

Seminar: Turing Award Winners - Implementing merge sort in CLU, a programming language invented by Barbara Liskov's Research Team

Nakarin Srijumrat

December 16, 2020

1 Introduction

As a part of the Seminar, this document represents my submission for the programming project. My assignment was to implement the merge sort algorithm invented by John von Neumann in 1945 using the programming language CLU. CLU was developed and released by Barbara Liskov and her team in 1975 and is considered a major contribution in the field of object-oriented programming.

2 The Merge Sort Algorithm

Merge sort is a sorting algorithm of the divide-and-conquer category. The algorithm takes an array consisting of n elements and divides the list recursively into smaller sublists until all sublists are of size 1. The resulting list will be subsequently merged in ascending element order until one final list remains: the original list in a sorted state.

It is a stable algorithm, meaning the order of equal elements is preserved. Its time complexity is $n \log n$ for best-, average-, and worst-case scenarios.

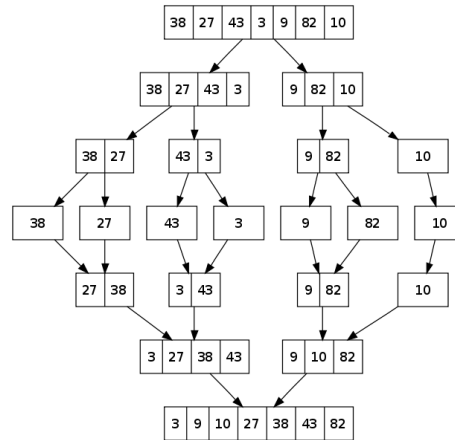


Figure 1: Visualization of the merge sort algorithm

3 Implementation

```
1 mergesort = cluster is create , merge , mergeSort
2
3 % default constructor
4 create = proc() returns (cvt)
5     return()
6 end create
7
8 % merges two subarrays of arr[]
9 merge = proc(arr: array[int], l: int, m: int, r: int)
10     int: n1 := m - l + 1
11     int: n2 := r - m
12
13
14 % create temp arrays
15 array[int]: L := array[int]$create(n1)
16 array[int]: R := array[int]$create(n2)
17
18 % copy data to temp arrays L[] and R[]
19 for x: int in L$indexes() do
20     L[x] = arr[l + x]
21 end
22 for y: int in R$indexes() do
23     L[y] = arr[m + 1 + y]
24 end
25
26 % merge temp arrays back to arr[l...r]
27 int: i := 0 % index first subarray
28 int: j := 0 % index second subarray
29 int: k := 0 % index merged
30
31 while i < L$size() and j < R$size() do
32     if L[i] < R[j] then
33         arr[k] := L[i]
34         i := i + 1
35     else
36         arr[k] := R[j]
37         j := j + 1
38     end
39
40     k := k + 1
41 end
42
43 % copy rest of L[]
44 while i < L$size() do
45     arr[k] := L[i]
46     i := i + 1
47     k := k + 1
48 end
```

```

49
50         % copy rest of R[]
51         while i < L$size() do
52             arr[k] := L[i]
53             i := i + 1
54             k := k + 1
55         end
56     end merge
57
58     sort = proc(arr: array[int], l: int, r: int)
59         % recursive invocation
60         if l < r then
61             int: m := l + (r - l) / 2
62             sort(arr, l, m)
63             sort(arr, m + 1, r)
64             merge(arr, l, m, r)
65         end
66     end sort
67
68 end mergesort

```

This implementation uses all three module types: two procedures are implemented within a cluster of the name *mergesort* and iterators are used to copy values from one array to another.

The process of applying the code above would be to create an object of the type *mergesort* and then run the sort function with the corresponding parameters:

```

1 mergesort: ms := mergesort$create()
2 array[int]: collection := array[int]$create[2, 5, 23, 1, 7, 4]
3 ms$sort(collection, 0, collection$size())

```

4 Conclusion

Considering the current state of object-oriented programming, the CLU code shows similarity to modern languages like Python in regards to style and structure. For a programmer, CLU leaves a strong impression of being a high-level programming language with the code not straying too far from pseudo-code.

For being a near 50 year old programming language, CLU has proved early on that this object-oriented is going to leave a mark on the programming landscape.