Merge Sort

https://shorturl.at/CbvYO

What is Sort?

- Sorts you might have learned
  - Bubble Sort
  - Insertion Sort
  - Selection sort
  - …
- However, each above takes $O(N^2)$ for worst cases
- Can it be much faster?

Divide & Conquer 分治法

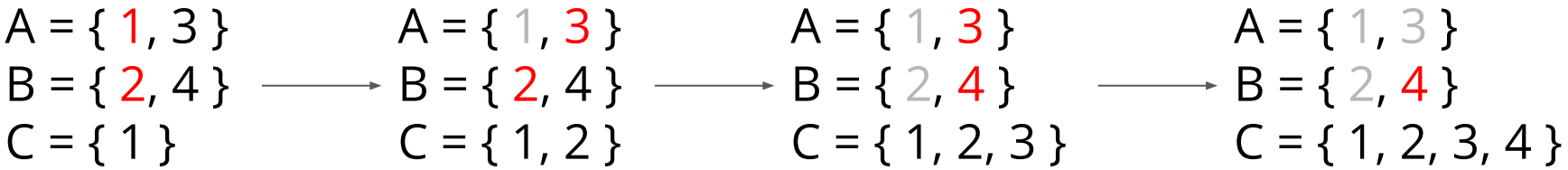
- Divide
  - Partition the target problem into sub-problems
  - Usually sub-problems have same (input) size
- Conquer
  - Solve the sub-problems recursively
  - Merge the sub-problems to solve the target problem

