Pseudocode

enum dataState:

presorted,

reversesorted,

random

struct record:

size,

state,

mergeTime, bubbleTime, insertionTime

RunAlgs( size, dataState ):

GenerateData(B, size, dataState)

A=B

startTime = currentTime()

mergesort(A, 0, size-1) // we index from 0

mergesortTime = currentTime() – startTime

A=B

startTime = currentTime()

bubblesort(A, 0, size-1) // we index from 0

bubblesortTime = currentTime() – startTime

A=B

startTime = currentTime()

insertionsort(A, 0, size-1) // we index from 0

insertionsortTime = currentTime() – startTime

return record( size, dataState, mergesortTime, bubblesortTime, insertionsortTime )

// 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():

times = []

// Let’s test powers of sqrt(2),

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

// 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( long k = 1; k < MAX\_EXPONENT; k++ ):

times.append( RunAlgs( floor( (sqrt(2) ^ k) + 0.5 ), random ) )

times.append( RunAlgs( floor( (sqrt(2) ^ k) + 0.5 ), presorted ) )

times.append( RunAlgs( floor( (sqrt(2) ^ k) + 0.5 ), reversesorted ) )

output\_to\_file( times )