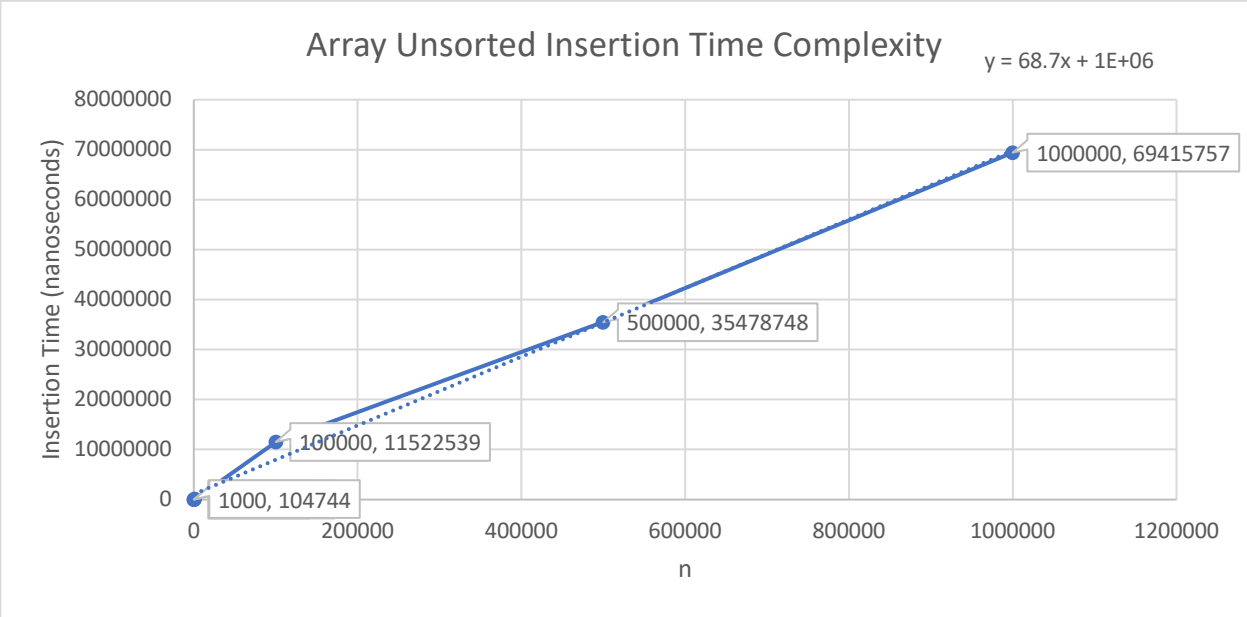
Eli Laird
CS 7350
Homework 1

Problem 1:

- n = {10, 100, 1000, 100000, 500000, 1000000}
- Upper Bound = $O(n)$
  - The linearity of the upper bound can be seen every time we increase n by a factor of 10, the time also increases by a factor of 10.
- Estimated time for n = 1 Trillion
  - Since the upper bound is $O(n)$, we can expect the value at n=100 to increase by a factor of 10 billion. Using this we can estimate the value of t for n = 1 Trillion to be n(100) * 10 billion = 1.1802e14 nanoseconds.

Array Unsorted Order	
n	t
10	2017
100	11802
1000	104744
100000	11522539
500000	35478748
1000000	69415757