Merging two subarrays

Iteratively insert the minimum into a new array

Time: O(n), n is the size of array

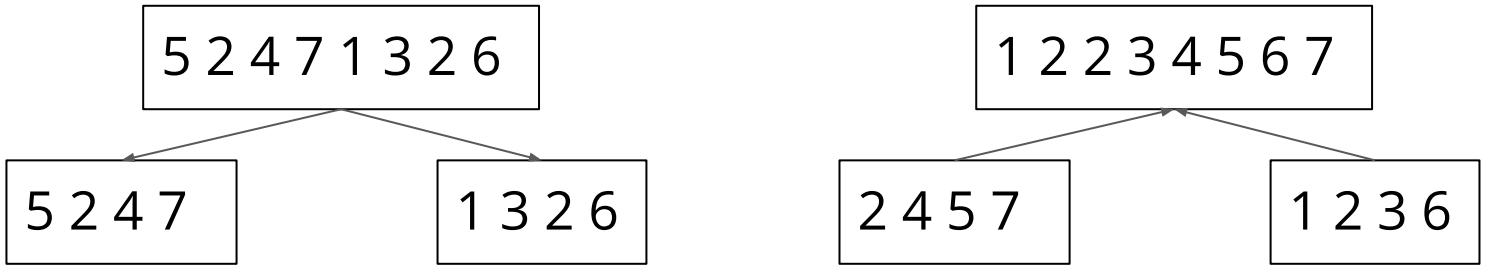
A = { 1, 3 }
B = { 2, 4 }
C = {}



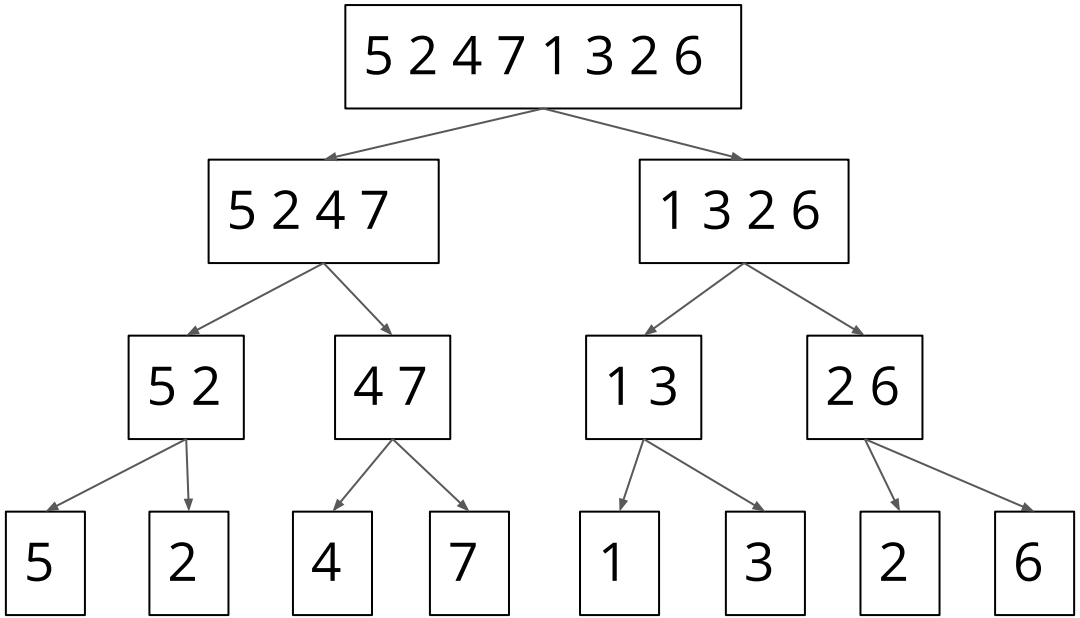
Merge Sort

Divide: Partition the input array into two

Conquer: Recursively sort the subarrays. Merge the sorted subarrays.

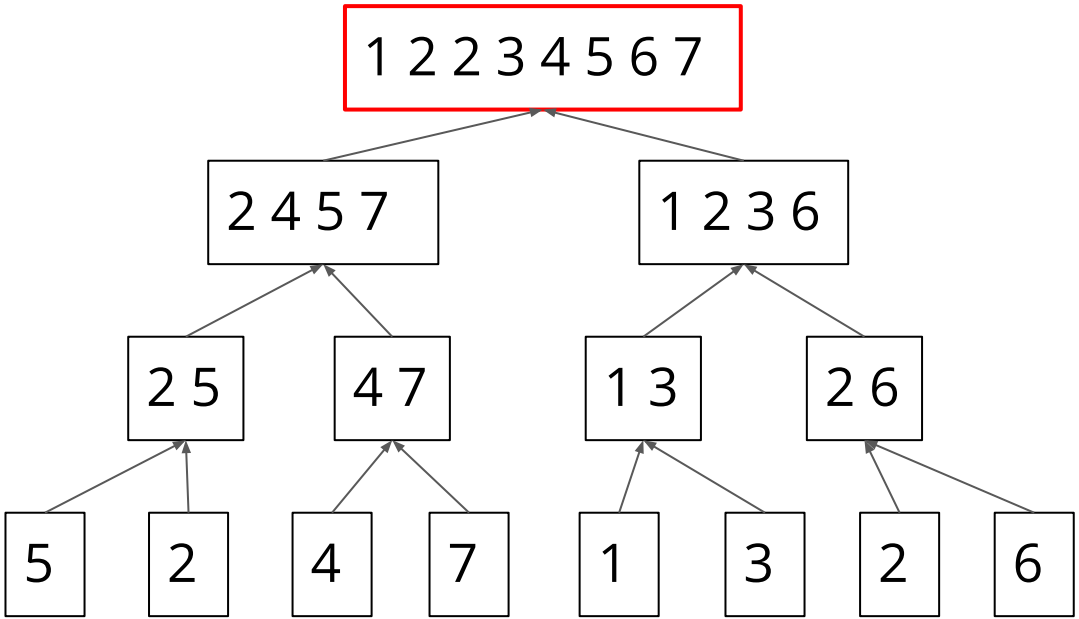


Divide:



←sorted arrays with length = 1

Conquer:



Merge Sort Time Complexity

Let T(n) be runtime for merge sort n numbers

$$T(n) = \begin{cases} c & \text{if } n = 1, \\ 2T(n/2) + cn & \text{if } n > 1, \end{cases}$$

sort an array with length one requires constant runtime

Partition to two subarray with length = n/2 Sort them recursively

merging two sorted array with length = n/2

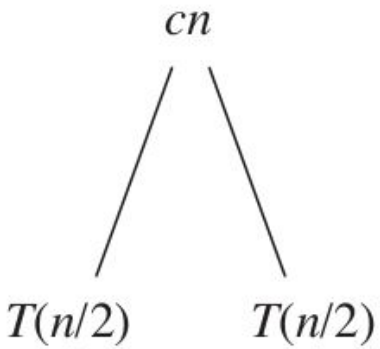
Merge Sort Time Complexity

$$T(n) = \begin{cases} c & \text{if } n = 1 , \\ 2T(n/2) + cn & \text{if } n > 1 , \end{cases}$$

