Merge sort summary

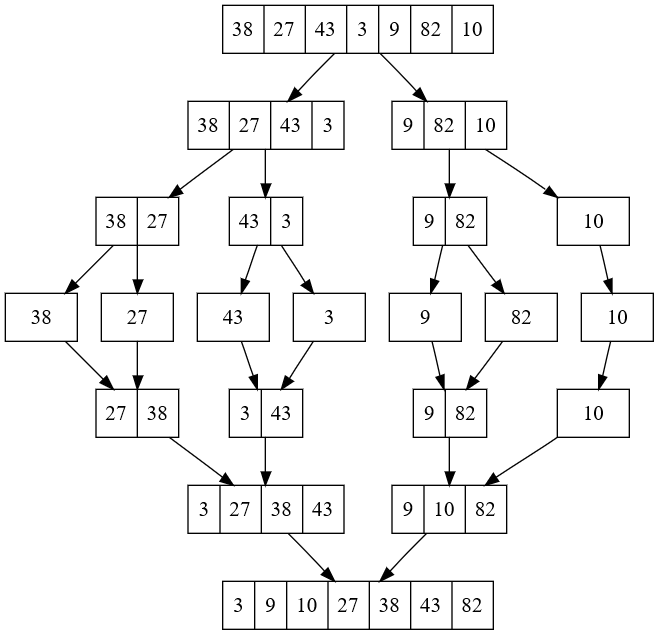
# Overview of Merge sort

The Merge sort algorithm was developed by John von Neumann in 1945. Merge sort is considered to be a divide and conquer algorithm. Compared to Quicksort Merge sort performs fewer comparison operations, but it is especially useful if the data has to be read in sequence.

On average, Merge sort takes O(n log n) comparisons to sort n items. Worst case performance is also O(n log n), giving its performance some predictability.

# Run-through of algorithm

Consider an array of 10 integers that are not ordered. The merge sort algorithm will first try to divide the list into increasingly smaller sub lists. The smallest possible are lists of size 1, this is the base case. When this point is reached the sub lists are rejoined. For the first time – joining 2 sub lists of size 1 one comparison is performed and the result is considered sorted.



Subsequently larger sub lists must be joined. But although the sub lists are sorted they each contain elements that will exclude the possibility of the lists simply being appended. Sub lists with element [27, 38] and [3, 43]. Are put together not sorted. Instead a new sub list is created by iterating through each sub list and selecting the lowest values. The process of joining adjacent lists is continued until the entire list is returned.

#### Run-time and memory use

Merge sort may be fairly stable when it comes to its running time. However, it does require more memory use – usually on the order of (*n*) auxiliary space – compared to Heapsort. For RAM-bases data sets Quicksort is faster, but for slower accessible data (hard-drive) merge sort is still preferred. This is because the results can be stored on the hard-drive. As such merge sort is preferred on systems with a relatively low amount of RAM>

Python uses an algorithm called Timsort, which combines Merge sort and insertion sort. Java 7 now also uses Timsort as the default sorting algorithm.

# Implementation in Python and Java

Below is my implementation Python

import sys

import timeit

class Mergesort(object):

def \_\_init\_\_(self):

print()

def mergesort(self, unsorted):

if len(unsorted) > 1:

mid = len(unsorted)//2

leftHalf = unsorted[:mid]

rightHalf = unsorted[mid:]

self.mergesort(leftHalf)

self.mergesort(rightHalf)

i = 0

j = 0

k = 0

while i < len(leftHalf) and j < len(rightHalf):

if leftHalf[i] < rightHalf[j]:

unsorted[k] = leftHalf[i]

i = i + 1

else:

unsorted[k] = rightHalf[j]

j = j + 1

k = k + 1

# Now deal with the residual parts of the sub-lists, they should already be sorted

while i < len(leftHalf):

unsorted[k] = leftHalf[i]

i = i + 1

k = k + 1

while j < len(rightHalf):

unsorted[k] = rightHalf[j]

j = j + 1

k = k + 1

def main():

unsorted = [7, 3, 8, 2, 1, 9, 4, 6, 5, 0]

merge = Mergesort()

merge.mergesort(unsorted)

print(unsorted)

if \_\_name\_\_ == "\_\_main\_\_":

main()

## And now in Java

public class Mergesort {

private int[] arrayInts;

private int[] helperArray;

private int length;

public int[] sort(int[] arrayInts) {

this.arrayInts = arrayInts;

this.length = arrayInts.length;

this.helperArray = new int[this.length];

this.mergesort(0, this.length-1);

return arrayInts;

}

public void mergesort(int low, int high) {

if (low < high) {

int middle = low + (high - low) / 2;

// sort left of middle

mergesort(low, middle);

// sort right of middle

mergesort(middle+1, high);

// merge lists

merge(low, middle, high);

}

}

public void merge(int low, int middle, int high){

// copy array into helper array

for (int i = low; i <= high; i++) {

helperArray[i]= arrayInts[i];

}

int x = low;

int y = middle+1;

int z = low;

while(x <= middle && y <= high) {

if (helperArray[x] <= helperArray[y]) {

arrayInts[z] = helperArray[x];

x++;

} else {

arrayInts[z] = helperArray[y];

y++;

}

z++;

}

while(x <= middle) {

arrayInts[z] = helperArray[x];

x++;

z++;

}

}

}

And the test class that creates the array and calls the functions.

package Sorting;

import org.junit.Test;

public class TestMergesort {

@Test

public void simpleTest() {

int[] unsorted = new int[]{7, 3, 8, 2, 1, 9, 4, 6, 5, 0};

Mergesort merge = new Mergesort();

int[] sorted = merge.sort(unsorted);

for (int i = 0; i < sorted.length; i++) {

System.out.print(sorted[i] + " ");

}

}

}