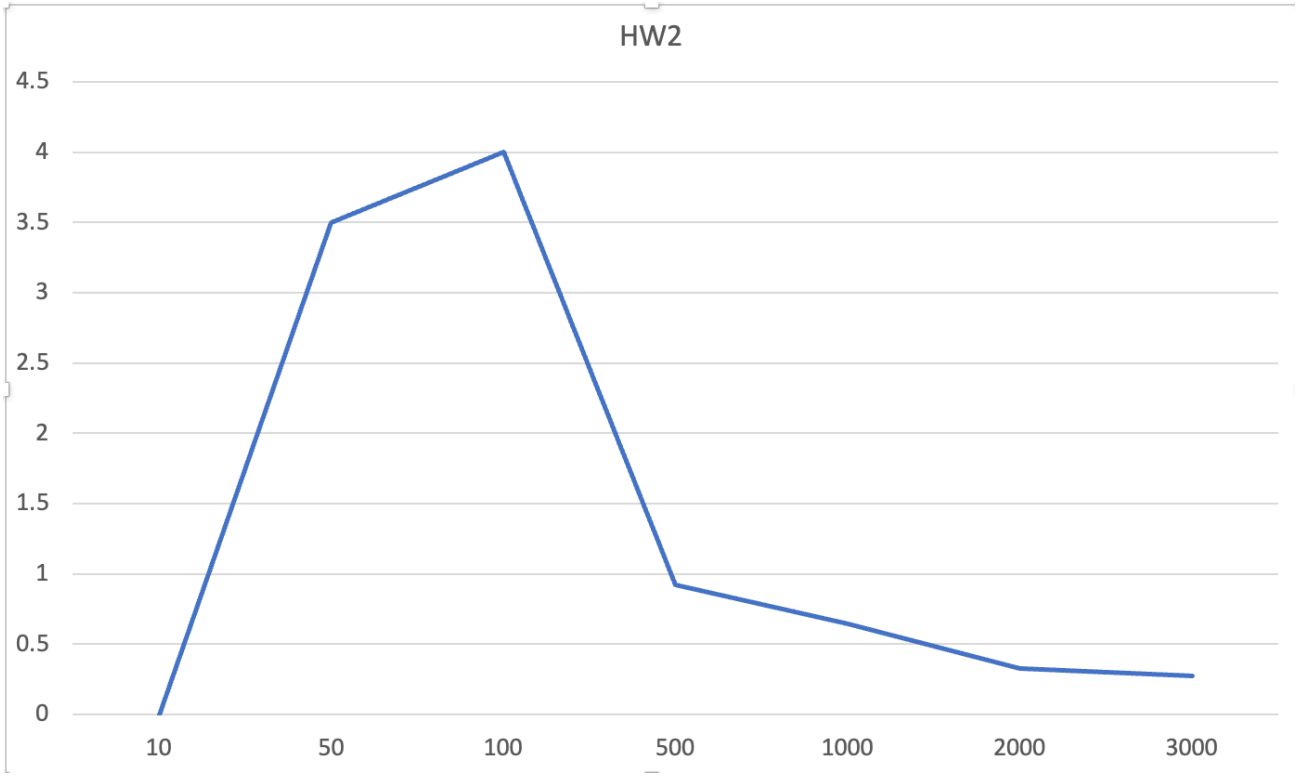
HW2

21800471 유준호

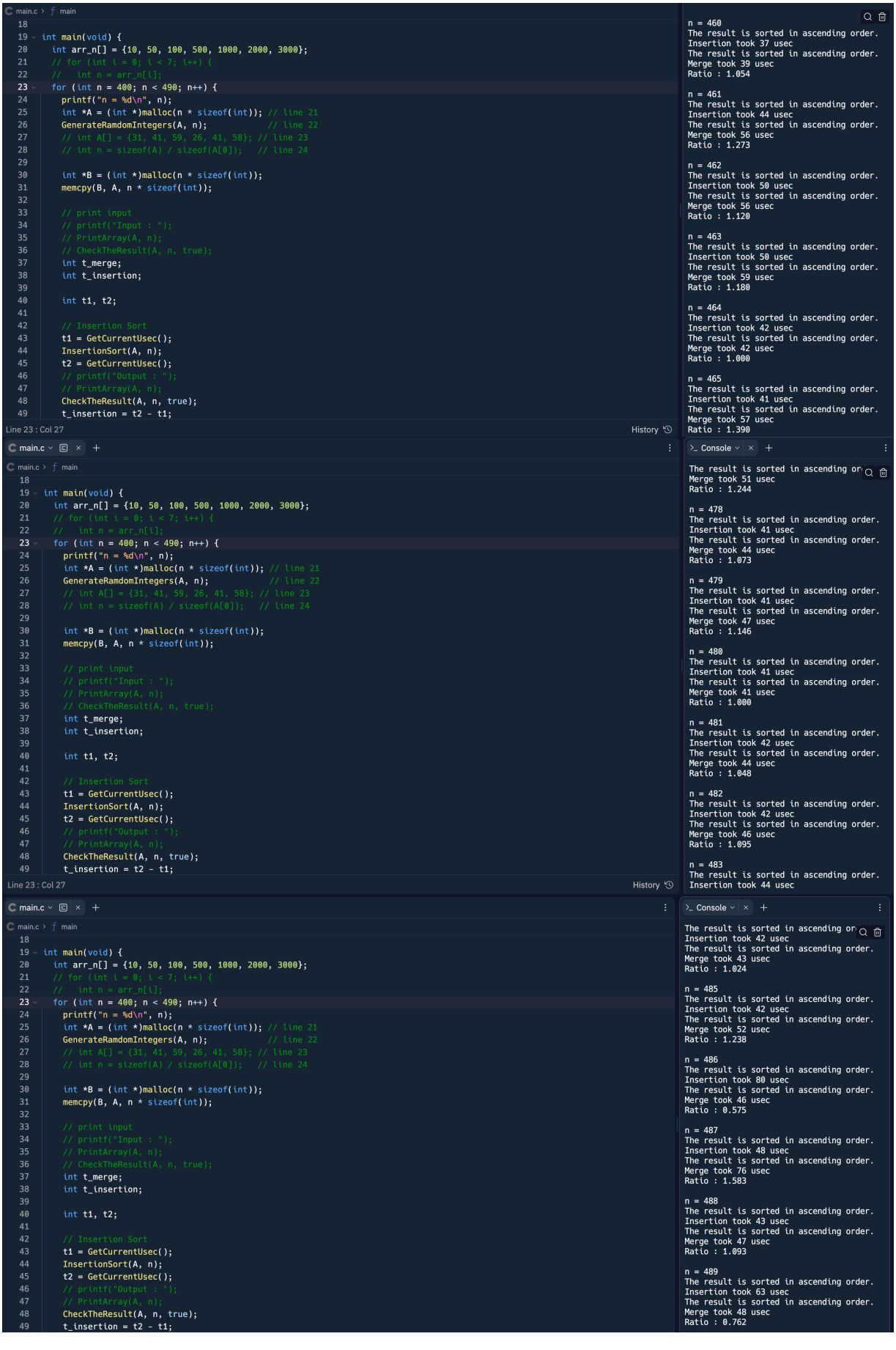
- Take the second coding assignment entitled “02 InsertionSort & MergeSort”.
    - Implement InsertionSort() function. (Use the given template code.)
    - Implement MergeSort() function. (Use the given template code.)
    - Try to understand how the Merge() function.
        - You can find the pseudo-code of the merge function in the textbook.
    - (step 4) Print the execution time (in usec) of each sorting function and print them.
    - (step 5) Calculate and print the **ratio** of the execution times of the two sorting functions. (Ex: MergeSort execution time / InsertionSort execution time)
    - Comment out the line 23 and 24
        - int A[] = {31, 41, 59, 26, 41, 58}; // line 23
        - int n = sizeof(A)/sizeof(A[0]); // line 24
    - Uncomment the line 20, 21, and 22.
        - //int n = 10; // line 20
        - //int* A = (int*)malloc(n*sizeof(int)); // line 21
        - //GenerateRamdomIntegers(A, n); // line 22
    - Set the input size "n" to 20, Repeat (step 4) and (step 5) above
    - **Submit your code to the repl.it stystem by clicking the submit button**.
    - For n = 10, 50, 100, 500, 1000, 2000, 3000, get the ratio values by doing (step 5); and make a table.
        - It would be very good if you make a graph from the table.
        - Please note that the ratio may vary from run to run even with the same n depending on the server computer's condition.
        - Try running your code at least three times and calculate the average value.
    - Find n (approximately) where the insertion sort gets slower than the merge sort
        - i.e. where the ratio becomes around 1.
        - Insert this result to the table.
    - **Write a report** and include the table you created.
        - **Explain why insertion sort takes longer time than merge sort does when the input size is large.**
    - **Submit your report to the [Assignments] menu(here) on HDLMS**.
        - PDF file format is preferred.
- FYI
    - If you want to check if an array is sorted in ascending (or descending) order, use CheckTheResult function.
    - If you want to print an array on the console window, use PrintArray function.