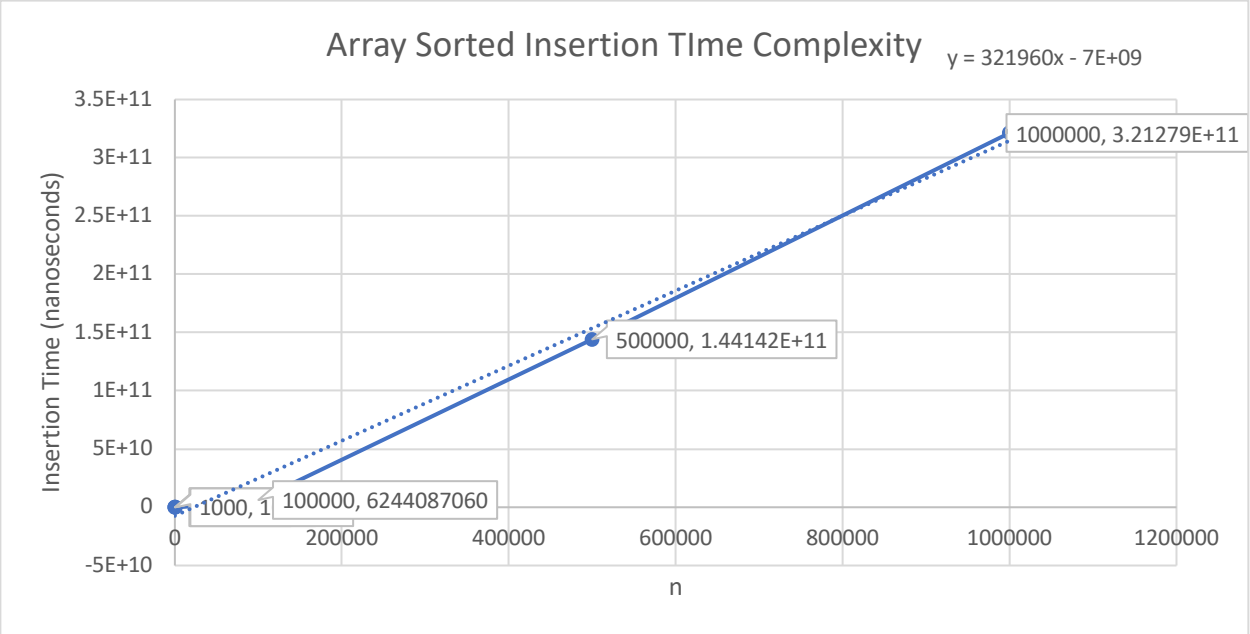
Code:

```
long long problem1(unsigned long int n, std::default_random_engine generator){

    using namespace std::chrono;

    //initialize random number generator
    std::uniform_int_distribution<unsigned long int> dist(1,n);

    //int array for storage
    unsigned long int arr[n];

    //take the current time
    high_resolution_clock::time_point start = high_resolution_clock::now();

    for(unsigned long int i = 0; i < n; i++)
        arr[i] = dist(generator);

    high_resolution_clock::time_point end = high_resolution_clock::now();
    auto total_time = duration_cast<nanoseconds>(end - start).count();
    std::cout << "n: " << n << " t: " << total_time << std::endl;

    return total_time;

}
```

Problem 2:

- n = {10, 100, 1000, 100000, 500000, 1000000}
- Upper Bound = $O(n^2)$
  - The upper bound can be seen to be $O(n^2)$ when we increase *n* by a factor of 10, *t* will increase by a factor of 10^2
- Estimated time for n = 1 Trillion
  - Because the upper bound is quadratic, increasing the value of n=100 by a factor of 10,000,000,000 will get us n = 1 Trillion, so we increase *t* by a factor of 10 billion ^ 2 and we get the estimate of 2.437e24 nanoseconds

Array Sorted Order	
n	t (nanoseconds)
10	1981
100	24370
1000	1001237
100000	6244087060
500000	1.4414E+11
1000000	3.2128E+11



