Merge sort

Merge Sort
Idea

- Suppose we only know how to merge two sorted sets of elements into one
  - $Merge\ \{1, 5, 9\}\ with\ \{2, 11\} \rightarrow\ \{1, 2, 5, 9, 11\}.$

- Idea of merge sort:
  - Merge each pair of elements into sets of 2.
  - Merge each pair of sets of 2 into sets of 4.
  - Repeat previous step for sets of 4 and so on.
  - Final step: merge 2 sets of *n/2* elements to obtain a fully sorted set.

Merge Sort
Divide and Conquer method

- A powerful problem solving technique.

- Divide-and-conquer method solves problem in the following steps:
  - Divide step:
    - Divide the large problem into smaller problems.
    - Recursively solve the smaller problems.
  - Conquer step:
    - Combine the results of the smaller problems to produce the result of the larger problem.

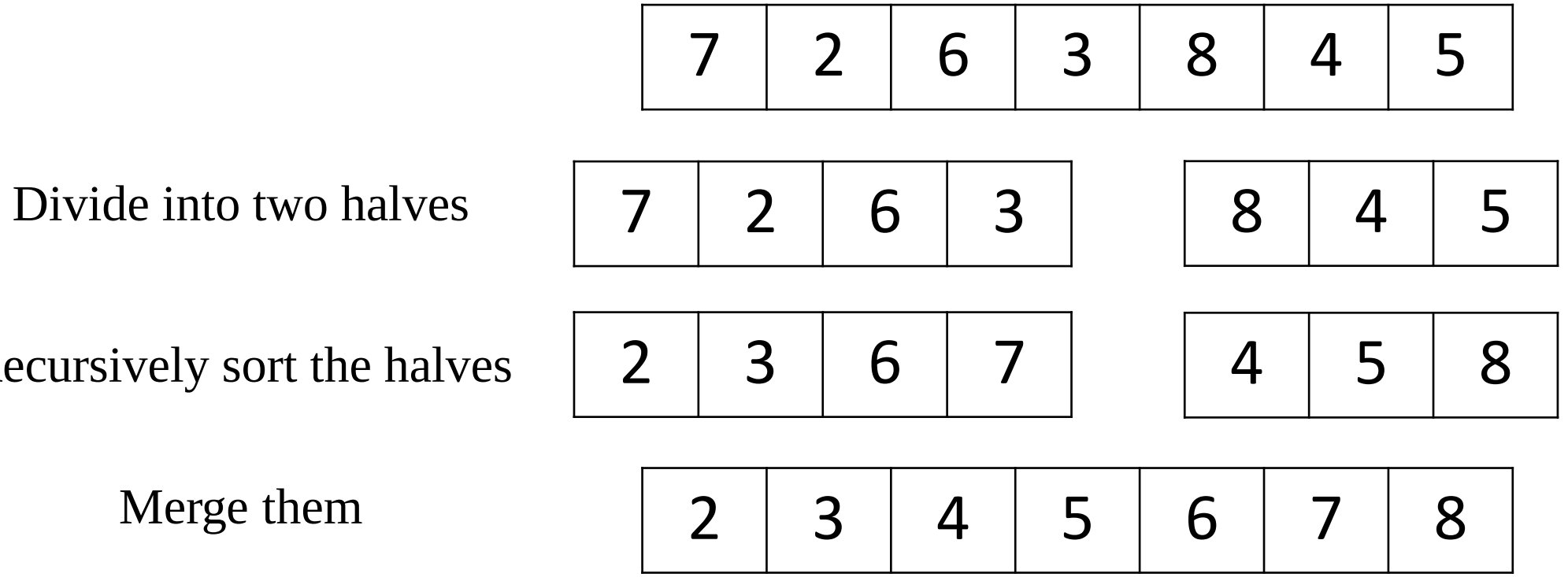
Merge Sort
Divide and Conquer method

- Merge Sort is a divide-and-conquer sorting algorithm.

