codecademy

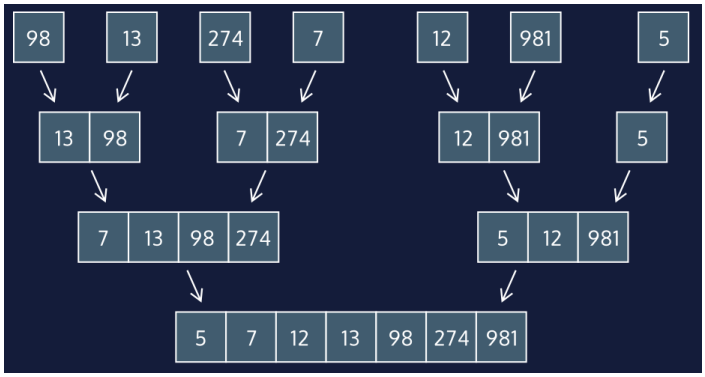
Merge Sort

Merge Sort Merging

Merge Sort is a divide and conquer algorithm. It consists of two parts:

1) splitting the original list into small
2) merging back the presorted 1-element l

The merging portion is iterative and takes 2 sublists. The first element of the left sublist is compared to the first element of the right sublist. If it is smaller, it is added to a new sorted list, and removed from the left sublist. If it is bigger, the first element of the right sublist is added instead to the sorted list and then removed from the right sublist. This is repeated until either the left or right sublist is empty. The remaining non-empty sublist is appended to the sorted list.



Big-O Runtime for Merge Sort

The Merge Sort algorithm is divided into two parts. The first part repeatedly splits the input list into smaller lists to eventually produce single-element lists. The best, worst and average runtime for this part is Θ(log N). The second part repeatedly merges and sorts the single-element lists to twice its size until the original input size is achieved. The best, worst and average runtime for this part is Θ(N). Therefore, the combined runtime is Θ(N log N).

Merge Sort Implementation in Python

We can implement the Merge Sort algorithm in Python using two functions, `merge_sort(lst)`, the main function and `merge(left, right)`, a helper function.

```
def merge_sort(lst):
  if len(lst) <= 1:
    return lst
  middle = len(lst) // 2
  left = lst[:middle]
  right = lst[middle:]
  sleft = merge_sort(left)
  sright = merge_sort(right)
  return merge(sleft, sright)

def merge(left, right):
  result = []
```

```
while (left and right):
  if left[0] < right[0]:
    result.append(left[0])
    left.pop(0)
  else:
    result.append(right[0])
    right.pop(0)
if left:
  result += left
if right:
  result += right
return result
```

Print Share