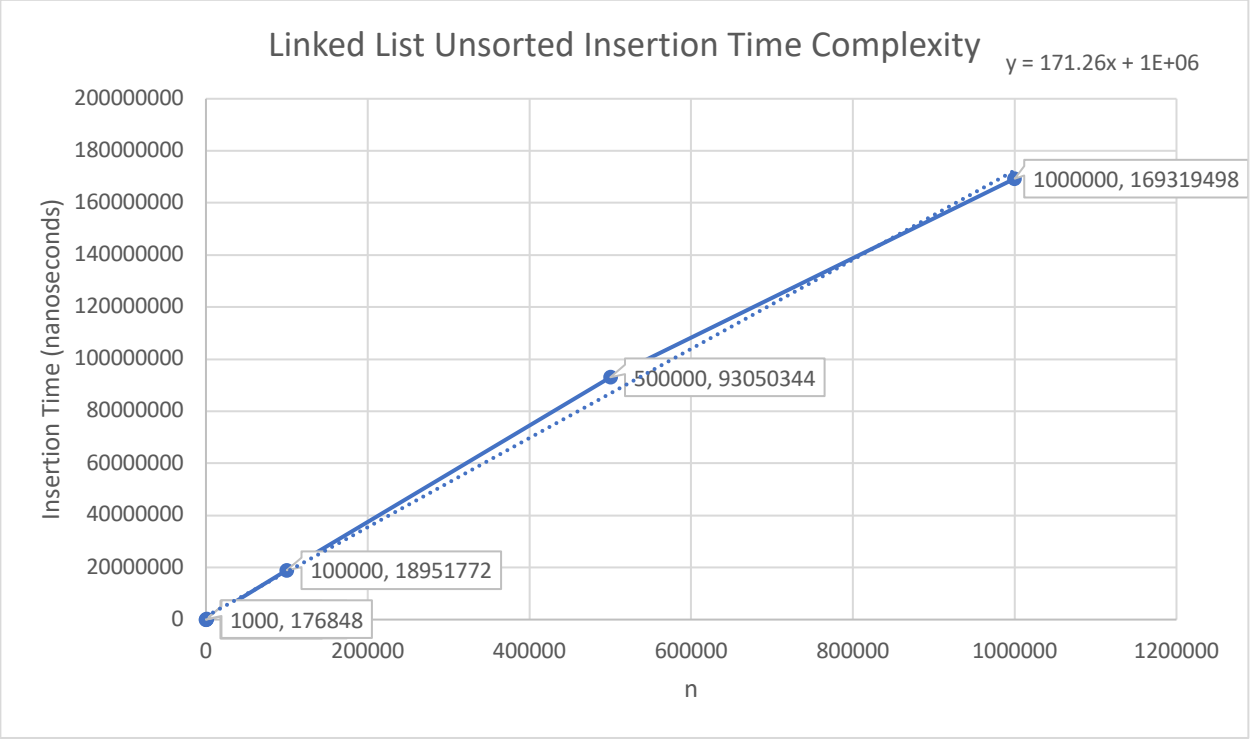
Code:

```
long long problem2(unsigned long int n, std::default_random_engine generator){

    using namespace std::chrono;

    //initialize random number generator

    std::uniform_int_distribution<unsigned long int> dist(1,n);

    //int array for storage
    unsigned long int arr[n];

    //take the current time
    high_resolution_clock::time_point start = high_resolution_clock::now();

    int counter = 0; //used to track number of elements in arr
    for(unsigned long int i = 0; i < n; i++){
        int val = dist(generator);

        if(val > arr[counter - 1] || counter == 0){
            arr[counter] = val;
            counter++;
        }
        else{
            /* Adapted from https://www.geeksforgeeks.org/search-insert-and-delete-
in-a-sorted-array/ */
            int i;
            for (i = counter - 1; (i >= 0 && arr[i] > val); i--)
                arr[i + 1] = arr[i];

            arr[i + 1] = val;
            counter++;
        }

    }

    high_resolution_clock::time_point end = high_resolution_clock::now();
    auto total_time = duration_cast<nanoseconds>(end - start).count();

    std::cout << "n: " << n << " t: " << total_time << std::endl;

    return total_time;

}
```

Problem 3:

- n = {10, 100, 1000, 100000, 500000, 1000000}
- Upper Bound = $O(n)$
  - We can see in the table that the upper bound is $O(n)$ when we increase n by a factor of 10, t increases by a factor of 10 as well.
- Estimated time for n = 1 Trillion
  - Since the upper bound is $O(n)$, we can expect the value at n=100 to increase by a factor of 10 billion. Using this we can estimate the value of t for n = 1 Trillion to be n(100) * 10 billion = 1.8735e14 nanoseconds.

Linked List Random Order	
n	t (nanoseconds)
10	2755
100	18735
1000	176848
100000	18951772
500000	93050344
1000000	169319498