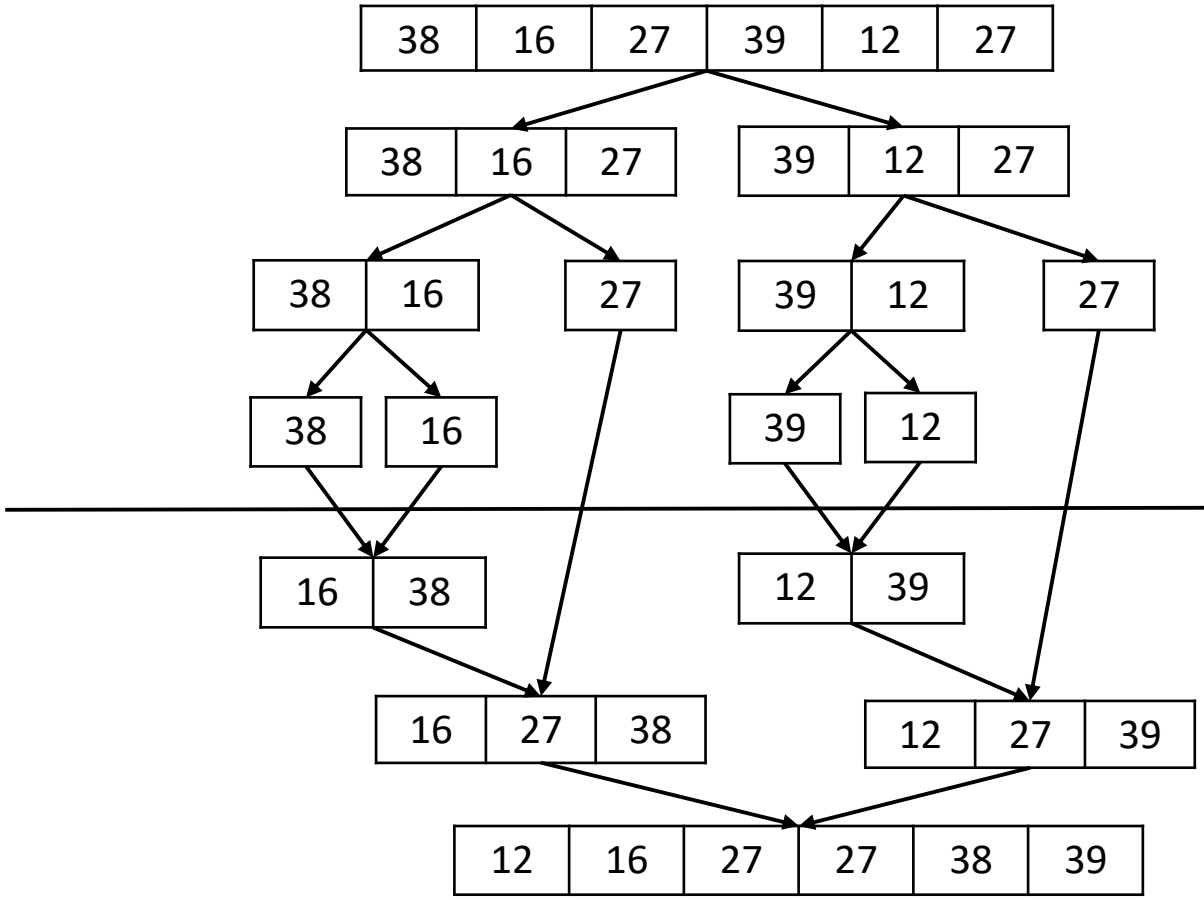
- Divide step:
  - Divide the array into two (almost equal) halves.
  - Recursively sort the two halves.

- Conquer step:
  - *Merge* the two halves to form a sorted array.

Merge Sort
Divide and Conquer method



Merge Sort Example



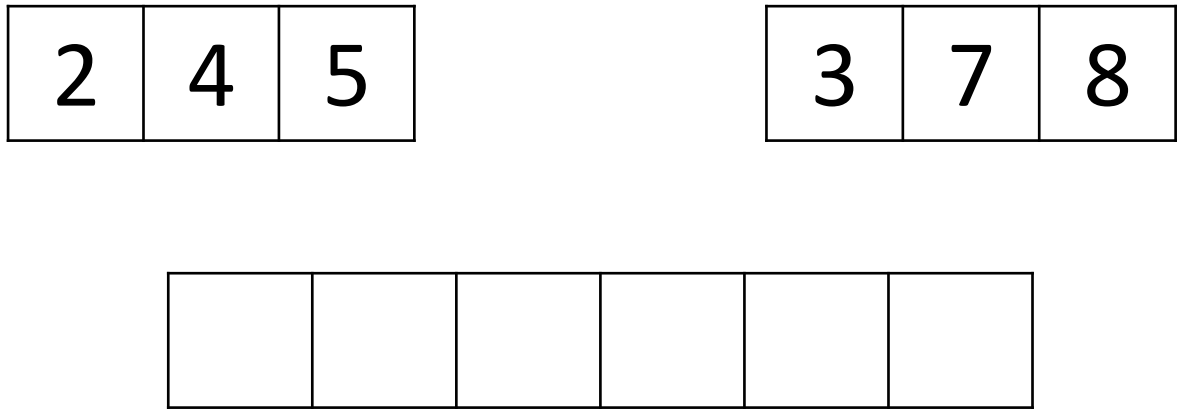
Divide Phase:
Recursive call to mergeSort().

