Lab 6 - Merge Sort

Problem 1:

Explain what you think the worst-case, big-Oh complexity and the best-case, big-Oh complexity of merge sort is. Why do you think that?

ANS

The worst case can be that each element in an array needs to be sorted, and the best case is that the array is already sorted. Both worst-case and best-case, it has the same big-Oh complexity $O(nlogn)$.

The complexity of merge sort equals to subarray sorting time plus merge time. According to the pseudo code of merge sort, we can see that whether the element in an array is located at the right place, and the array is already sorted or not, we always halve the array into shorter pieces recursively, until there is only one element in a subarray, and then combine all the subarrays.

Therefore, the worst case and best case as a matter of fact do not affect the big-Oh complexity of merge sort.

Assuming that dividing the array and combing the array take in total cn time, and the total time quired by merge sort is $T(n)$. For each division, we are actually halving the array, and for each smaller array, it needs $\frac{1}{2}T(\frac{n}{2})$ time.Then we have

$$T(n) = 2T\left(\frac{n}{2}\right) + cn$$

$$= 2(2T\left(\frac{n}{4}\right) + \frac{cn}{2}) + cn$$

$$= 2\left[2\left(2T\left(\frac{n}{8}\right) + \frac{cn}{4}\right) + \frac{cn}{2}\right] + cn$$

$$\ldots \quad = 2^k T\left(\frac{n}{2^k}\right) + kcn$$

Let $n = 2^k, k = \ log_2 n$

$$T(n) = nT(1) + cnlog_2 n = n + cnlog_2 n$$

$$T(n) \in O(nlogn)$$

Therefore, we conclude that both the worst-case and best-case big-Oh complexity of merge sort are $O(nlogn)$.

Problem 2:

Merge sort as we have implemented in there is a recursive algorithm as this is the easiest way to think about it. It is also possible to implement merge sort iteratively:

```
// Iteratively sort subarray `A[low…high]` using a temporary array
void mergesort(int A[], int temp[], int low, int high)
{
    // divide the array into blocks of size `m`
    // m = [1, 2, 4, 8, 16…]
    for (int m = 1; m <= high - low; m = 2 * m)
    {
        // for m = 1, i = 0, 2, 4, 6, 8…
        // for m = 2, i = 0, 4, 8…
        // for m = 4, i = 0, 8…
        // …
        for (int i = low; i < high; i += 2*m)
        {
            int from = i;
            int mid = i + m - 1;
            int to = min(i + 2*m - 1, high);

            merge(A, temp, from, mid, to);
        }
    }
}
```

Explain what you think the worst-case, big-Oh complexity and the best-case, big-Oh complexity is for this iterative merge sort. Why do you think that?

ANS

Both the worst-case and best-case big-Oh complexity of iterative merge sort are $O(nlogn)$.

The analysis of iterative merge sort is quite the same as that of recursive merge sort. From the codes above, we can see that in the iterative merge sort, the array is also divided into small pieces. Then all the subarrays are combined together to obtain a sorted array.

The only difference between recursive merge sort and iterative merge sort is that the recursive merge sort starts from halving the array. While the iterative merge sort starts from the smallest pieces of the array, that is subarray only containing one element.

No matter we are applying recursion or iteration in a merge sort, the underlying logic is always divide and conquer. And for iterative merge sort, the best case and the worst case also make no difference because even if the array is already sorted, we still have to starting from the smallest pieces and combining them once and once again.

Therefore, we conclude that both the worst-case and best-case big-Oh complexity of iterative merge sort are $O(nlogn)$.