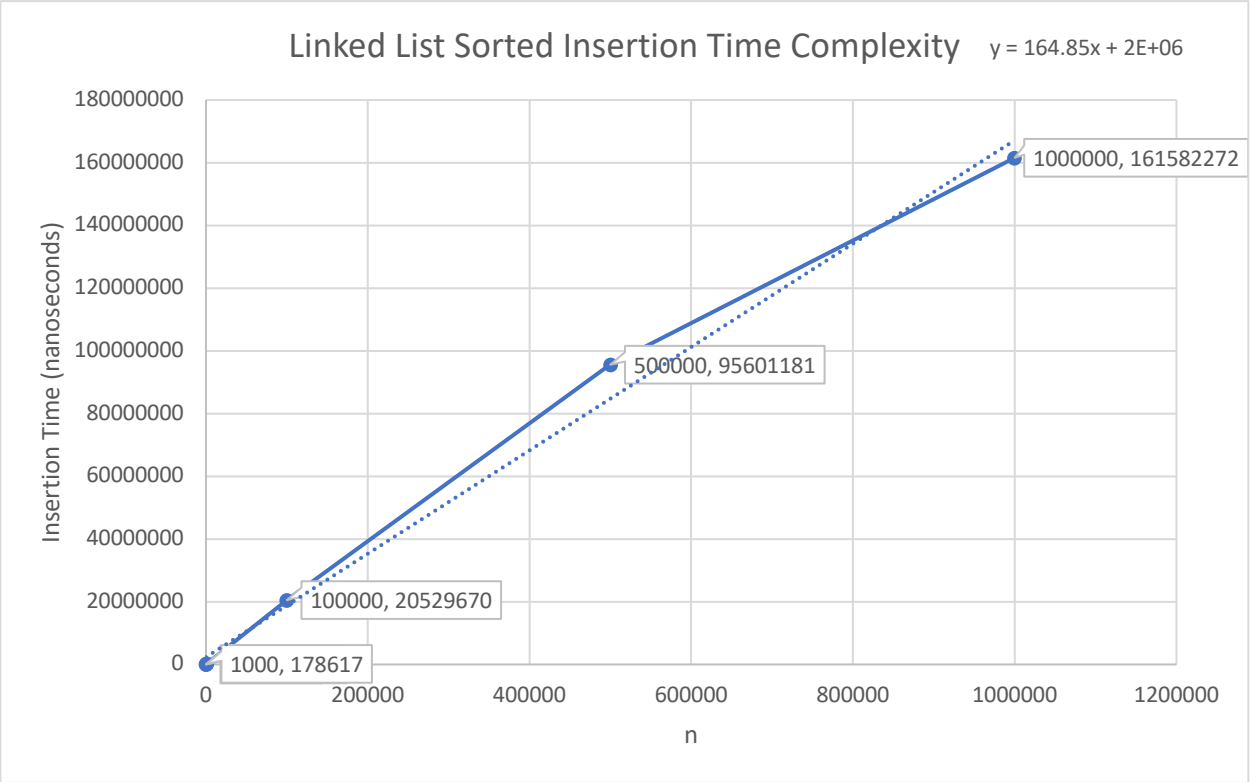
Code:

```
long long problem3(unsigned long int n, std::default_random_engine generator){
    using namespace std::chrono;

    //initialize random number generator
    std::uniform_int_distribution<unsigned long int> dist(1,n);

    //int array for storage
    LinkedList* list = new LinkedList();

    //take the current time
    high_resolution_clock::time_point start = high_resolution_clock::now();

    for(unsigned long int i = 0; i < n; i++){
        list->add(dist(generator));
    }

    high_resolution_clock::time_point end = high_resolution_clock::now();

    auto total_time = duration_cast<nanoseconds>(end - start).count();

    std::cout << "n: " << n << " t: " << total_time << std::endl;


    return total_time;
}
```

Problem 4:

- n = {10, 100, 1000, 100000, 500000, 1000000}
- Upper Bound = $O(n)$
  - From the table we see that the upper bound is $O(n)$ when we increase n by a factor of 10, the value of t increases by a factor of 10 as well.
- Estimated time for n = 1 Trillion
  - Since the upper bound is $O(n)$, we can expect the value at n=100 to increase by a factor of 10 billion. Using this we can estimate the value of t for n = 1 Trillion to be n(100) * 10 billion = 1.9349e14 nanoseconds.