Conquer phase:
Merge steps.

Merge Sort Implementation

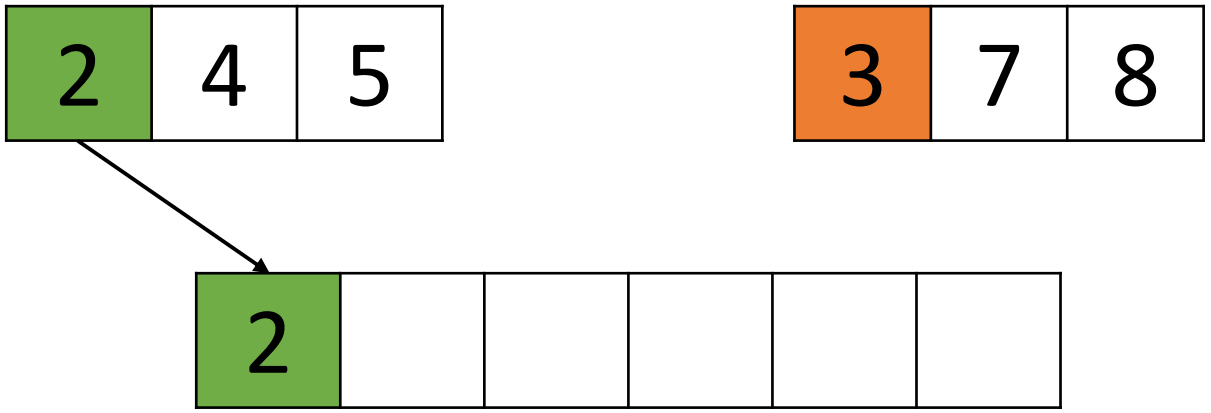
```
void merge_sort(vector<int>& arr)
{
    if (arr.size() <= 1)
        return;
    vector<int> left, right;
    copy(arr.begin(), arr.begin() + arr.size() / 2, back_inserter(left));
    copy(arr.begin() + arr.size() / 2, arr.end(), back_inserter(right));
    merge_sort(left);
    merge_sort(right);

    arr = merge(left, right);
}
```

