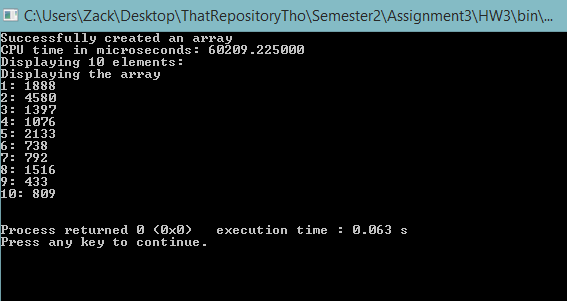
Testing CPU time for Array(removing at middle):



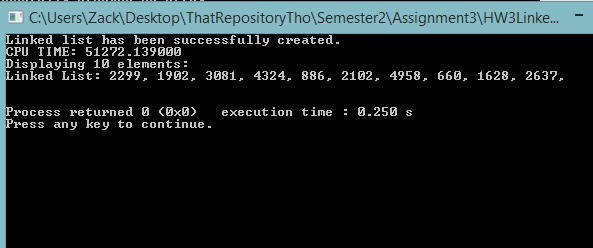
Testing CPU time for Linked List(removing at middle):



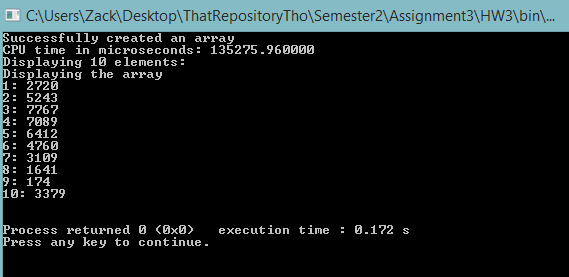
Testing CPU time for Array(removing outliers):



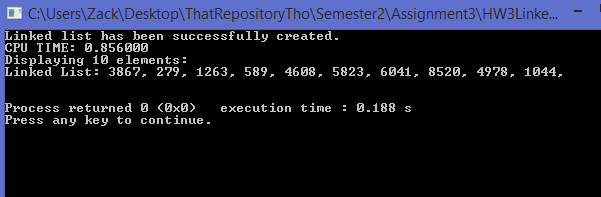
Testing CPU time for Linked list(removing outliers):



Testing CPU time for array(smoothing array):



Testing CPU time for Linked list(smoothlng list):



|  |  |
| --- | --- |
| n= 10,000 | **Check the data in the middle(microseconds)** |
| array | O(1); time= 0.855 |
| linked list | O(n); time= 20.1 |

|  |  |
| --- | --- |
| n=10,000 | **Replace the data in the middle(microseconds)** |
| array | O(1); time= 111.19 |
| linked list | O(1); time= 19.672 |

|  |  |  |
| --- | --- | --- |
| n=10,000 | **Add the element/node at the start(microseconds)** | **Add the element/node at the middle, after the previous element/node(microseconds)** |
| array | O(1); time=2.566 | O(1); time= 14.54 |
| linked list | O(1); time=0.855 | O(1); time= 38.061 |

|  |  |  |
| --- | --- | --- |
| n=10,000 | **Remove the element/node at the start(microseconds)** | **Remove the element/node at the middle, after the previous element/node(microseconds)** |
| array | O(n); time= 26.942 | O(n); time= 14.112 |
| linked list | O(1); time= 0.856 | O(n); time=18.816 |

|  |  |
| --- | --- |
| n= 10,000 | **Iterate over all the elements and remove the ones with data bigger than min(RAND\_MAX, n)/2. (microseconds)** |
| array | O(n); time= 60,209 |
| linked list | O(n); time= 51,272.139 |

|  |  |
| --- | --- |
| n=10,000 | **Iterate over all the elements and smooth it inserting an average between each pair of elements in the original linked list(microseconds)** |
| Array | O(n); time= 135,275.96 |
| linked list | O(n); time= 0.856 |

Comments:

I found that when adding or removing more than one thing, linked lists were faster. This is because with linked list you don’t have to shift everything so it doesn’t take very long. Arrays are faster at looking through data and replacing data.