```
/**
 * Recursive maximum contiguous subsequence sum algorithm.
 * Finds maximum sum in subarray spanning a[left..right].
 * Does not attempt to maintain actual best sequence.
 * Divide-and conquer algorithm: the input is divided into
 * two halves.
 * The maximum contiguous subsequence sum can occur in one
 * of three ways
 * 1. It resides entirely in the first half
 * 2. It resides entirely in the second half.
 * 3. It begins in the first hald but ends in the second half.
 */
private static int maxSumRec( int [ ] a, int left, int right ) {
  if( left == right ) // Base case
    if( a[ left ] > 0 )
      return a[ left ];
    else
      return 0;
  int center = ( left + right ) / 2;
  int maxLeftSum = maxSumRec( a, left, center );
  int maxRightSum = maxSumRec( a, center + 1, right );

  int maxLeftBorderSum = 0, leftBorderSum = 0;
  for( int i = center; i >= left; i-- ) {
    leftBorderSum += a[ i ];
    if( leftBorderSum > maxLeftBorderSum )
      maxLeftBorderSum = leftBorderSum;
  }

  int maxRightBorderSum = 0, rightBorderSum = 0;
  for( int i = center + 1; i <= right; i++ ) {
    rightBorderSum += a[ i ];
    if( rightBorderSum > maxRightBorderSum )
      maxRightBorderSum = rightBorderSum;
  }
  return max3( maxLeftSum, maxRightSum,
      maxLeftBorderSum + maxRightBorderSum );
}

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


/**
 * Linear-time maximum contiguous subsequence sum algorithm.
 * At any point in time, the algorithm can correctly give an
 * answer to the subsequence problem for the data it has
 * already read: it is an online algorithm.
 * An online algorithm that requires only constant space and
 * runs in linear time is just about as good as possible.
 * The correctness of this algoritm is not obvious and hard to
 * prove
 */
public static int maxSubSum4( int [ ] a ) {
  int maxSum = 0, thisSum = 0;

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

    if( thisSum > maxSum )
      maxSum = thisSum;
    else if( thisSum < 0 )
      thisSum = 0;
  }
  return maxSum;
}
```

~~add~~ all positions

if the current sum is −, That means it's better to start new, record it largest

```
 /**
 * Cubic maximum contiguous
 * subsequence sum algorithm.
 */
 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;
        }

    return maxSum;
}


/**
 * Quadratic maximum contiguous subsequence sum algorithm.
 */
 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;
      }
   }

   return maxSum;
 }
```

for every start pt
keep adding ~~[illegible]~~ points on
and record as max,
if really big