Pseudocode

enum dataState:

presorted,

reversesorted,

random

// Precondition: *A* is a reference to an array with the data,

// *start* is the index of the first element in the left division,

// *m* is the index of the first element in the right division,

// and *end* is the index of the last element in the right division.

// The left and right divisions must each be sorted.

// Postcondition: The elements from *start* to *end*, inclusive, are in

// increasing sorted order.

merge( A, start, m, end ):

size = end – start + 1

for( k = start; k < m; k++ )

L[k - start] = A[i]

L[m – start] = ∞

i = j = 0 // we are indexing from zero

for( k = 0; k < size; k++ ):

if( L[i] < A[j + m] || j > end – m ):

A[k + start] = L[i]

i++

else:

A[k + start] = A[j + m]

j++

// Precondition: *A* is a reference to an array with the data that is to be

// sorted from *min* to *max* inclusive.

// Postcondition: The data in *A* from *min* to *max* are in increasing sorted

// order.

mergesort( A, min, max ):

if( max != min ):

center = floor( (max + min + 1) / 2 )

mergesort( A, min, center – 1 )

mergesort( A, center, max )

merge( A, min, center, max )

main():

foreach ds in dataState:

// Let’s test powers of sqrt(2),

// by doing so we will see how powers of 2 affect merge sort.

// Loop invariant: at each comparison of the guard,

// times contains data from k-1 tests of data sizes of

// the nearest whole numbers to the first k-1 powers of sqrt(2).

for( exponent = 1; exponent < MAX\_EXPONENT; exponent++ ):

n = floor( (sqrt(2) ^ exponent) + 0.5 )

GenerateData( original\_data, n, ds )

if( !isSorted( data, n, ds ) ):

Error()

data = original\_data

t = currentTime()

mergesort( data, 0, n-1 )

duration = currentTime() – t

if( !isSorted( data, n, presorted ) ):

Error()

output( time )

data = original\_data

t = currentTime()

insertionsort( data, 0, n-1 )

duration = currentTime() – t

if( !isSorted( data, n, presorted ) ):

Error()

output( time )

data = original\_data

t = currentTime()

bubblesort( data, 0, n-1 )

duration = currentTime() – t

if( !isSorted( data, n, presorted ) ):

Error()

output( time )

output( exponent )

output( n )