$$T(n)$$

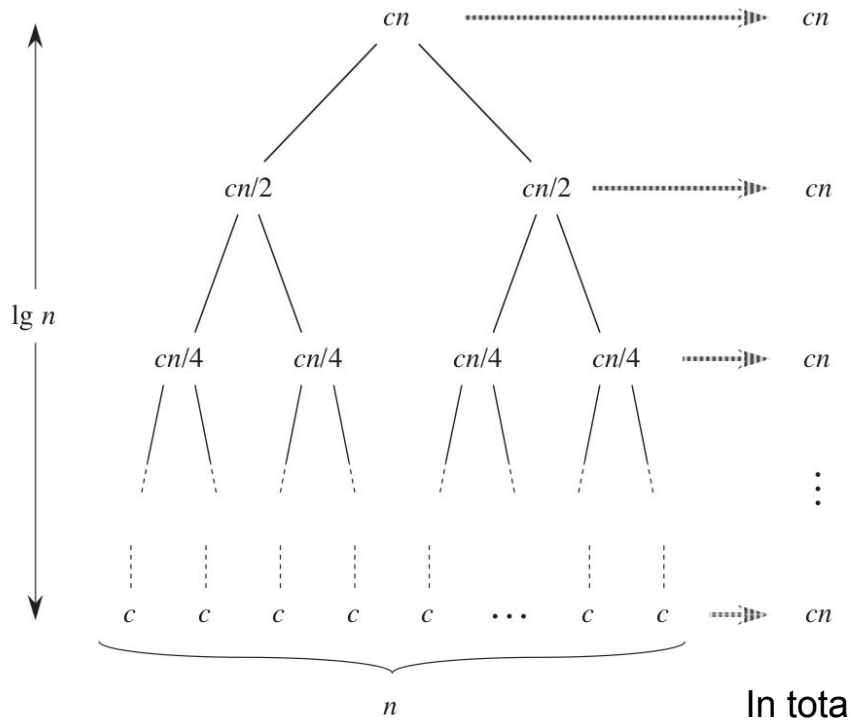
Merge Sort Time Complexity

$$T(n) = \begin{cases} c & \text{if } n = 1 , \\ 2T(n/2) + cn & \text{if } n > 1 , \end{cases}$$



Merge Sort Time Complexity

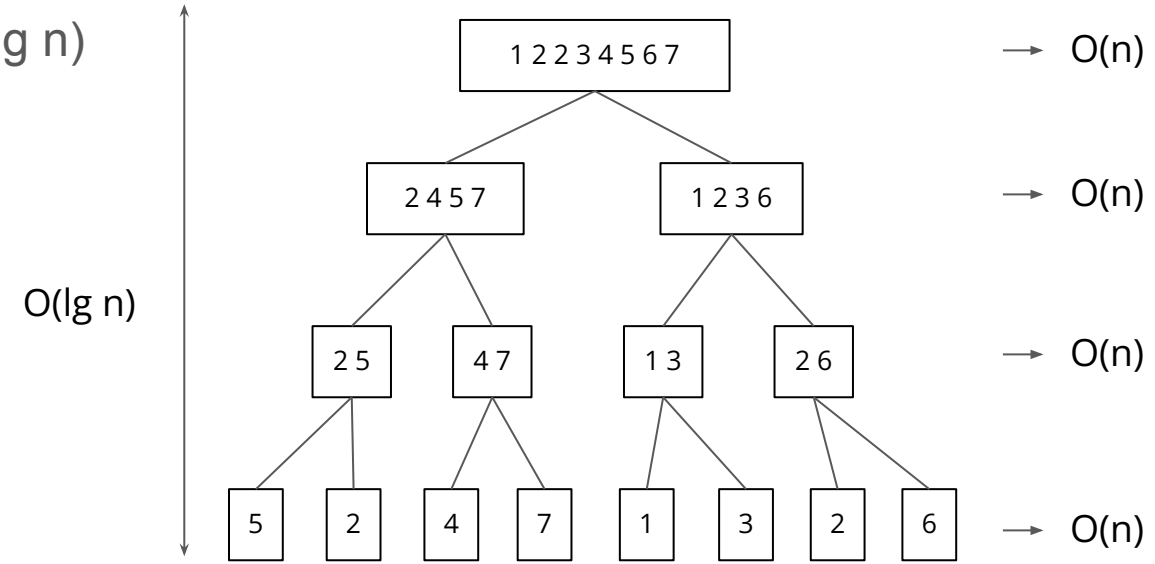
$$T(n) = \begin{cases} c & \text{if } n = 1 , \\ 2T(n/2) + cn & \text{if } n > 1 , \end{cases}$$