Merge Sort
Merge example



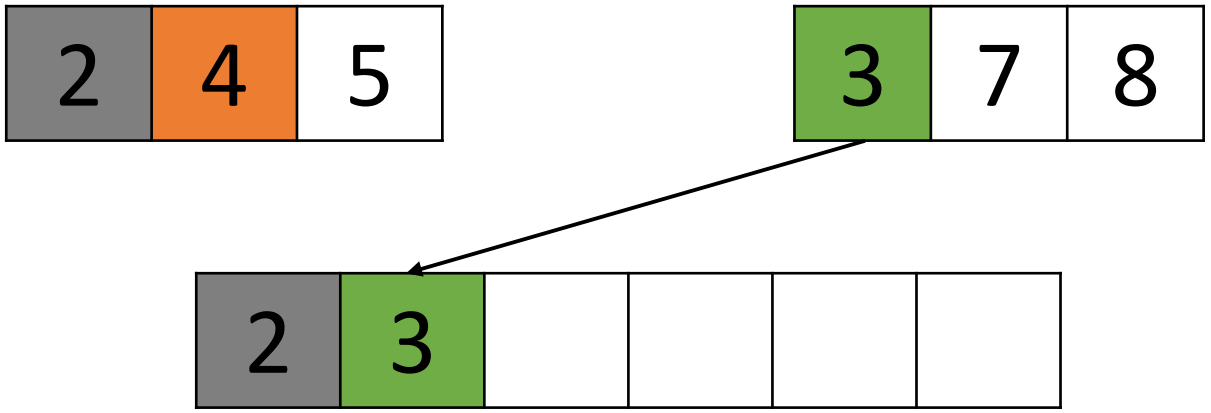
We want to merge 2 sorted arrays

Merge Sort
Merge example



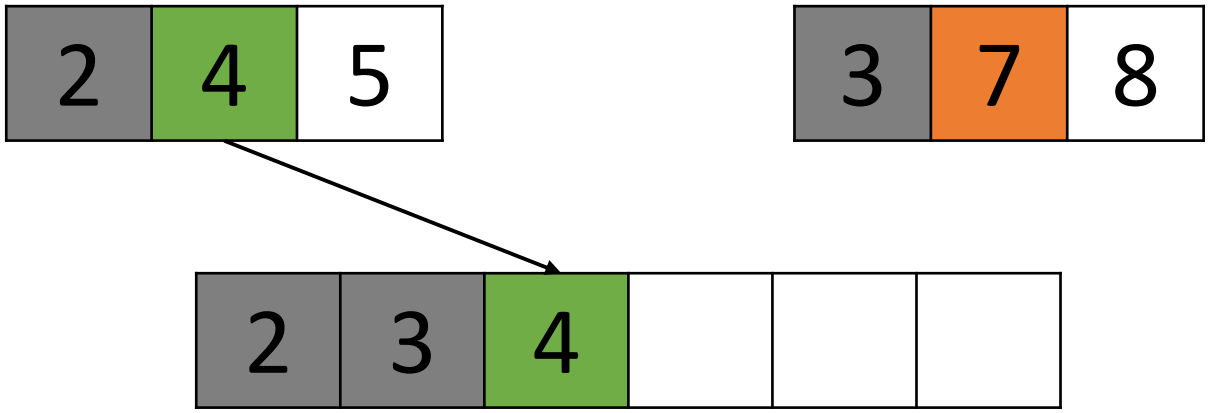
On the first step we take first element from first array, because 2 < 3