Linked List Sorted	
n	t (nanoseconds)
10	4817
100	19349
1000	178617
100000	20529670
500000	95601181
1000000	161582272



Code:

```
long long problem4(unsigned long int n, std::default_random_engine generator){
    using namespace std::chrono;

    //initialize random number generator

    std::uniform_int_distribution<unsigned long int> dist(1,n);

    //int array for storage
    LinkedList* list = new LinkedList();

    //take the current time
    high_resolution_clock::time_point start = high_resolution_clock::now();

    int counter = 0; //used to track number of elements in arr
    for(unsigned long int i = 0; i < n; i++){

        //if(i % 1000 == 0) std::cout << i <<std::endl;
        int val = dist(generator);


        if(list->isEmpty() || val > list->tail->getData()){
            list->add(val);
            counter++;
        }
        else{ /* Adapted from https://www.geeksforgeeks.org/given-a-linked-list-
which-is-sorted-how-will-you-insert-in-sorted-way/ */
            Node* curr = list->head;
            while(curr->next != nullptr && curr->next->getData() < val)
                curr = curr->next;
            Node* valNode = new Node(val);
            valNode->next = curr->next;
            curr->next = valNode->next;
        }
    }

    high_resolution_clock::time_point end = high_resolution_clock::now();
    auto total_time = duration_cast<nanoseconds>(end - start).count();
    std::cout << "n: " << n << " t: " << total_time << std::endl;

    return total_time;
}
```

Full Code:

Main.cpp

```cpp
int main() {

    int n = 6;
    unsigned long long int n_list[] = {10, 100, 1000, 100000, 500000, 1000000};
    long long times_1[6];
    long long times_2[6];
    long long times_3[6];
    long long times_4[6];

    std::random_device rd;
    std::default_random_engine generator(rd());

    //run experiments 1-4
    for(int i = 0; i < n; i++) {
        times_1[i] = problem1(n_list[i], generator);
        times_2[i] = problem2(n_list[i], generator);
        times_3[i] = problem3(n_list[i], generator);
        times_4[i] = problem4(n_list[i], generator);
    }

    std::ofstream fout("problem1_output.csv");
    std::ofstream fout2("problem2_output.csv");
    std::ofstream fout3("problem3_output.csv");
    std::ofstream fout4("problem4_output.csv");


    fout << "n,t\n";
    fout2 << "n,t\n";
    fout3 << "n,t\n";
    fout4 << "n,t\n";
    for(int i = 0; i < n; i++){
        fout << n_list[i] << "," << times_1[i] << "\n";
        fout2 << n_list[i] << "," << times_2[i] << "\n";
        fout3 << n_list[i] << "," << times_3[i] << "\n";
        fout4 << n_list[i] << "," << times_4[i] << "\n";
    }

    fout.close();
    fout2.close();
    fout3.close();
    fout4.close();
    return 0;
}
```

Linked List (.h)

```h
#include <cstdlib>
#include <iostream>

#ifndef HOMEWORK_1_LINKEDLIST_H
#define HOMEWORK_1_LINKEDLIST_H


class Node
{
public:
    Node* next;
    Node(){next = nullptr;}
    Node(unsigned long int n){data = n;}
    unsigned long int data;
    friend class LinkedList;

    unsigned long int getData(){
        return data;
    }

    void setData(unsigned long int n){ data = n;}
};

using namespace std;

class LinkedList
{
public:
    unsigned long int length;
    Node* head;
    Node* tail;

    LinkedList();
    ~LinkedList();
    void add(unsigned long int data);
    void print();
    Node* operator[](unsigned long int);
    bool isEmpty();
};


#endif //HOMEWORK_1_LINKEDLIST_H
```

Linked List (.cpp)

```cpp
#include "LinkedList.h"

LinkedList::LinkedList(){
    this->length = 0;
    this->head = nullptr;
    this->tail = nullptr;
}

LinkedList::~LinkedList(){
    if(head == nullptr)
        return;
}

void LinkedList::add(unsigned long int data){
    Node* node = new Node();
    node->data = data;
    if(head == nullptr){
        this->head = node;
        this->tail = node;
        node->next = nullptr;
        this->length++;
    }
    else{
        this->tail->next = node;
        node->next = nullptr;
        this->tail = node;
        this->length++;
    }
}

void LinkedList::print(){
    Node* head = this->head;
    while(head){
        std::cout << head->data << std::endl;
        head = head->next;
    }
}

Node* LinkedList::operator[](unsigned long int i) {
    if(head == nullptr)
        return nullptr;
    Node* temp = head;
    int j = 0;
    while(temp){
        if(j == i)
            return temp;
        temp = temp->next;
        j++;
    }
    return temp;
}

bool LinkedList::isEmpty() {
    if(head == nullptr)
        return true;
    else
        return false;
}
```