In total cn * lg n = O(n lg n)

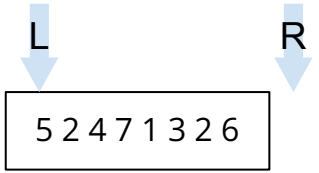
Merge Sort

- Recursively sort two subarrays
- Merge two sorted subarrays
- Time complexity: O(n lg n)
  - Recursion depth: O(lg n)
  - Merge: O(n)



Merge Sort Implementation

```
using Iter = vector<int>::iterator;
void merge_sort(Iter L, Iter R) {
  if (L+1 >= R) {
    return;
  } else {
    Iter M = L + (R - L) / 2;
    merge_sort(L, M);
    merge_sort(M, R);
    merge(L, M, R);
  }
}
```



[L, R) be left-closed, right-open interval

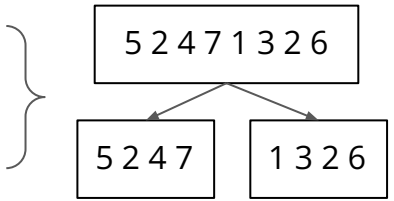
Merge Sort Implementation

```
using Iter = vector<int>::iterator;
void merge_sort(Iter L, Iter R) {
  if (L+1 >= R) {
    return;
  } else {
    Iter M = L + (R - L) / 2;
    merge_sort(L, M);
    merge_sort(M, R);
    merge(L, M, R);
  }
}
```

Small sub-problem that is easy to solve

Merge Sort Implementation

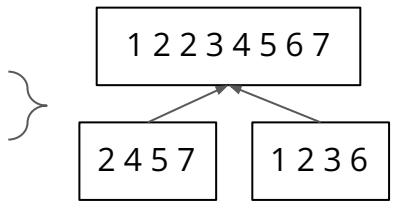
```
using Iter = vector<int>::iterator;
void merge_sort(Iter L, Iter R) {
  if (L+1 >= R) {
    return;
  } else {
    Iter M = L + (R - L) / 2;
    merge_sort(L, M);
    merge_sort(M, R);
    merge(L, M, R);
  }
}
```



Divide
Conquer: solve sub-problems recursively

Merge Sort Implementation

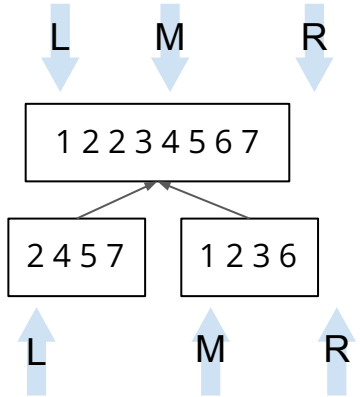
```
using Iter = vector<int>::iterator;
void merge_sort(Iter L, Iter R) {
  if (L+1 >= R) {
    return;
  } else {
    Iter M = L + (R - L) / 2;
    merge_sort(L, M);
    merge_sort(M, R);
    merge(L, M, R);
  }
}
```



Conquer: merge sub-problems

Merge Sort Implementation

```
void merge(Iter L, Iter M, Iter R) {
  static vector<int> buff;
  buff.clear();
  Iter i = L, j = M;
  while (i != M && j != R) {
    if (*i < *j) {
      buff.emplace_back(*i), i++;
    } else {
      buff.emplace_back(*j), j++;
    }
  }
  for (; i != M; i++) buff.emplace_back(*i);
  for (; j != R; j++) buff.emplace_back(*j);
  copy(buff.begin(), buff.end(), L);
}
```



Class Implementation

Merge Sort:

https://ideone.com/3iI6m2