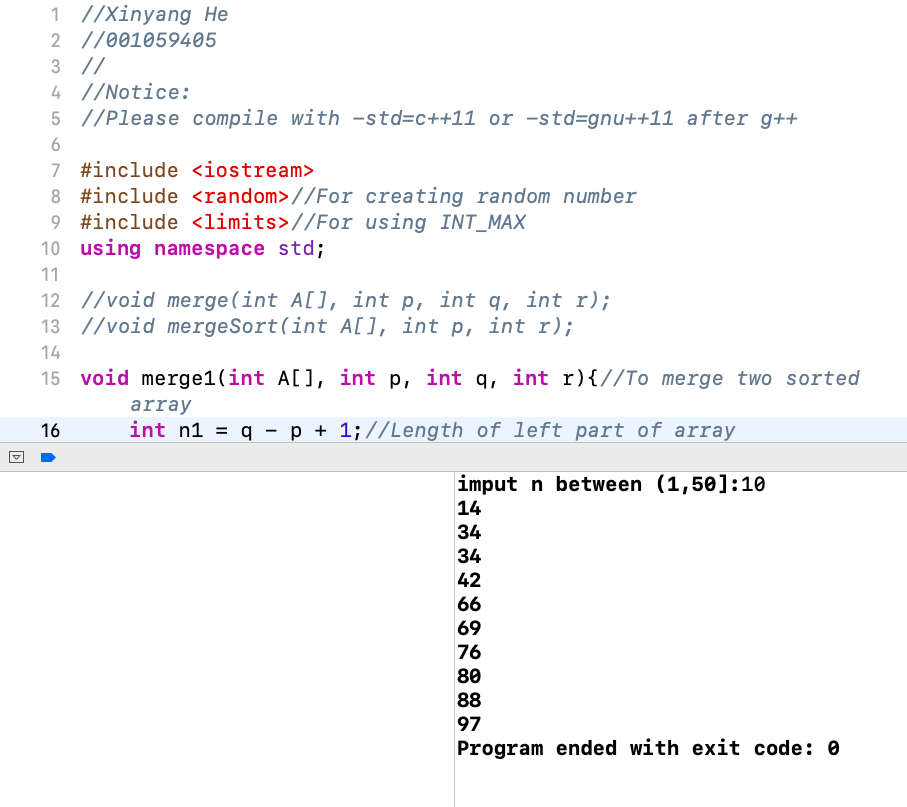
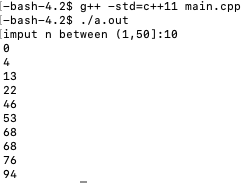
Problem 1

In Xcode:



In COE Linux:



Problem 2

The following code could be tested by call merge2() instead of merge1() at ‘main.cpp’.

**void** merge2(**int** A[], **int** p, **int** q, **int** r){//To merge two sorted array

**int** n1 = q - p + 1;//Length of left part of array

**int** n2 = r - q;//Length of right part of array

**int** \*L = **new** **int**[n1];//Empty left sorted array

**int** \*R = **new** **int**[n2];//Empty right sorted array

**for**(**int** i = 0; i < n1; i++){//Fill the value of L[] from A[]'s left part

L[i] = A[p + i];

}

**for**(**int** j = 0; j < n2; j++){//Fill the value of R[] from A[]'s right part

R[j] = A[q + j + 1];

}

**int** i = 0;//Set the initial value of loop

**int** j = 0;

**int** k = p;

**while**(i < n1 && j < n2){//Make sure L[i] and R[j] have value

**if**(L[i] <= R[j])

A[k++] = L[i++];//Fill A[] with smaller integer

**else**

A[k++] = R[j++];

}

**while**(i < n1)//After comparing, fill A[] with remaining integer

A[k++] = L[i++];

**while**(j < n2)

A[k++] = R[j++];

**delete**[] L;//Release dynamic array

**delete**[] R;

}

Problem 3

a)

It is called Bubble Sort. It repeatedly visits unsorted arrays, compares two adjacent elements in turn from the last one to front, swaps them while former one is larger than later one. After one step, the first element at current array will be smallest. Then repeat step without sorted part until no adjacent elements need to be exchanged, and the element columns have been sorted. (65 words)

b)

Solution 1:

At first step, the times of comparing is n-1, and at second step, the times of comparing is n-2…Until the last step, the time of comparing is one.

So the worst time formula is

T(n) = (n - 1) + (n – 2) + … + 1 = n \* (n -1) / 2 =

So the time complexity is O(n^2)

Solution 2:

|  |  |  |
| --- | --- | --- |
| Step | Cost | Times |
| 1 | C1 | n |
| 2 | C2 |  |
| 3 | C3 |  |
| 4 | C4 |  |

If this array is sorted from large to small, it will be the worst case. For each i = 1, 2, …, n-1, we find that = i for i = 1, 2, …, n - 1.

The worst-case running time is the following quadratic function of n:

T(n) = c1 \* n + c2 \* + c3 \* + c4 \*

= c1 \* n + c2 \* n\*(n – 1) / 2 +c3 \* (n – 2) \* (n – 1) / 2 + c4 \* (n – 2) \* (n – 1) / 2

=

= for constant a, b and c

So the time complexity is O(n^2)

Problem 4

Recurrence equation:

Solve:

W(n) = W(n-1) + (n – 1)

= [W(n – 2) + (n – 2)] + (n – 1)

= W(n – 2) + (n – 2) + (n – 1)

= …

= W(1) + 1 + 2 + … + (n – 2) + (n – 1)

= 0 + 1 + 2 + … + (n – 2) + (n – 1)

= n \* (n – 1) / 2

=

So time complexity is O(n^2)