Exercise – Sort

1. Using the pseudocode for the Insertion Sort algorithm presented in the lecture, illustrate the operation of insertion sort on the array A = {31, 41, 59, 26, 41, 58}

   Draw a diagram similar to the one presented alongside the pseudocode

2. Using the pseudocode presented in the lecture, implement the Bubble Sort and Insertion Sort algorithms

   The following code segment demonstrates how you can profile the execution time of a program.

```
#include <iostream>
#include <chrono>

using namespace std;
using namespace std::chrono;

void function()
{
    long long number = 0;

    for( long long i = 0; i != 2000000; ++i )
    {
       number += 5;
    }
}

int main()
{
    high_resolution_clock::time_point t1 = high_resolution_clock::now();
    function();
    high_resolution_clock::time_point t2 = high_resolution_clock::now();

    auto duration = std::chrono::duration_cast<std::chrono::microseconds>(
              t2 - t1 ).count();

    cout << duration;
    return 0;
}
```

   Record the execution time of both your Bubble Sort and Insertion Sort algorithms and compare the result.

Challenge:

1. Implement and profile the Merge Sort algorithm

2. Although merge sort runs in O(n log n) worst-case time and insertion sort runs in $O(n^2)$ worst-case time, the constant factors in insertion sort make it faster for small *n*.

   Thus, it makes sense to use insertion sort within merge sort when sub-problems become sufficiently small.

   Modify your Merge Sort implementation so that *n / k* sub-lists of length *k* are sorted using insertion sort and then merged using the standard merging mechanism, where *k* is a value to be determined