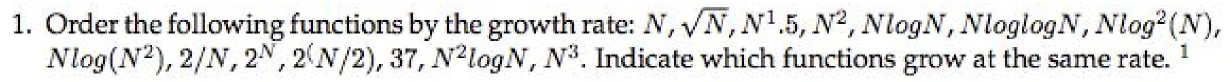
Robert Max Williams

Assignment 2, September 26, 2017

[A] Monogram on White Background

[B] Monogram on Red Background
[C] Monogram on Black Background



1. Order the following functions by the growth rate: N, $\sqrt{N}$, $N^1.5$, N^2, $NlogN$, $NloglogN$, $Nlog^2(N)$, $Nlog(N^2)$, $2/N$, 2^N, $2^(N/2)$, 37, N^2logN, N^3. Indicate which functions grow at the same rate. [1]

Order 1/N

$2/N$

Order C

$.5$

37

Order N^1/2

$\sqrt{N}$

Order N

N

N^1

$2(N/2)$

Order

$Nlog(log(N))$

Order N logN

$Nlog(N)$

$Nlog(N^2)$

Order

$Nlog(N)^2$

Order N^2

N^2

$N^2log(N)$

Order N^3

N^3

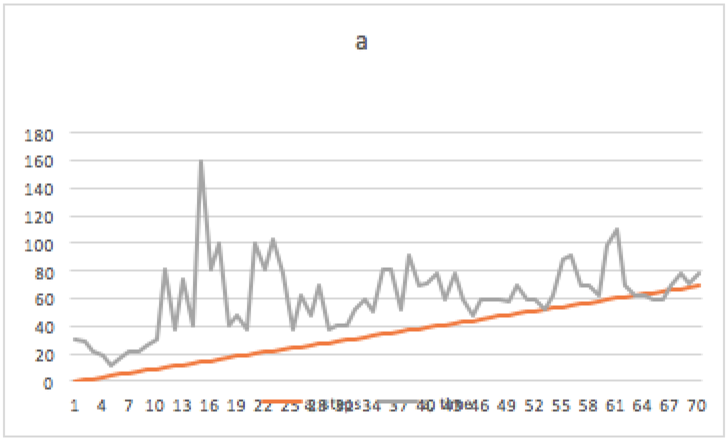
Order 2^N

2^N

Problem 2. Running time, code for each, and test actual code

(a)

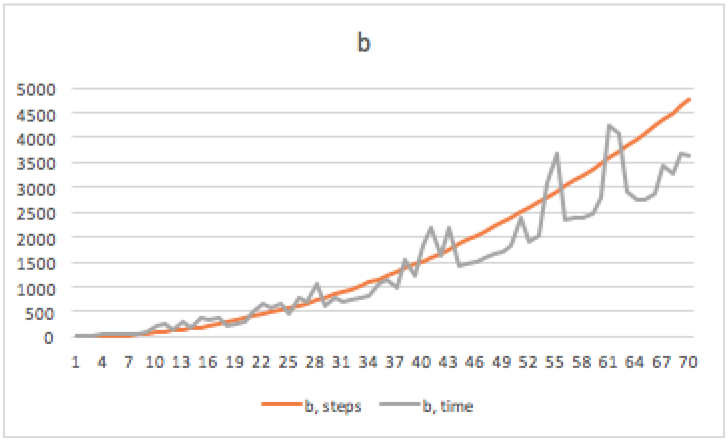
```
sum = 0;
for( i = 0; i < n; ++i )
    ++sum;
```



Running time n

(b)

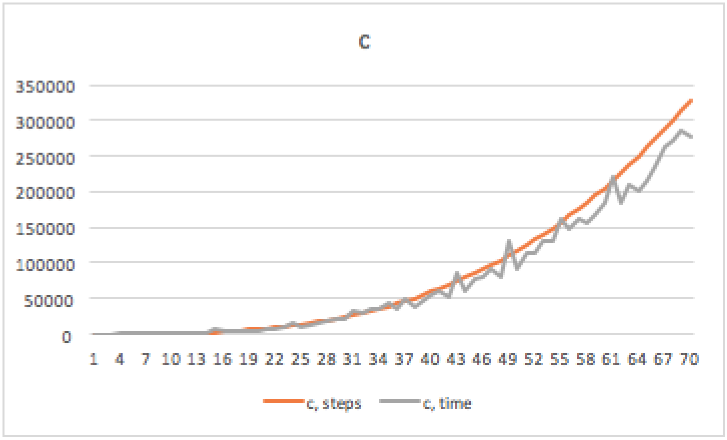
```
sum = 0;
for( i = 0; i < n; ++i )
    for( j = 0; j < n; ++j )
    ++sum;
```



