

# Problem Set 13 - Solution

## Dijkstra's Algorithm, Mergesort

1. Suppose you are given this undirected graph in which the vertices are towns, and the edges are toll roads between them. The weight of an edge is the dollar amount of toll.



Use Dijkstra's shortest paths algorithm to determine the minimum toll route from A to all other cities.

- Show each step of the algorithm in tabular form. Here's the table after the initial step:

Done	D[B]	D[C]	D[D]	D[E]	D[F]	D[G]	D[H]	D[I]
A	4,A	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	8,A	$\infty$

Note that along with the distance, the "previous" vertex is also shown.

- Draw the shortest path tree induced on the graph.

### SOLUTION

Done	D[B]	D[C]	D[D]	D[E]	D[F]	D[G]	D[H]	D[I]
A	4,A	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	8,A	$\infty$
B	12,B	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	8,A	$\infty$
H	12,B	$\infty$	$\infty$	$\infty$	9,H	$\infty$	15,H	$\infty$
G	12,B	$\infty$	$\infty$	11,G	$\infty$	$\infty$	15,H	$\infty$
F	12,B	25,F	21,F	$\infty$	15,H	$\infty$	$\infty$	$\infty$
C	19,C	21,F	$\infty$	$\infty$	14,C	$\infty$	$\infty$	$\infty$
I	19,C	21,F	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
D	19,C	21,F	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
E	21,F	$\infty$						

Note that along with the distance, the "previous" vertex is also shown.

The shortest path tree induced on the graph:



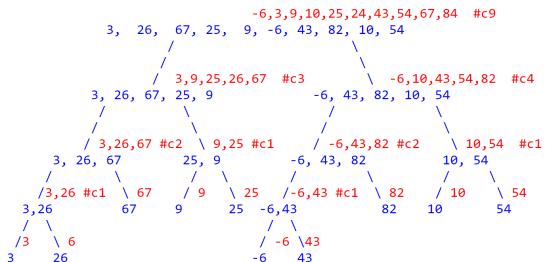
2. Trace the mergesort algorithm on the following list:

3, 26, 67, 25, 9, -6, 43, 82, 10, 54

Show the resulting recursion tree, with the to-be-sorted original and sub-lists at each node, and the number comparisons for each merge. (Assume that if there is an odd number of entries in an array, the left part has one more entry than the right after the split.)

### SOLUTION

The original and sorted result indicated above and below:



3. Mergesort works well on linked lists since it doesn't need any extra space. Given the following linked list node class:

```
public class LLNode<T extends Comparable<T>> {
    public T info;
    public LLNode<T> next;
    ...
}
```

complete the following method to "split" the linked list in half:

```
// splits the given list in half such that the return value is
// a reference to the first node of the second half. Also, the
// "next" field of the last node in the first half is set to null.
static <T extends Comparable<T> LLNode<T> split(LLNode<T> list) {
    // COMPLETE THIS METHOD
}
```

### SOLUTION

```
static <T extends Comparable<T> LLNode<T> split(LLNode<T> list) {
    if (list == null) return null;

    int size=0;
    LLNode<T> ptr;
    for (ptr=list; ptr != null; ptr = ptr.next) {
        size++;
    }
    size = (size+1)/2; ptr=list;
    while (size > 1) {
        ptr = ptr.next;
        size--;
    }
    LLNode<T> second = ptr.next;
    ptr.next = null;
    return second;
}
```

4. In Problem Set 3, #6, you saw how to merge two sorted linked lists of integers, into a single sorted list without duplicates. Here is the header of a modified version of that method, that merges lists of Comparable objects (not just ints), while preserving duplicates, if any. Complete this method using recursion. Your method should recycle the nodes in the original lists (no new nodes should be created).

```
// merge the lists l1 and l2 into a single linked list, whose
// first node is referenced by the return value - no additional
// linked list nodes are used
static <T extends Comparable<T>> LLNode<T> merge(LLNode<T> l1, LLNode<T> l2) {
    // COMPLETE METHOD USING RECURSION, NO NEW NODES TO BE CREATED
}
```

Using this merge solution, and the solution to the split in the previous problem, complete the mergesort implementation:

```
// Sorts the input linked list using mergesort, and returns the front of
// the sorted linked list. DOES NOT CREATE ANY ADDITIONAL NODES.
```

```

public static <T extends Comparable<T>
LLNode<T> mergesort(LLNode<T> list) {
    // COMPLETE THIS METHOD
}

SOLUTION

// merge the lists l1 and l2 into a single linked list, whose
// first node is referenced by the return value - no additional
// linked list nodes are used
static <T extends Comparable<T>> LLNode<T> merge(LLNode<T> l1, LLNode<T> l2) {
    // COMPLETE METHOD USING RECURSION, NO NEW NODES TO BE CREATED
    if (l1 == null) { return l2; }
    if (l2 == null) { return l1; }
    if (l1.info.compareTo(l2.info) < 0) {
        l1.next = merge(l1.next, l2);
        return l1;
    } else {
        l2.next = merge(l1, l2.next);
        return l2;
    }
}

public static <T extends Comparable<T>
LLNode<T> mergesort(LLNode<T> list) {
    // empty or single node list
    if (list == null || list.next == null) {
        return list;
    }
    // split
    LLNode<T> second = split(list);

    // recursive calls
    list = mergesort(list);
    second = mergesort(second);

    // merge
    return merge(first, second);
}

```