Merge Sort
Merge example



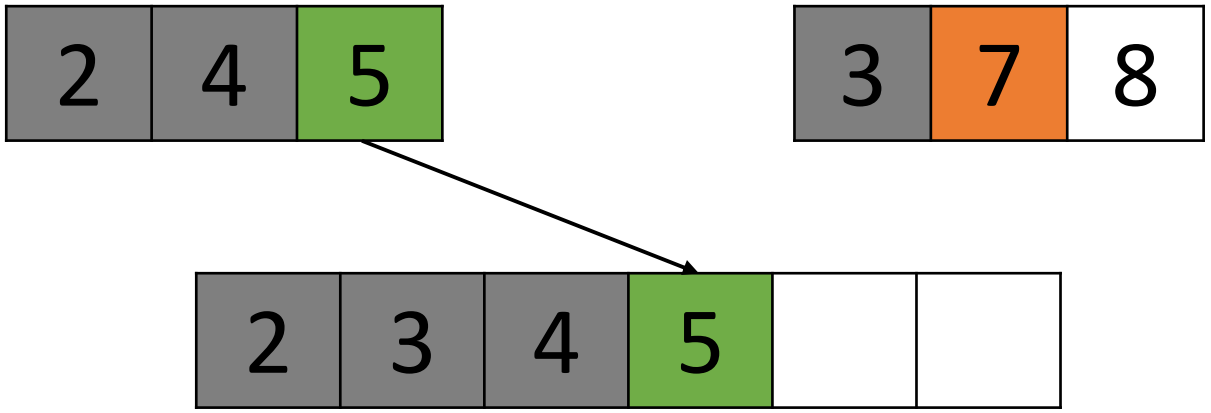
Then first element from second array

Merge Sort
Merge example



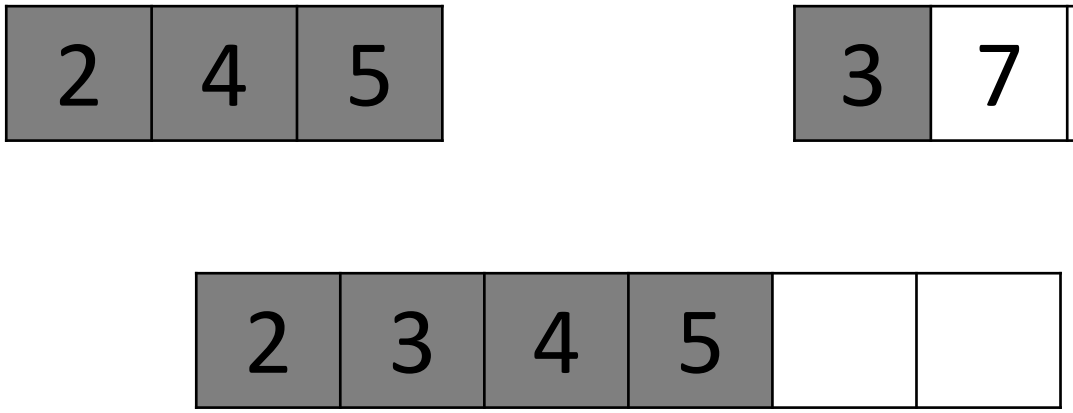
Then second element from first array

Merge Sort
Merge example



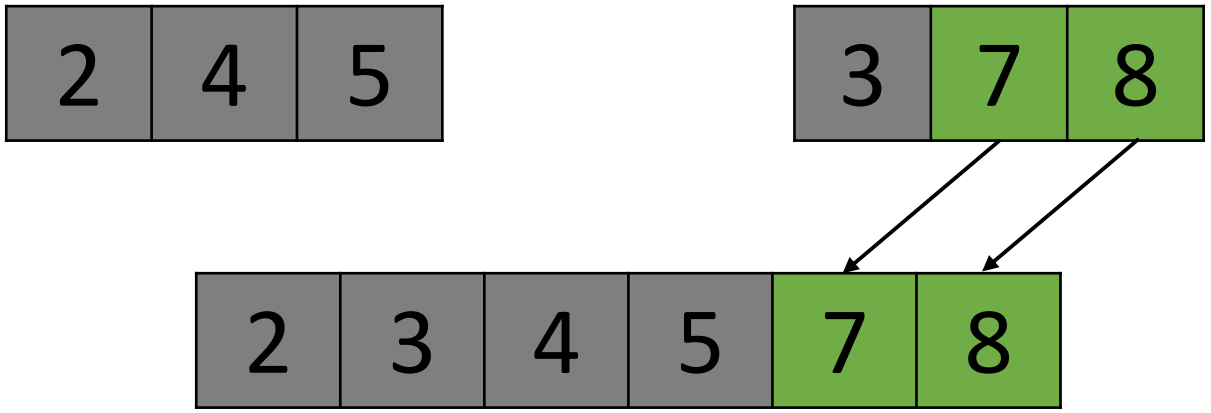
Then third element from first array

Merge Sort
Merge example



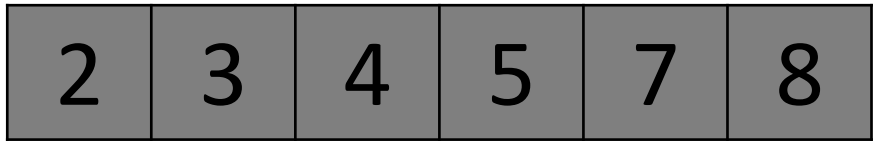
First array has ended, so simply add the remained part of second array to our merged array

Merge Sort
Merge example



First array has ended, so simply add the remained part of second array to our merged array

Merge Sort
Merge example



Arrays merged

Merge Sort
Merge implementation

```
vector<int> merge(const vector<int>& left,
                  const vector<int>& right)
{
    vector<int> merged;
    int left_index = 0, right_index = 0;
    while (left_index < left.size() &&
           right_index < right.size())
    {
        if (left[left_index] < right[right_index])
            merged.push_back(left[left_index++]);
        else
            merged.push_back(right[right_index++]);
    }
    while (left_index < left.size())
        merged.push_back(left[left_index++]);
    while (right_index < right.size())
        merged.push_back(right[right_index++]);
    return merged;
}
```