Running time n^2

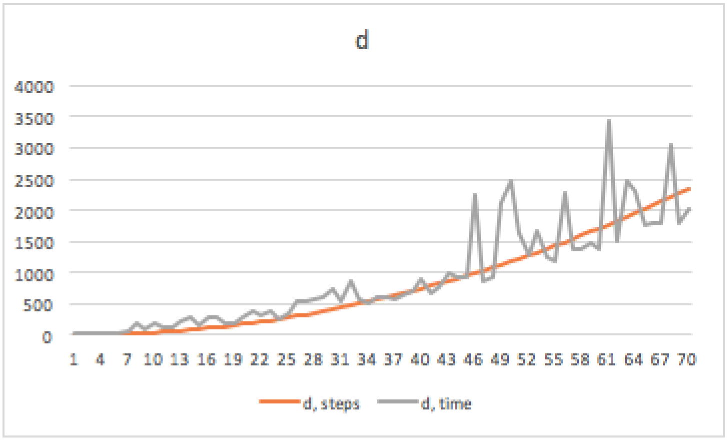
```
        ++sum;
```

(c)
```
sum = 0;
for( i = 0; i < n; ++i )
        for( j = 0; j < n * n; ++j )
            ++sum;
```



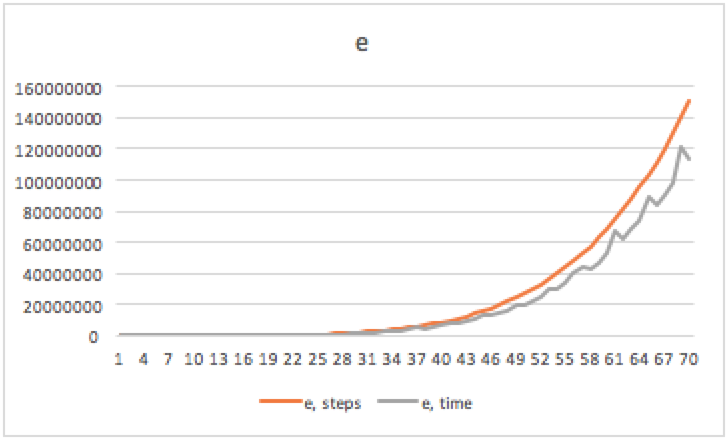
Running time n^3

(d)
```
sum = 0;
for( i = 0; i < n; ++i )
    for( j = 0; j < i; ++j )
        ++sum;
```



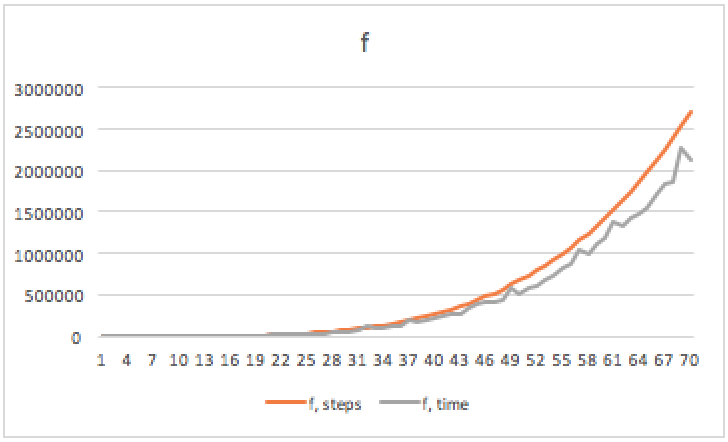
Running time (n^2)/2

(e)
```
sum = 0;
for( i = 0; i < n; ++i )
  for( j = 0; j < i * i; ++j )
      for( k = 0; k < j; ++k )
          ++sum;
```



Running time (n^5)/10

(f)
```
sum = 0;
for( i = 1; i < n; ++i )
 for( j = 1; j < i * i; ++j )
       if( j % i == 0 )
         for( k = 0; k < j; ++k )
            ++sum;
```



I found an equivialent summation and worked it down to a closed form of order **n^4**

$$\sum_{i=1}^{n}\sum_{j=1}^{i-1} i*j = \frac{n^2(n+1)^2}{8} - \frac{n(n+1)(2n+1)}{12}$$

