**//steven guo**

**//merge sorting program**

**//10/20/19**

**import** java.util.Random;

**public** **class** mergeSort {

**static** **void** merge(**int** arr[], **int** l, **int** m, **int** r)

{

// Find sizes of two subarrays to be merged

**int** n1 = m - l + 1;

**int** n2 = r - m;

/\* Create temp arrays \*/

**int** L[] = **new** **int** [n1];

**int** R[] = **new** **int** [n2];

/\*Copy data to temp arrays\*/

**for** (**int** i=0; i<n1; ++i)

L[i] = arr[l + i];

**for** (**int** j=0; j<n2; ++j)

R[j] = arr[m + 1+ j];

/\* Merge the temp arrays \*/

// Initial indexes of first and second subarrays

**int** i = 0, j = 0;

// Initial index of merged subarry array

**int** k = l;

**while** (i < n1 && j < n2)

{

**if** (L[i] <= R[j])

{

arr[k] = L[i];

i++;

}

**else**

{

arr[k] = R[j];

j++;

}

k++;

}

/\* Copy remaining elements of L[] if any \*/

**while** (i < n1)

{

arr[k] = L[i];

i++;

k++;

}

/\* Copy remaining elements of R[] if any \*/

**while** (j < n2)

{

arr[k] = R[j];

j++;

k++;

}

}

**static** // Main function that sorts arr[l..r] using

// merge()

**void** sort(**int** arr[], **int** l, **int** r)

{

**if** (l < r)

{

// Find the middle point

**int** m = (l+r)/2;

// Sort first and second halves

*sort*(arr, l, m);

*sort*(arr , m+1, r);

// Merge the sorted halves

*merge*(arr, l, m, r);

}

}

//function that fills array with random number from 1-10000

**public** **static** **int**[] randArray(**int** size, **int**[] arr)

{

Random rand = **new** Random();

**int** number;

**for**(**int** i = 0; i < size-1; i++)

{

number = rand.nextInt(10000)+1;

arr[i] = number;

}

**return** arr;

}

//function that fills array with sorted numbers

**public** **static** **int**[] sortedArray(**int** size, **int**[] arr)

{

**for**(**int** i = 0; i < size-1; i++)

{

arr[i] = i;

}

**return** arr;

}

//function that fills array with sorted numbers except every 10th element is random

**public** **static** **int**[] almostSortedArray(**int** size, **int**[] arr)

{

Random rand = **new** Random();

**for**(**int** i = 0; i < size-1; i++)

{

**if**(i%10==0)

{

arr[i] = rand.nextInt(10000)+1;

}

**else**

{

arr[i] = i;

}

}

**return** arr;

}

**public** **static** **void** main(String arr[])

{

**final** **int** SIZE = 1000;

**final** **int** SIZE1 = 10000;

**final** **int** SIZE2 = 100000;

**long** time = System.*nanoTime*();

**int**[] arr1 = **new** **int**[SIZE];

**int**[] arr2 = **new** **int**[SIZE1];

**int**[] arr3 = **new** **int**[SIZE2];

*sort*(*randArray*(SIZE, arr1),0,arr1.length-1);

System.***out***.println("It took " + ((System.*nanoTime*() - time)/1e-9) + " seconds for random array with 1000 elements");

time = System.*nanoTime*();

*sort*(*randArray*(SIZE1, arr2),0,arr2.length-1);

System.***out***.println("It took " + ((System.*nanoTime*() - time)/1e-9) + " seconds for random array with 10000 elements");

time = System.*nanoTime*();

*sort*(*randArray*(SIZE2, arr3),0,arr3.length-1);

System.***out***.println("It took " + ((System.*nanoTime*() - time)/1e-9) + " seconds for random array with 100000 elements");

time = System.*nanoTime*();

*sort*(*sortedArray*(SIZE, arr1),0,arr1.length-1);

System.***out***.println("It took " + ((System.*nanoTime*() - time)/1e-9) + " seconds for sorted array with 1000 elements");

time = System.*nanoTime*();

*sort*(*sortedArray*(SIZE1, arr2),0,arr2.length-1);

System.***out***.println("It took " + ((System.*nanoTime*() - time)/1e-9) + " seconds for sorted array with 10000 elements");

time = System.*nanoTime*();

*sort*(*sortedArray*(SIZE2, arr3),0,arr3.length-1);

System.***out***.println("It took " + ((System.*nanoTime*() - time)/1e-9) + " seconds for sorted array with 100000 elements");

time = System.*nanoTime*();

*sort*(*almostSortedArray*(SIZE, arr1),0,arr1.length-1);

System.***out***.println("It took " + ((System.*nanoTime*() - time)/1e-9) + " seconds for almost sorted array with 1000 elements");

time = System.*nanoTime*();

*sort*(*almostSortedArray*(SIZE1, arr2),0,arr2.length-1);

System.***out***.println("It took " + ((System.*nanoTime*() - time)/1e-9) + " seconds for almost sorted array with 10000 elements");

time = System.*nanoTime*();

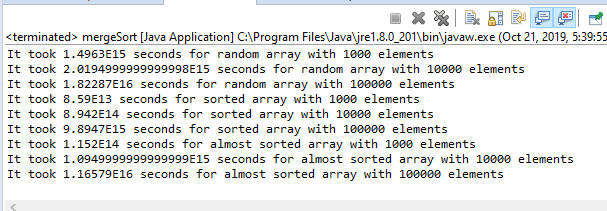
*sort*(*almostSortedArray*(SIZE2, arr3),0,arr3.length-1);

System.***out***.println("It took " + ((System.*nanoTime*() - time)/1e-9) + " seconds for almost sorted array with 100000 elements");

time = System.*nanoTime*();

}

}



The time complexity of merge sort is O(*n* log *n*). A merge sort divides the unsorted list into n sub lists, each containing one element. Then it repeatedly merge sub lists to produce new sorted sub lists until there’s only one sub list remaining which will be the sorted list. The average and best case is O(n) and worst case is O(*n* log *n*). If the running time of merge sort for a list of length n is T(n), then the recurrence T(n) = 2T(n/2) + n follows from the definition of the algorithm (apply the algorithm to two lists of half the size of the original list, and add the n steps taken to merge the resulting two lists). My theory is that all times should be around the same depending on how many elements there are. Based on the results, it seems to be true for everything except the sorted array. Which is my worst case.