Merge Sort

Motivation. What is the problem with elementary sorts?

Elementary sorts are too slow for large arrays

- Assume using insertion sort and the cost of one comparison operation ~ 0.1 ns

Elements	Comparisons	Time
10K	~25M	Instant
1M	~250B	~25 second
10M	~25T	~42 minutes
100M	~2,5Q	~3 days
1B	~250Q	~289 days

Can we do better?

- Yes, it seems so.
  - Merge Sort
  - Quick Sort
  - Heap Sort

But first let’s talk about recursion

Recursion

- Recursion in computer science is a method of solving a problem where the solution depends on solutions to smaller instances of the same problem (as opposed to iteration)
- A recursive function is a function that calls itself during its execution

Recursion. Example

- Computing factorial

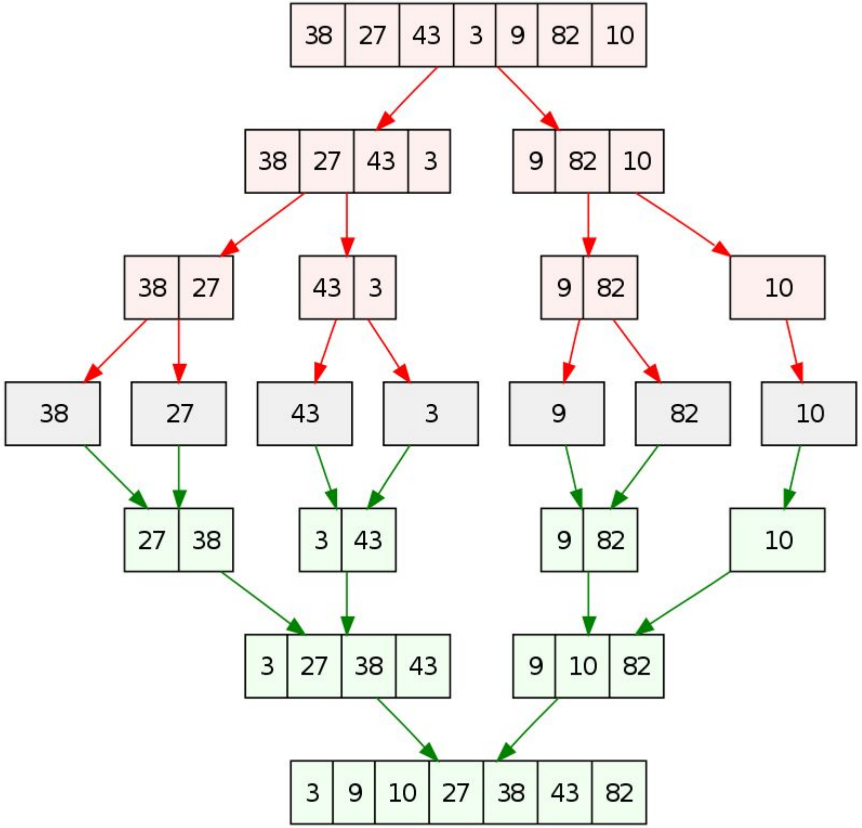
```
def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)


print(factorial(5)) # Output: 120
```

Merge sort. Idea

- Merge sort is based on a “divide-and-conquer” idea:
  - Divide array into two halves
  - Sort them separately
  - Then merge them