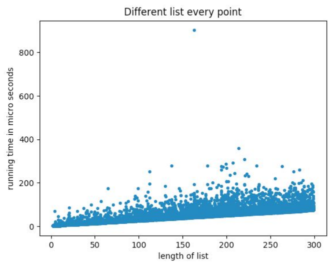
Code for generating this data, included as problem2.py:

```python
import csv
import time

def func_a(n):
    summ = 0
    for i in range(n):
        summ += 1
    return summ

def func_b(n):
    summ = 0
    for i in range(n):
        for j in range(n):
            summ += 1

    return summ

def func_c(n):
    summ = 0
    for i in range(n):
        for j in range(n**2):
            summ += 1

    return summ

def func_d(n):
    summ = 0
    for i in range(n):
        for j in range(i):
            summ += 1

    return summ

def func_e(n):
    summ = 0
    for i in range(n):
        for j in range(i**2):
            for k in range(j):
                summ += 1

    return summ

def func_f(n):
    summ = 0
    for i in range(1, n):
        for j in range(1, i**2):
            if j % i == 0:
                for k in range(j):
                    summ += 1

    return summ
```

```python
with open('function-times.csv', 'w') as csvfile:
    writer = csv.writer(csvfile)

    fieldnames = ['n', 'a, steps', 'a, time', 'b, steps', 'b, time', 'c, steps', 'c, time',
                  'd, steps', 'd, time', 'e, steps', 'e, time', 'f, steps', 'f, time']
    writer.writerow(fieldnames)
    for n in range(70):
        row = [n]
        for func in [func_a, func_b, func_c, func_d, func_e, func_f]:
            start = time.time()
            count = func(n)
            duration = (time.time() - start)*10**7
            print(n, count, duration)
            row += [count, duration]
        writer.writerow(row)
```

Problem 3: find efficient algorithms

(a) Find the minimum subsequence sum.



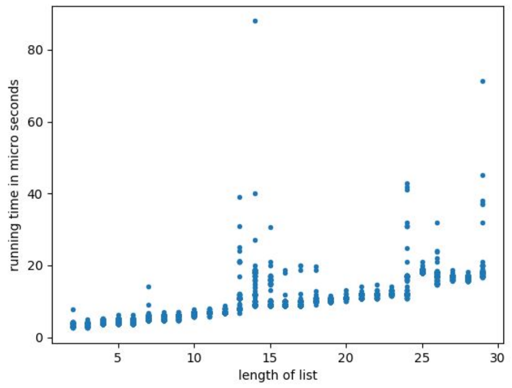
Code:

```python
def max_subsequence(seq):
    max_so_far = 0
    max_ending_here = 0

    for i, x in enumerate(seq):
        max_ending_here += x
        if max_ending_here < 0:
            max_ending_here = 0
        if max_so_far < max_ending_here:
            max_so_far = max_ending_here
    return max_so_far

def min_subsequence(seq):
    negative_seq = (-x for x in seq)
    return -max_subsequence(negative_seq)
```

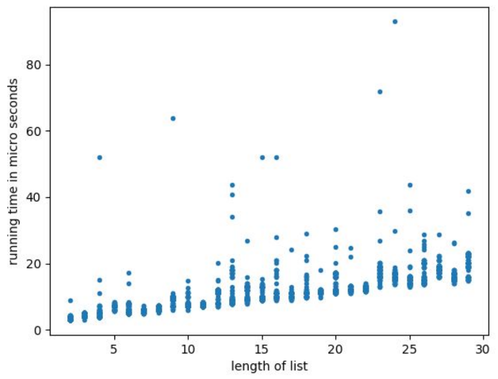
This could have been shorter, but I couldn’t pass up the opportunity to manumipulate inputs to use one function for a different purpose.

(b) Find the maximum subsequence product.

In this figure, the same list is worked on in each column (same length), to show that computation time can vary greatly by the random conditions that the code is running in, with the OS multitasking heavily and python running garbage collector and other non-deterministic time operations.



The next figure shows the same code, but never reusing the same list.



It seem that it runs in linear time, with noise cause primarily by the operating system. There is nothing in the code to suggest anything but linear running time, 2n to be more specific.

