Ex1.

(i)

We have already known that for the normal mergesort of n numbers, the time complexity is:

And for the k-mergesort of n numbers, assume that the time complexity is , the complexity of comparison is a constant m. Other operation and comparison need time complexity d .We have the recurrence equation:

Finally we can solve this recurrence equation that:

Therefore, we conclude that the time complexity , which is the same as the normal mergesort.

(ii)

Assume comparison operation needs time complexity c, and the time complexity of assignment is d.

The time complexity of bubble sort is:

The time complexity of mergesort is:

And we can get the formula:

So, we can get the answer for

We know that , so we can get ,

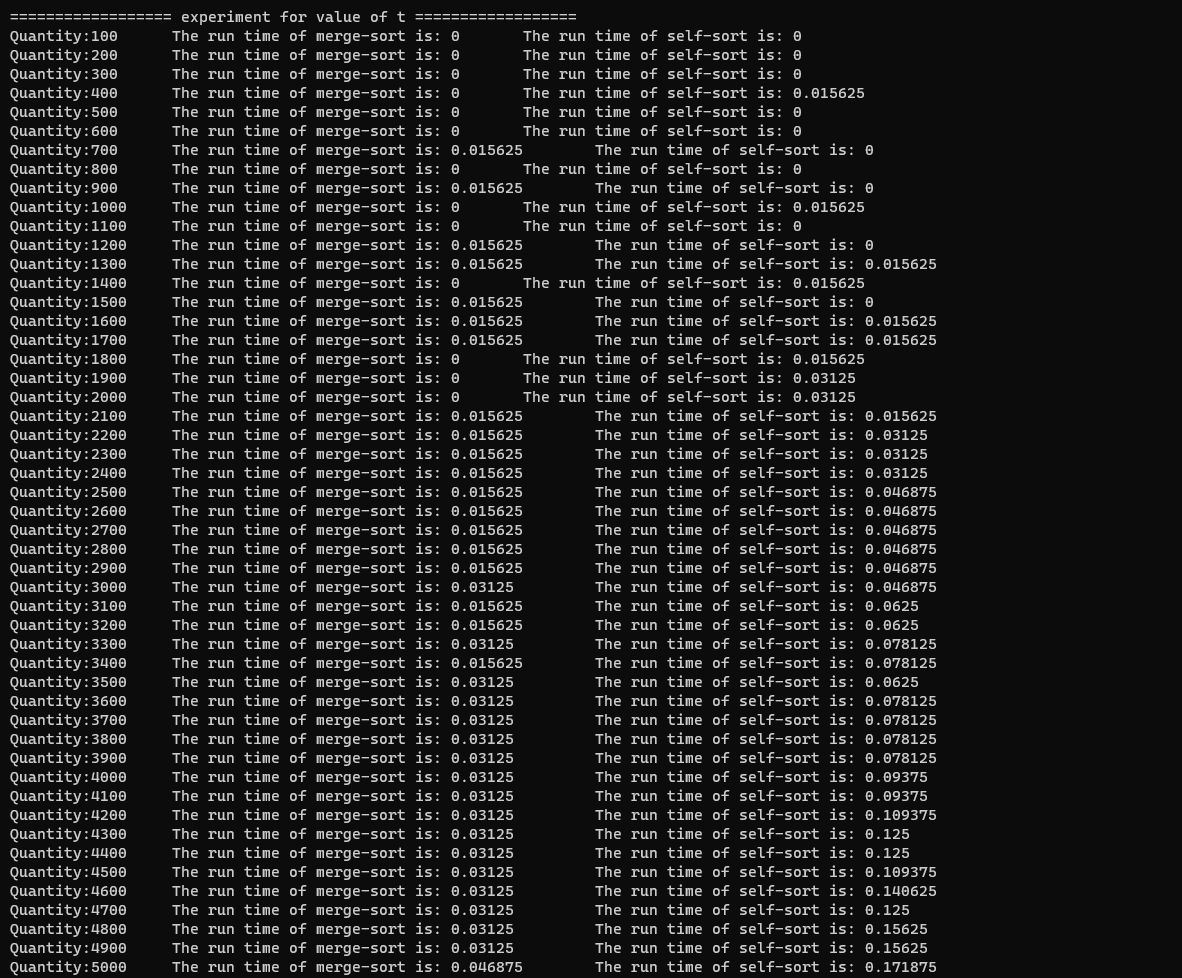
Therefore,

So, we need to solve the equation for n. we will get the equation that

We solve that , since 1 can not be the number of array, so we solve the theorical solution t=6.31972, approximately equal to 6.

From our experiment, which is shown in the output of program, we can see that the experiment t=2100. (it may be influenced by the performance of different computer and the random array)

Experiment output:



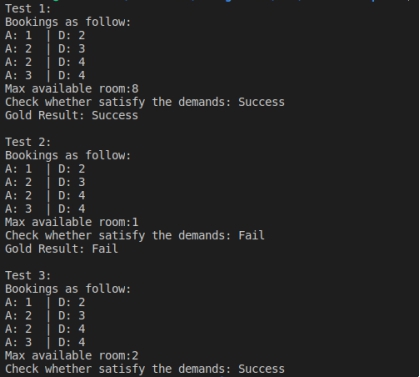
Ex2.

1. First, we put the data into two lists(vector), arrival date and the departure date. And use the merge sort to sort the list from small to big. And use another number “num” to track the number of rooms which are used. And its original value is 0.

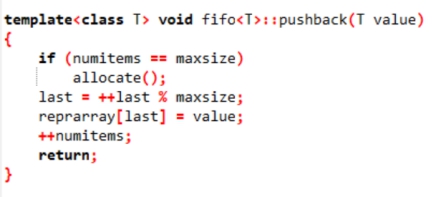
And compare the two first element, if the arrival date is smaller than the departure date, make the arrival date’s index plus one and “num” plus one, which means a customer is in. If not, make the departure date plus one and “number” minus one, which means a customer is out. We do this until the arrival date list to the end(success) or the “number” is greater than the “maximum rooms”(fail)

Then we can analyze the time complexity. The process of creating the lists has the time O(n). And the merge sort has time O(nlogn). The comparison of the two lists has time O(n). So in total, its time is O(nlogn).

1. The three test results are here and the code is in the attach file.



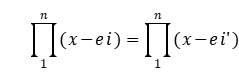
Ex3.

We can build two stacks, the first one, A, is to store queue in the right order, which is used for pushback. While the other one, B, is storing the queue in the inverted order, which is used for popfront. When you push element(s) in stack A, just like normal one (figure 1) , the amortised time is (1). And When you pop element(s) in stack B, the first situation is not empty, just pop out, just like normal one, the amortised time is  (1). The second situation, stack B is empty, we need copy. And for copy elements from A to B, we need copy n times, which can share with n element and amortised time is  (1).

Above all, I think the amortised time is  (1).

Ex.4

1. When [e­1, …, eni] is a permutation of [e1’, …, eni’]. They can be sorted as the same list. Therefore, we can get the equation as follows:



P(x) is always 0.

1. A non-zero polynomial of degree n has **at most** n roots. If we choose from p values, the probability of choosing the root is smaller than 

Given that  , we get , which indicates the result of

the evaluation is zero with probability at most .