Roman Millem
Assignment 2
September 28, 2018

Problem 1: Ordering the following functions by growth rate from least to greatest:

1. $\frac{2}{N}$
2. 37
3. $\sqrt{N}$
4. N
5. $Nloglog(N)$
6. $Nlog(N)$ **and** $Nlog(N^2)$ **grow at the same rate**
7. $Nlog^2(N)$
8. $N^{1.5}$
9. N^2
10. $N^2log(N)$
11. N^3
12. $2^{\frac{N}{2}}$
13. 2^N

Problem 2:

This problem asks to write a procedure that returns the common elements in the two given sorted lists, L1 and L2.

My routine uses an iterator for each list and a new list for the common elements. I utilize a nested loop to loop through L2 for each element in L1. This allows the routine to find a common element if the lists are different sizes. If the values that each iterator points to are equal then that value is pushed to the back of the common elements list.

```
1
2   template <typename list>
3
4   //Procedure that iterates through two sorted integer lists and returns a integer list of common elements
5   list commonElements (list L1, list L2){
6
7       //Initialize the list of common elements
8       list common;
9       //Declare and iterator for each list
10      auto itr1 = L1.begin();
11      auto itr2 = L2.begin();
12
13      //Loop through first list
14      while(itr1 != L1.end()){
15
16          //Loop through second list
17          while(itr2 != L2.end())
18          {
19              //Check to see if common element
20              if(*itr1 == *itr2)
21
22                  //Adds value to the back of common element list
23                  common.insert(L1.end(),*itr2);
24
25              //i=Increment L2 iterator
26              ++itr2;
27          }//End L2 loop
28
29          //Increment L1 iterator
30          ++itr1;
31
32          //Sets L2 iterator to beginning of L2
33          itr2 = L2.begin();
34
35      }//End L1 loop
36
37      //Returns list of common elements
38      return common;
39  }
40
```

Problem 3:

The purpose of this problem is to write an algorithm for printing a singly linked list in reverse, using only constant space.

Problem 3 is all about nodes. A node is composed of a data point, and a pointer to the next node. So in order to reverse a singly linked list, I used 3 nodes. One node to represent the current position, starting at the head, a second to represent the position before and a third to represent the position after; both with initial values. I iterated through the list until the 'current' node reached the end. Each iteration consists of incrementing 'next', linking 'current' to 'previous', incrementing 'previous' followed by incrementing 'current'. After the loop, 'current' is linked to 'previous' one last time. This effectively causes the head to be at the end of the list linking to the previous node which links to the one before it and so on.

```
1   template <typename list>
2
3   void reverse(list& L){
4
5       //Assign element values to Nodes
6       //Node for iterative position, position previous, and position after
7       Node *current = head;
8       Node *next = NULL;
9       Node *previous = NULL;
10
11      //While loop to alter values of list elements
12      while(current != NULL){
13
14          //'next' is assigned the data of the node 'current' is linked to
15          next = current->next
16          //Node 'current' is linked to 'previous', reversing the pointer
17          current->next = previous;
18          //Increment 'previous'
19          previous = current;
20          //Increment 'current'
21          current = next;
22      }
23
24      //current is linked to 'previous', finalizing the reversal
25      current->next = previous;
26      head->next = current;
27  }
28
```

Problem 4:

According to the C++ standard, for the vector, a call to push back, pop back, insert, or erase invalidates (potentially makes stale) all iterators viewing the vector. Why?

Calls to push back, pop back, erase, and the like cause memory to be reallocated. The iterators pointing at the altered positions can be invalidated, potentially causing iterators pointing to them to be invalidated, and so on.

Problem 5:

Efficiently implement a stack class using a singly linked list, with no header or tail nodes.