[Table]

N	Insertion time(us)	Merge time(us)	Ratio(t_merge/t_insertion)
10	0	1	infinity
50	2	7	3.500
100	3	12	4.000
500	53	49	0.925
1000	178	115	0.646
2000	815	265	0.325
3000	1597	434	0.272



Ratio approximately becomes 1 when the n is between 470 and 490. This is my example.



```c
int main(void) {
  int arr_n[] = {10, 50, 100, 500, 1000, 2000, 3000};
  // for (int i = 0; i < 7; i++) {
  //   int n = arr_n[i];
  for (int n = 400; n < 490; n++) {
    printf("n = %d\n", n);
    int *A = (int *)malloc(n * sizeof(int)); // line 21
    GenerateRamdomIntegers(A, n);            // line 22
    // int A[] = {31, 41, 59, 26, 41, 58}; // line 23
    // int n = sizeof(A) / sizeof(A[0]);   // line 24

    int *B = (int *)malloc(n * sizeof(int));
    memcpy(B, A, n * sizeof(int));

    // print input
    // printf("Input : ");
    // PrintArray(A, n);
    // CheckTheResult(A, n, true);
    int t_merge;
    int t_insertion;

    int t1, t2;

    // Insertion Sort
    t1 = GetCurrentUsec();
    InsertionSort(A, n);
    t2 = GetCurrentUsec();
    // printf("Output : ");
    // PrintArray(A, n);
    CheckTheResult(A, n, true);
    t_insertion = t2 - t1;
```

```
n = 460
The result is sorted in ascending order.
Insertion took 37 usec
The result is sorted in ascending order.
Merge took 39 usec
Ratio : 1.054

n = 461
The result is sorted in ascending order.
Insertion took 44 usec
The result is sorted in ascending order.
Merge took 56 usec
Ratio : 1.273

n = 462
The result is sorted in ascending order.
Insertion took 50 usec
The result is sorted in ascending order.
Merge took 56 usec
Ratio : 1.120

n = 463
The result is sorted in ascending order.
Insertion took 50 usec
The result is sorted in ascending order.
Merge took 59 usec
Ratio : 1.180

n = 464
The result is sorted in ascending order.
Insertion took 42 usec
The result is sorted in ascending order.
Merge took 42 usec
Ratio : 1.000

n = 465
The result is sorted in ascending order.
Insertion took 41 usec
The result is sorted in ascending order.
Merge took 57 usec
Ratio : 1.390
```

```
The result is sorted in ascending or
Merge took 51 usec
Ratio : 1.244

n = 478
The result is sorted in ascending order.
Insertion took 41 usec
The result is sorted in ascending order.
Merge took 44 usec
Ratio : 1.073

n = 479
The result is sorted in ascending order.
Insertion took 41 usec
The result is sorted in ascending order.
Merge took 47 usec
Ratio : 1.146

n = 480
The result is sorted in ascending order.
Insertion took 41 usec
The result is sorted in ascending order.
Merge took 41 usec
Ratio : 1.000

n = 481
The result is sorted in ascending order.
Insertion took 42 usec
The result is sorted in ascending order.
Merge took 44 usec
Ratio : 1.048

n = 482
The result is sorted in ascending order.
Insertion took 42 usec
The result is sorted in ascending order.
Merge took 46 usec
Ratio : 1.095

n = 483
The result is sorted in ascending order.
Insertion took 44 usec
```

```
The result is sorted in ascending or
Insertion took 42 usec
The result is sorted in ascending order.
Merge took 43 usec
Ratio : 1.024

n = 485
The result is sorted in ascending order.
Insertion took 42 usec
The result is sorted in ascending order.
Merge took 52 usec
Ratio : 1.238

n = 486
The result is sorted in ascending order.
Insertion took 80 usec
The result is sorted in ascending order.
Merge took 46 usec
Ratio : 0.575

n = 487
The result is sorted in ascending order.
Insertion took 48 usec
The result is sorted in ascending order.
Merge took 76 usec
Ratio : 1.583

n = 488
The result is sorted in ascending order.
Insertion took 43 usec
The result is sorted in ascending order.
Merge took 47 usec
Ratio : 1.093

n = 489
The result is sorted in ascending order.
Insertion took 63 usec
The result is sorted in ascending order.
Merge took 48 usec
Ratio : 0.762
```

Q. why insertion sort takes longer time than merge sort does when the input size is large?

It is because of the difference in their time complexity. Insertion sort has a time complexity of O(N^2). Merge sort has a time complexity of O(NlogN). This difference causes a performance difference according to the input size. When the input size is large, Insertion sort requires a large number of comparisons and swaps, resulting in a significant amount of time spent on these operations. In contrast, Merge sort divides the list into smaller sub-lists, sorts each sub-list separately, and then merges them back together. This approach reduces the number of comparisons and swaps required, resulting in a faster sorting time.