Matt Hyatt
COMP 272

time complexities of algorithms

```
alg 1
    O(n^2)
alg2
    O(n)
alg3
    O(n)
alg4
    O(n)
```

manual explanation of maximum subsequence algorithms

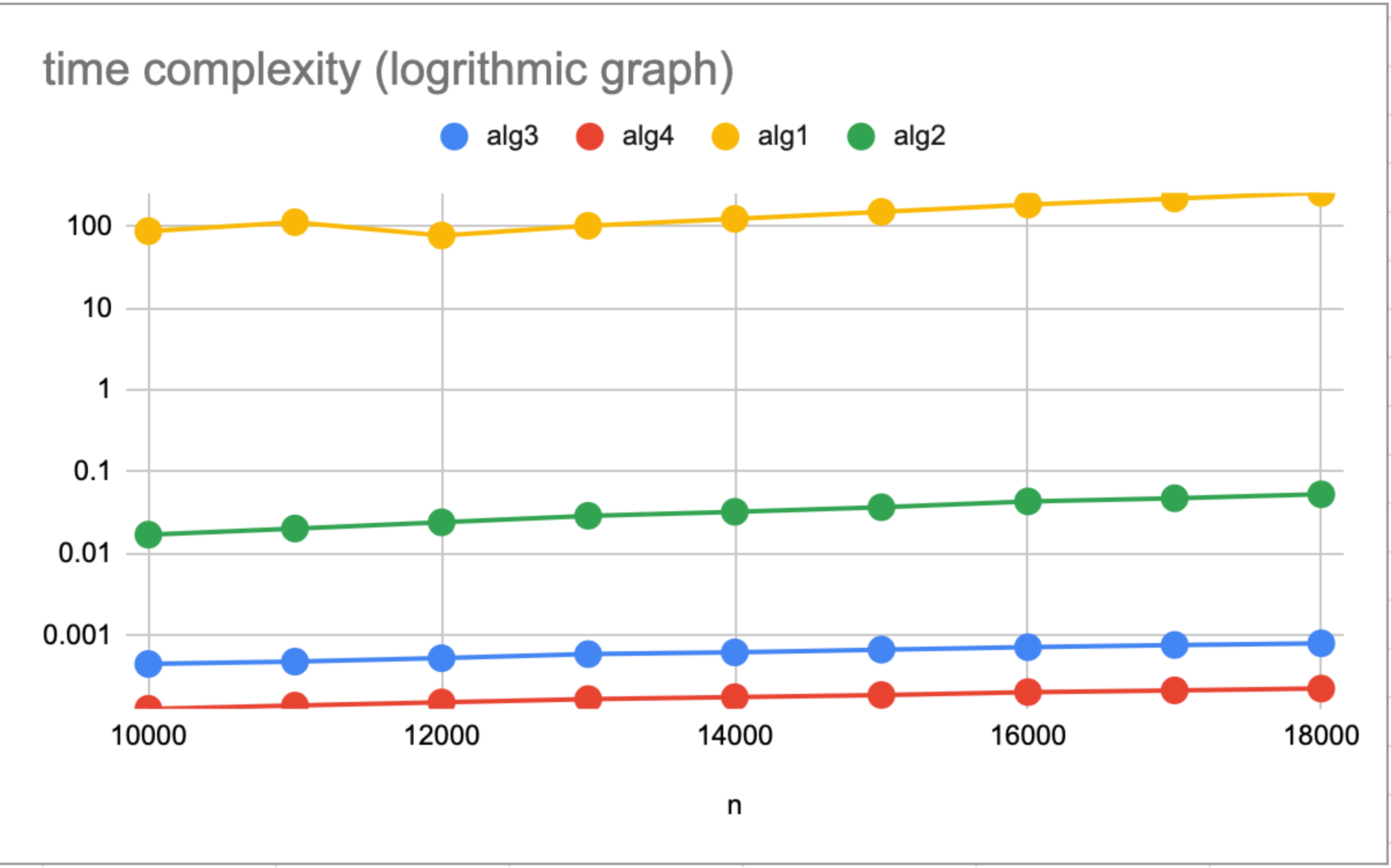
```
maxSubSum1
    for every potential start value in the sequence
        for every potential end value in the sequence
            compute the sum between the given start and end
            check if this is the greatest recorded sum so far

maxSubSum2
    for every potential start value in the sequence
        check intermediately if this value is the greatest recorded sum
        add the next value in the sequence and recheck

maxSubSum3
    **for this algorithm I did not completely understand the way it worked**
    partitions the array into pieces in order to find the maximum sum
    uses comparissons to efficiently eliminate pieces of the sequence
        which would not benefit the maxSum
    then it combines the pieces back together to compute the maxSum

maxSubSum4
    running through the sequence only once...
        add the next value in the sequence to the intermediate sum
        check intermediately if this value is the greatest recorded sum
        check intermediately if this value is negative
            if it is then restart this pattern with the
            next value in the sequence being the new start of the sequence

            because the only time you would need to check different
            start values is when the sum becomes negative
```


time complexity (logrithmic graph)
alg3
alg4
alg1
alg2
100
10
1
0.1
0.01
0.001
10000
12000
14000
16000
18000
n

import java.util.Random;