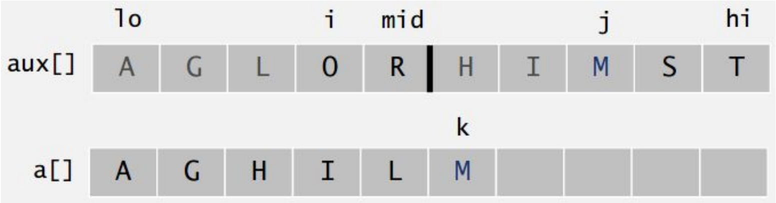
Merge sort. Example



Method merge() implementation

```python
def merge(input_arr, aux_arr, lo, mid, hi):
    for k in range(lo, hi + 1):
        aux_arr[k] = input_arr[k]

    i, j = lo, mid + 1
    for k in range(lo, hi + 1):
        if i > mid:
            input_arr[k] = aux_arr[j]
            j += 1
        elif j > hi:
            input_arr[k] = aux_arr[i]
            i += 1
        elif aux_arr[j] < aux_arr[i]:
            input_arr[k] = aux_arr[j]
            j += 1
        else:
            input_arr[k] = aux_arr[i]
            i += 1
```



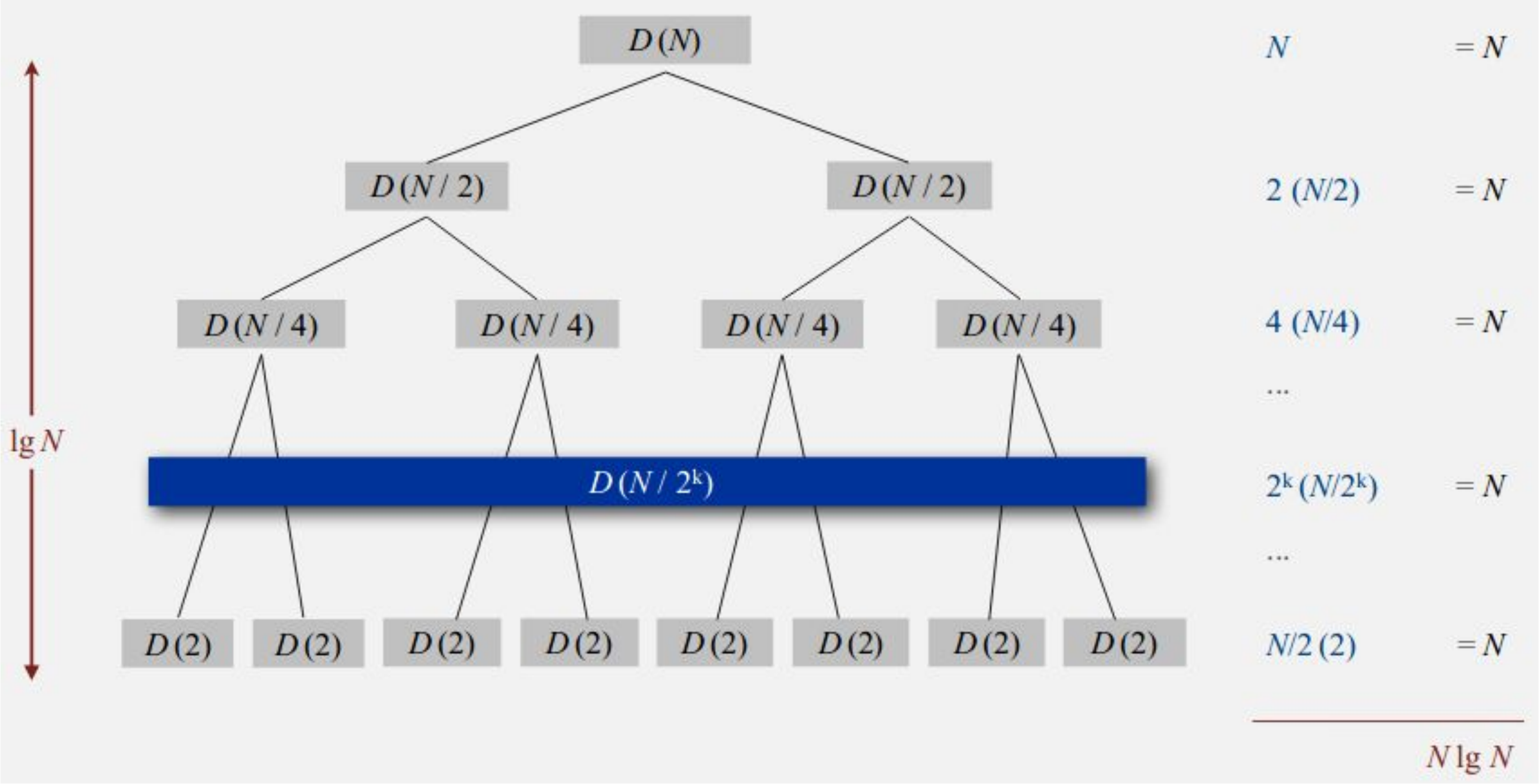
Merge sort implementation

```python
def sort_req(input_arr, aux_arr, lo, hi):
   if hi <= lo:
       return

   mid = lo + (hi - lo) // 2
   sort_req(input_arr, aux_arr, lo, mid)
   sort_req(input_arr, aux_arr, mid + 1, hi)
   merge(input_arr, aux_arr, lo, mid, hi)


def merge_sort(input_arr):
   aux_arr = [None] * len(input_arr)
   sort_req(input_arr, aux_arr, 0, len(input_arr) - 1)
```

