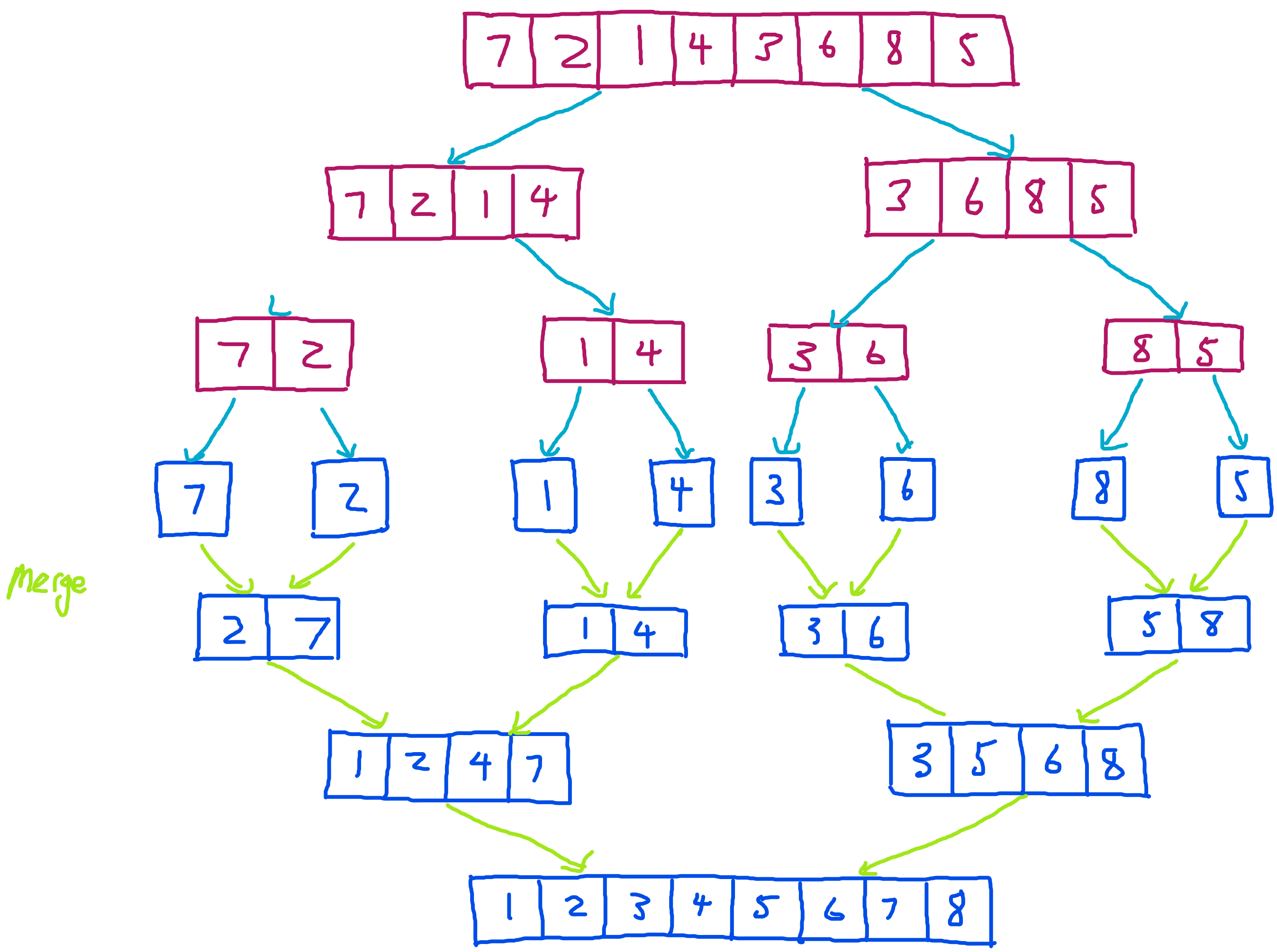
Merge Sort

```
mergeSort(arr):
    if len(arr)==0:
        return arr
    first_half ← first half of arr
    second_half ← second half of arr
    sorted ← merge(mergeSort(first_half), mergeSort(second_half))
    return sorted

merge(sorted1, sorted2):
    i,j ←
    newArr ← empty array
    while i < len(arr1) and j < len(arr2):
        if sorted1[i] ≤ sorted2[j]
            newArr[i+j] ← sorted1[i]
            i = i+1
        else:
            newArr[i+j] ← sorted2[j]
            j = j+1
    return newArr
```

In merge sort, we recursively divide the array by half, then call the same procedure on each half. Afterwards, both halves will be sorted (recursive leap of faith). We then merge the two halves (go thru both lists, compare leftmost unadded items of each list, then add the smaller value.

See example below on why this works.

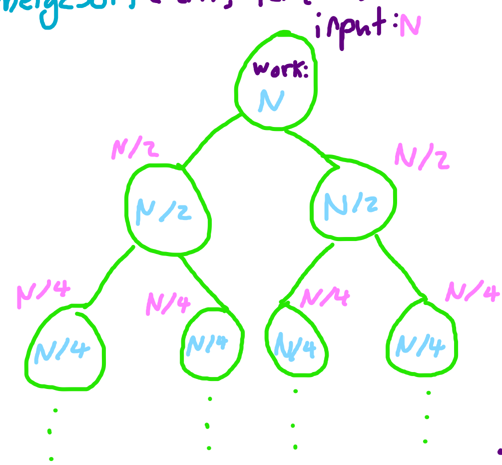


Runtime? $\Theta(N \log N)$

① Realize that merge takes $\Theta(N)$ time.

② There are two recursive calls in mergeSort, each call with $\frac{1}{2}$ the input

mergeSort(arr), len(arr)=N



Level	Work
0	N
1	$\frac{N}{2}+\frac{N}{2}=N$
2	$\frac{N}{4}+\frac{N}{4}+\frac{N}{4}+\frac{N}{4}=N$
⋮	⋮
$\log N$	N

$$N \cdot \log N = \boxed{\Theta(N \log N)}$$