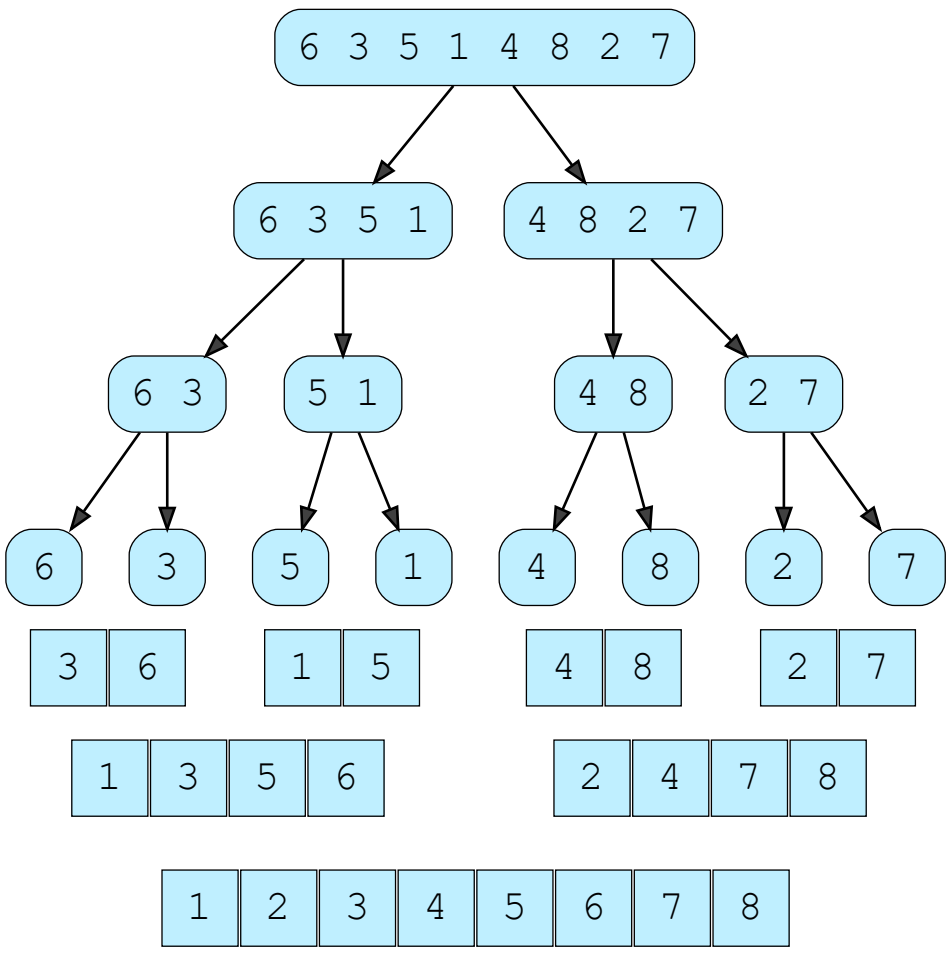
Merge Sort

In this lesson, we'll learn how merge sort works and see its implementation.

We'll cover the following

Divide and conquer

Merge is a divide and conquer algorithm. The array to be sorted is divided into two halves and each half becomes sorted by recursively splitting into more halves. After the two halves become sorted, they're merged into a sorted array.



Merge sort has two parts:

- Divide - Split the array recursively into two halves until you are left with single elements
- Conquer - Merge the smaller sorted arrays

Code

```
#include <iostream>
using namespace std;

void merge(int arr[], int start, int mid, int end) {
  int sz1 = mid - start + 1;
  int sz2 = end - mid;
  int L[sz1], R[sz2]; // temp arrays, we'll copy the 2 subarrays here

  for (int i = 0; i < sz1; i ++) L[i] = arr[i + start];
  for (int i = 0; i < sz2; i ++) R[i] = arr[i + mid + 1];

  // Merge sorted arrays L and R into arr[start...end]
  int l = 0, r = 0, a = start;
  while (l < sz1 && r < sz2) {
    if (L[l] < R[r])
      arr[a++] = L[l++];
    else
      arr[a++] = R[r++];
  }

  // one of L or R will exhaust first, we'll copy the remaining to arr
  while (l < sz1)  // R exhauseted
    arr[a++] = L[l++];
  while (r < sz2)  // L exhauseted
    arr[a++] = R[r++];
}

void merge_sort(int arr[], int start, int end) {
  if (start >= end)
    return;

  int mid = (start + end) / 2;

  merge_sort(arr, start, mid);
  merge_sort(arr, mid + 1, end);

  // Merge sorted arrays arr[start...mid] and arr[mid+1...end]
  merge(arr, start, mid, end);
}

int main() {
  int N = 8;
  int arr[N] = {6, 3, 5, 1, 4, 8, 2, 7};

  merge_sort(arr, 0, N - 1);

  for (int i = 0; i < N; i++)
    cout << arr[i] << " ";
```

```
  return 0;
}
```

Explanation

Starting with an array of size 8, recursively split it into two halves continuously until the size of the array becomes 1. This is our exit condition. Since the array of size 1 is sorted, we can just return from there.

`merge_sort(arr, start, end)` calls itself with smaller arrays `merge_sort(arr, start, mid)` and `merge_sort(arr, mid+1, end)`. The two subarrays are sorted when the recursive call ends so we end up with two sorted arrays, `arr[start...mid]` and `arr[mid+1...end]`.

Now we merge them into a sorted array in place, so that after merging, `arr[start...end]` is sorted.

Time complexity

We split into two halves every time, so there are $logN$ levels in the recursion tree. Each level has N elements and merging arrays in one level is $O(N)$. Overall, we do this merging in $logN$ time.

Time complexity: $O(NlogN)$

In the next lesson, we'll discuss *quick-sort*, another $O(NlogN)$ sorting algorithm.