Code:

```python
"""
Minimum sum and max product subsequence
"""
from random import randint as dice

def gen_seq(length):
    seq = list()
    for _ in range(length):
        seq.append(dice(-100, 100))
    return tuple(seq)

def max_product_forward(seq):
    """
    tried to find max product by multing everything since last zero
    fails with odd number of negatives tail heavy, so needs to be ran twice,
    once forward and once negative, to ensure those solutions aren't hiding from us
    Only works for whole number lists
    """
    this_start = 0
    this_product = 1
    max_product = 0
    max_start, max_end = (0, 0)
    for i, x in enumerate(seq):
        this_product *= x

        if x == 0:
            this_start = i+1
            this_product = 1
            continue

        if this_product > max_product:
            max_product = this_product
            max_start, max_end = this_start, i+1

        ##print(seq[max_start:max_end])

    return (max_product, max_start, max_end)

def max_product(seq):
    """
    calls max_product_forward with seq and seq reversed, and returns whichever one
    makes the bigger number
    """
    forward_sum, forward_start, forward_end = max_product_forward(seq)
    backward_sum, backward_start, backward_end = max_product_forward(seq[::-1])

    if forward_sum > backward_sum:
        return seq[forward_start:forward_end]
    else:
        return (seq[::-1][backward_start:backward_end])[::-1]
```

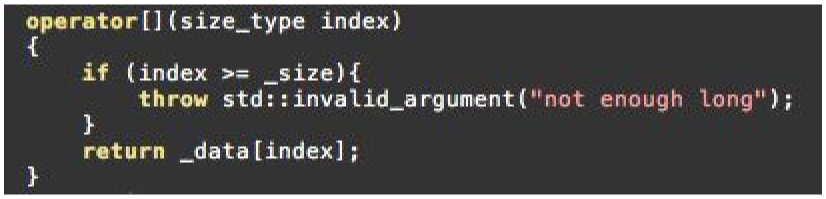
Problem 4:

Finder intersection of two std::list

```
//Given two sorted lists, L1 and L2, return their intersection i.e. elements
in L2 not found in L1
#include <iostream>
#include <list>

using namespace std;

int main()
{
    list<int> L1 = {2, 4, 6, 8, 10};
    list<int> L2 = {1, 2, 3, 4, 5};

    auto iter1 = L1.begin(), end1 = L1.end();
    auto iter2 = L2.begin(), end2 = L2.end();

    while(iter2 != end2 && iter1 != end1){
        if (*iter1 == *iter2){
            iter2++;
            iter1++;
        }
        else{
            cout << *iter2 << endl;
            iter2++;
        }
    }
}
```

This code is a great improvement on my first attempt using nested *for loops*. The logic is greatly simplified by the fact the lists are sorted. It performs a number of operations equal to L1+L2, or N+M.

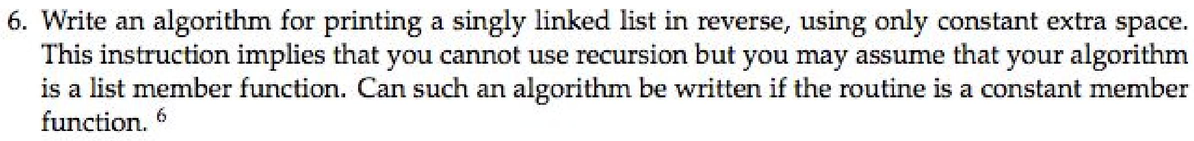
Problem 5:



```
operator[](size_type index)
{
    if (index >= _size){
        throw std::invalid_argument("not enough long");
    }
    return _data[index];
}
```

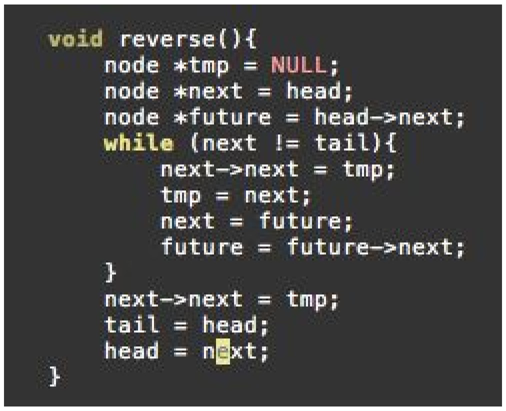
If the index exceeds the size, an err is thrown.

Problem 6:



6. Write an algorithm for printing a singly linked list in reverse, using only constant extra space. This instruction implies that you cannot use recursion but you may assume that your algorithm is a list member function. Can such an algorithm be written if the routine is a constant member function. [6]

The crux move is to realize you can reverse it and print it with small constant storage.



```
void reverse(){
    node *tmp = NULL;
    node *next = head;
    node *future = head->next;
    while (next != tail){
        next->next = tmp;
        tmp = next;
        next = future;
        future = future->next;
    }
    next->next = tmp;
    tail = head;
    head = next;
}
```

My implementation is not very elegant but it gets the job done.