I created a node struct to make a stack using a singly linked list. I used the nodes to dynamically modify the stack memory as the size is altered every push and pop.

```
#include <cstddef>
using namespace std;

template <typename Object>

class stack {
private:
    struct node{
        node() {
            next = NULL;
        }
        node(Object obj) : data(obj) {}                          //Node structure initialization list for data object
        node(Object obj, node* ptr) : data(obj), next(ptr) {}    //and for both data object and pointer
        Object data;
        node* next;
    };

public:
    stack(){
        head = NULL;                                  //Stack constructor to initialize head pointer and size
        SIZE = 0;
    };

    void push(Object obj){                            //Push implementation, create new node for first
        node* itr = new node(obj,head);               //added object and make it the head
        head = itr;
        ++SIZE;
    }
    void pop(){                                       //Pop implementation, assign new node the value of the
        node* itr = head->next;                       //node head is linked with. Delete head and set head to the
        delete head;                                  //created node
        head = itr;
        --SIZE;
    }

private:
    node * head;
    int SIZE;

};
```

Problem 6:

Run time:

(a) $O(N)$
(b) $O(N^2)$
(c) $O(N^3)$
(d) $O(N^2)$
(e) $O(N^5)$
(f) $O(N^4)$
(g) $O(logN)$

```
Function (A)

Time take by function O(N) at N = 16: 0 microseconds
Time take by function O(N) at N = 32: 0 microseconds
Time take by function O(N) at N = 65536: 0 microseconds
Time take by function O(N) at N = 131072: 998 microseconds
Time take by function O(N) at N = 262144: 997 microseconds

Process returned 0 (0x0)   execution time : 0.015 s
Press any key to continue.
```

```
Function (B)

Time take by function O(N^2) at N = 64: 0 microseconds
Time take by function O(N^2) at N = 128: 0 microseconds
Time take by function O(N^2) at N = 512: 997 microseconds
Time take by function O(N^2) at N = 1024: 2026 microseconds
Time take by function O(N^2) at N = 2048: 6977 microseconds

Process returned 0 (0x0)   execution time : 0.029 s
Press any key to continue.
```

```
Function (C)

Time take by function O(N^3) at N = 32: 0 microseconds
Time take by function O(N^3) at N = 64: 0 microseconds
Time take by function O(N^3) at N = 128: 5018 microseconds
Time take by function O(N^3) at N = 256: 31950 microseconds
Time take by function O(N^3) at N = 512: 221408 microseconds

Process returned 0 (0x0)   execution time : 0.277 s
Press any key to continue.
```

```
Function (D)

Time take by function O(N^2) at N = 64: 0 microseconds
Time take by function O(N^2) at N = 128: 0 microseconds
Time take by function O(N^2) at N = 512: 0 microseconds
Time take by function O(N^2) at N = 1024: 997 microseconds
Time take by function O(N^2) at N = 2048: 4021 microseconds

Process returned 0 (0x0)   execution time : 0.021 s
Press any key to continue.
```

```
Function (E)

Time take by function O(N^5) at N = 16 : 997 microseconds
Time take by function O(N^5) at N = 32 : 5019 microseconds
Time take by function O(N^5) at N = 64 : 172536 microseconds

Process returned 0 (0x0)   execution time : 0.197 s
Press any key to continue.
```

```
Function (F)

Time take by function O(N^4) at N = 16: 0 microseconds
Time take by function O(N^4) at N = 32: 5983 microseconds
Time take by function O(N^4) at N = 64: 169992 microseconds
Time take by function O(N^4) at N = 128: 5466374 microseconds

Process returned 0 (0x0)   execution time : 5.657 s
Press any key to continue.
```

```
Function (G)

Time take by function O(log N) at N = 256: 0 microseconds
Time take by function O(log N) at N = 512: 0 microseconds
Time take by function O(log N) at N = 131072: 0 microseconds
Time take by function O(log N) at N = 2^25: 0 microseconds
Time take by function O(log N) at N = 2^33: 0 microseconds

Process returned 0 (0x0)   execution time : 0.022 s
Press any key to continue.
```

These results line up well with my analysis. For example the growth rate for function E approaches 32 each time the sample size is double. So the growth rate matches with the theoretical growth rate of 2^5. Another example is how function A grows at a constant rate.