// This includes additional code not in the text that keeps // track of the starting and ending points of best sequence

public final class MaxSumTest { static private int seqStart = 0; static private int seqEnd = -1;

```
/**
 * Cubic maximum contiguous subsequence sum algorithm.
 * seqStart and seqEnd represent the actual best sequence.
 */
public static int maxSubSum1( int [ ] a ){
    int maxSum = 0;

    for( int i = 0; i < a.length; i++ )
        for( int j = i; j < a.length; j++ )
        {
            int thisSum = 0;

            for( int k = i; k <= j; k++ )
                thisSum += a[ k ];

            if( thisSum > maxSum )
            {
                maxSum   = thisSum;
                seqStart = i;
                seqEnd   = j;
            }
        }

    return maxSum;
}

/**
 * Quadratic maximum contiguous subsequence sum algorithm.
 * seqStart and seqEnd represent the actual best sequence.
 */
public static int maxSubSum2( int [ ] a ){
    int maxSum = 0;

    for( int i = 0; i < a.length; i++ )
    {
        int thisSum = 0;
        for( int j = i; j < a.length; j++ )
        {
            thisSum += a[ j ];
            if( thisSum > maxSum )
            {
                maxSum = thisSum;
                seqStart = i;
                seqEnd   = j;
            }
        }
    }

    return maxSum;
}

/**
 * Linear-time maximum contiguous subsequence sum algorithm.
 * seqStart and seqEnd represent the actual best sequence.
 */
public static int maxSubSum4( int [ ] a ){
    int maxSum = 0;
    int thisSum = 0;

    for( int i = 0, j = 0; j < a.length; j++ )
    {
        thisSum += a[ j ];

        if( thisSum > maxSum )
        {
            maxSum = thisSum;
            seqStart = i;
            seqEnd   = j;
        }
        else if( thisSum < 0 )
        {
            i = j + 1;
            thisSum = 0;
        }
    }

    return maxSum;
}


/**
 * Recursive maximum contiguous subsequence sum algorithm.
 * Finds maximum sum in subarray spanning a[left..right].
 * Does not attempt to maintain actual best sequence.
 */
private static int maxSumRec( int [ ] a, int left, int right ){
    int maxLeftBorderSum = 0, maxRightBorderSum = 0;
    int leftBorderSum = 0, rightBorderSum = 0;
    int center = ( left + right ) / 2;

    if( left == right )  // Base case
        return a[ left ] > 0 ? a[ left ] : 0;

    int maxLeftSum  = maxSumRec( a, left, center );
    int maxRightSum = maxSumRec( a, center + 1, right );

    for( int i = center; i >= left; i-- )
    {
        leftBorderSum += a[ i ];
        if( leftBorderSum > maxLeftBorderSum )
            maxLeftBorderSum = leftBorderSum;
    }

    for( int i = center + 1; i <= right; i++ )
    {
        rightBorderSum += a[ i ];
        if( rightBorderSum > maxRightBorderSum )
            maxRightBorderSum = rightBorderSum;
    }

    return max3( maxLeftSum, maxRightSum,
                 maxLeftBorderSum + maxRightBorderSum );
}

/**
 * Return maximum of three integers.
 */
private static int max3( int a, int b, int c )
{
    return a > b ? a > c ? a : c : b > c ? b : c;
}

/**
 * Driver for divide-and-conquer maximum contiguous
 * subsequence sum algorithm.
 */
public static int maxSubSum3( int [ ] a ){
    return a.length > 0 ? maxSumRec( a, 0, a.length - 1 ) : 0;
}
```

```
public static void getTimingInfo( int n, int alg )
{
    int [] test = new int[ n ];

    long startTime = System.currentTimeMillis( );;
    long totalTime = 0;

    int i;
    for( i = 0; totalTime < 4000; i++ )
    {
        for( int j = 0; j < test.length; j++ )
            test[ j ] = rand.nextInt( 100 ) - 50;

        switch( alg )
        {
          case 1:
            maxSubSum1( test );
            break;
          case 2:
            maxSubSum2( test );
            break;
          case 3:
            maxSubSum3( test );
            break;
          case 4:
            maxSubSum4( test );
            break;
        }

        totalTime = System.currentTimeMillis( ) - startTime;
    }

    System.out.print( String.format( "\t%12.6f", ( totalTime * 1000 / i ) / (double) 100000
}

private static Random rand = new Random( );

/**
 * Simple test program.
 */
public static void main( String [ ] args )
{
    int a[ ] = { 4, -3, 5, -2, -1, 2, 6, -2 };
    int maxSum;
    maxSum = maxSubSum1( a );
    System.out.println( "Max sum is " + maxSum + "; it goes"
                   + " from " + seqStart + " to " + seqEnd );
    maxSum = maxSubSum2( a );
    System.out.println( "Max sum is " + maxSum + "; it goes"
                   + " from " + seqStart + " to " + seqEnd );
    maxSum = maxSubSum3( a );
    System.out.println( "Max sum is " + maxSum );
    maxSum = maxSubSum4( a );
    System.out.println( "Max sum is " + maxSum + "; it goes"
                   + " from " + seqStart + " to " + seqEnd );

      // Get some timing info
    for( int n = 10000; n <= 18000; n += 1000 ) //int n = 100; ... ;n*=10
    {
        System.out.print( String.format( "N = %7d" , n ) );

        for( int alg = 1; alg <= 4; alg++ )
        {
            if( alg == 1 && n > 50000 )
            {
                System.out.print( "\t      NA    " );
                continue;
            }
            getTimingInfo( n, alg );
        }

        System.out.println( );
    }
}

}
```