Merge sort. Analysis



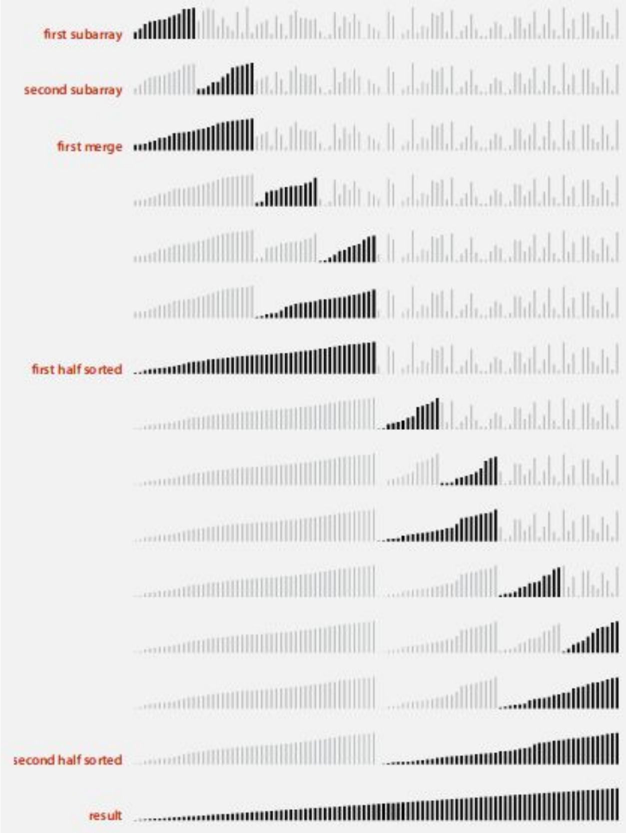
Merge sort. Performance

- Worst-case: O(n log(n))
- Best-case: O(n log(n))
- On average: O(n log(n))

Merge sort. Important characteristics

- Merge Sort is a stable sort which means that the same element in an array maintain their original positions with respect to each other
- The space complexity of Merge sort is O(n). This means that this algorithm takes a lot of space

Merge sort. Visualization



Merge sort implementation improvements

- Use insertion sort for small subarrays
  - Merge sort has too much overhead for tiny subarrays
  - Cutoff to insertion sort for ~7 items
- Stop if already sorted
  - If biggest item in left half ⩽ smallest item in right half
- Eliminate the copy to the auxiliary array
  - By switching the role of the input and auxiliary array in each recursive call

Merge sort vs. Insertion sort

- Assume the cost of one comparison or array access operation ~ 0.1 ns

Elements	Insertion sort	Merge sort
10K	Instant	Instant
1M	~25 second	Instant
10M	~42 minutes	Instant
100M	~3 days	~1.6 seconds
1B	~289 days	~18 seconds