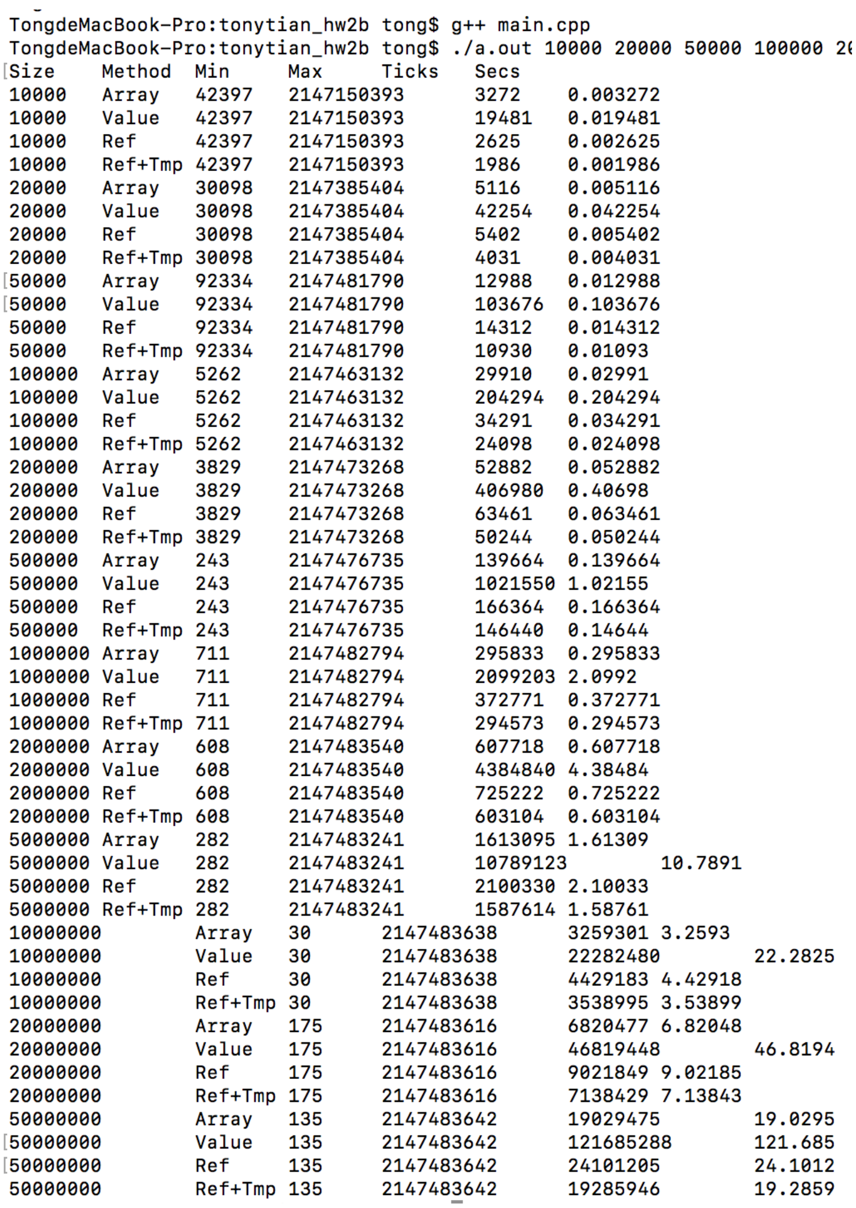

```
TongdeMacBook-Pro:tonytian_hw2b tong$ g++ main.cpp
TongdeMacBook-Pro:tonytian_hw2b tong$ ./a.out 10000 20000 50000 100000 20
Size    Method  Min     Max     Ticks   Secs
10000   Array   42397   2147150393      3272    0.003272
10000   Value   42397   2147150393      19481   0.019481
10000   Ref     42397   2147150393      2625    0.002625
10000   Ref+Tmp 42397   2147150393      1986    0.001986
20000   Array   30098   2147385404      5116    0.005116
20000   Value   30098   2147385404      42254   0.042254
20000   Ref     30098   2147385404      5402    0.005402
20000   Ref+Tmp 30098   2147385404      4031    0.004031
50000   Array   92334   2147481790      12988   0.012988
50000   Value   92334   2147481790      103676  0.103676
50000   Ref     92334   2147481790      14312   0.014312
50000   Ref+Tmp 92334   2147481790      10930   0.01093
100000  Array   5262    2147463132      29910   0.02991
100000  Value   5262    2147463132      204294  0.204294
100000  Ref     5262    2147463132      34291   0.034291
100000  Ref+Tmp 5262    2147463132      24098   0.024098
200000  Array   3829    2147473268      52882   0.052882
200000  Value   3829    2147473268      406980  0.40698
200000  Ref     3829    2147473268      63461   0.063461
200000  Ref+Tmp 3829    2147473268      50244   0.050244
500000  Array   243     2147476735      139664  0.139664
500000  Value   243     2147476735      1021550 1.02155
500000  Ref     243     2147476735      166364  0.166364
500000  Ref+Tmp 243     2147476735      146440  0.14644
1000000 Array   711     2147482794      295833  0.295833
1000000 Value   711     2147482794      2099203 2.0992
1000000 Ref     711     2147482794      372771  0.372771
1000000 Ref+Tmp 711     2147482794      294573  0.294573
2000000 Array   608     2147483540      607718  0.607718
2000000 Value   608     2147483540      4384840 4.38484
2000000 Ref     608     2147483540      725222  0.725222
2000000 Ref+Tmp 608     2147483540      603104  0.603104
5000000 Array   282     2147483241      1613095 1.61309
5000000 Value   282     2147483241      10789123        10.7891
5000000 Ref     282     2147483241      2100330 2.10033
5000000 Ref+Tmp 282     2147483241      1587614 1.58761
10000000        Array   30      2147483638      3259301 3.2593
10000000        Value   30      2147483638      22282480        22.2825
10000000        Ref     30      2147483638      4429183 4.42918
10000000        Ref+Tmp 30      2147483638      3538995 3.53899
20000000        Array   175     2147483616      6820477 6.82048
20000000        Value   175     2147483616      46819448        46.8194
20000000        Ref     175     2147483616      9021849 9.02185
20000000        Ref+Tmp 175     2147483616      7138429 7.13843
50000000        Array   135     2147483642      19029475        19.0295
50000000        Value   135     2147483642      121685288       121.685
50000000        Ref     135     2147483642      24101205        24.1012
50000000        Ref+Tmp 135     2147483642      19285946        19.2859
```