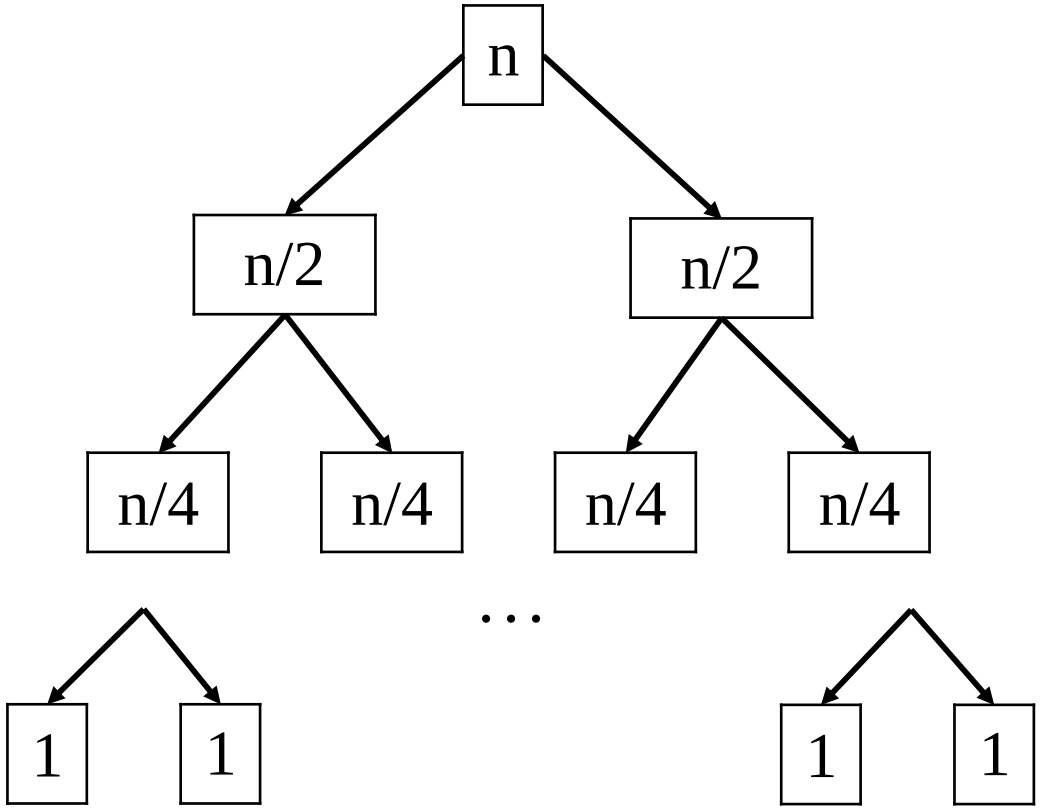
Merge Sort Analysis

Level 0: mergeSort n items

Level 1: mergeSort n/2 items

Level 2: mergeSort n/4 items

Level (lg n): mergeSort 1 item



Level 0: 1 call to mergeSort

Level 1: 2 calls to mergeSort

Level 2: 4 calls to mergeSort

Level $(\log_2(n))$: $2^{\log_2(n)} (= \boldsymbol{n})$ calls to mergeSort

$$\frac{n}{2^k} = 1 \rightarrow n = 2^k \rightarrow k = \log_2(n)$$

Merge Sort Analysis

- Level 0: 0 call to merge()
- Level 1: 1 calls to merge() with $n/2$ items in each half
  - $O(1 \cdot 2 \cdot n/2) = \mathrm{O(n)}$ time.
- Level 2: 2 calls to merge() with $n/2^2$ items in each half,
  - $O(2 \cdot 2 \cdot n/2^2) = \mathrm{O(n)}$ time.
- Level 3: 2^2 calls to merge() with $n/2^3$ items in each half,
  - $O(2^2 \cdot 2 \cdot n/2^3) = \mathrm{O(n)}$ time.
- …
- Level $(\lg n)$: $2^{\log(n)-1} (= n/2)$ calls to merge() with $n/2^{\log(n)}$ $(= 1)$ item in each half, O(n) time.
- Total time complexity = $O(n \lg(n))$.
- Optimal comparison-based sorting method.

Merge Sort
Pros and Cons

- Pros:
  - The performance is guaranteed, i.e. unaffected by original ordering of the input.
  - Suitable for extremely large number of inputs, can operate on the input portion by portion.

- Cons:
  - Requires additional storage during merging operation.
    - O(n) extra memory storage needed.