After running the four implementations of the merge sort, we can see some performance differences among them. For most of the time, the performance ranking should be:

Ref+Tmp > Array >= Ref > Value

There are mainly 3 differences on their implementations:

- The way of the outside arguments being passed into the function
- The data structure for storing the number sequence
- If additional data structures created and passed into the function

The reason has several aspects. Firstly, the Array and the Ref all have three arguments: the array structure and two indices of head and tail. The array structures are the cpp original array and the vector from STL. They are all shallow copied to be passed in, so the two functions did modifications in-place. For every depth of the recursion, there is a temporary array structure with the length n created. (assume n is the length of the original unsorted array, same below) Therefore, the only difference of array structure won't make much difference on the final performance.

The reason that Value is the slowest is that the function does hard copy for the array as argument every time the function is called, so for each depth of recursion, many arrays with n length in total would be created and copied due to the hard copy. Besides, I used "assign" to do the vector truncation, and then pass the truncated array into the function call. This operation would also do creation and copy for new array, which is also n complexity in total for each depth layer. Therefore, although is no temporary new array created for merging. The multiple times of new array creation would still be much slower of only one time of creation, whose memory is also collected immediately after using.

The reason that Ref+Tmp is the fastest is that it not only has the superiority of Array and Ref compared to Value. It doesn't have the temporary array created in each function calling for merging. It uses an additional array created previously by passing it as argument, so it saves a lot of time in sparing memory for creating new array, even if the numbers copying time complexity is still there. Therefore, Ref+Tmp slightly wins in the end.