A. Analyzing problem

Design and analyze binary search algorithms for sorted integer arrays.

1. Formally define the problem by specifying valid inputs and outputs, as well as their relationship.

Answer: **Binary search**

Input: A sorted integer array A with n elements $\langle A_1, A_2, A_3, \ldots, A_{n-1}, A_n \rangle$, where $A_1 \leq A_2 \leq A_3 \leq \ldots \leq A_{n-1} \leq A_n$, a target value T.

Output: The index of T. or returen -1.

2. In the pseudocode format defined in the textbook, write two versions of binary search, one recursive and the other non-recursive.

Algorithm 1 RECURSIVE-BINARY-SEARCH

```
1:  procedure RECURSIVE-BINARY-SEARCH(A[L, ..., H], T)
2:      if L > H then
3:          return NIL
4:      end if
5:      m = (L + H)/2
6:      if T == A[m] then
7:          return m
8:      else if T > A[m] then
9:          return RECURSIVE-BINARY-SEARCH(A[m + 1, ..., H], T)
10:     else
11:         return RECURSIVE-BINARY-SEARCH(A[L, ..., m − 1], T)
```

Algorithm 2 NON-RECURSIVE-BINARY-SEARCH

```
1:  procedure NON-RECURSIVE-BINARY-SEARCH(A, T)
2:      Let L = 1
3:      Let H = A.length
4:      while L < H do
5:          m = (L + H)/2
6:          if T == A[m] then
7:              return m
8:          else if T > A[m] then
9:              L = m + 1
10:         else
11:             H = m − 1
        return NIL
```

3.Prove the correctness of the non-recursive algorithm using loop invariant.

Answer:

Initialization: **A** is a sorted integer array and T $\in$ **A**, therefore $A[1] \leqslant T \leqslant A[A.length]$ establishes. Since $L = 1, H = A.length, A.length \geqslant 1$, then $L \leqslant H, A[L] \leqslant T \leqslant A[H]$ establishes.

Maintenance: In the following loop, the sorted integer array will not change, m is the average of L and H, so $L \leqslant m \leqslant H$ establishes. In each loop, L or H will change, then define L' and H' as the values of L and H in the end of the loop. Either $T \in A[L, m]$ or $T \in A[m + 1, H]$, so if $T \in A[L, m]$, then $T \leqslant A[m], L' = L, H' = m$, the $T \in A[L', H']$ establishes. The same reasoning leads to this establishment when $T \in A[m + 1, H$.

Termination: Since $L = 1, H = A.length, A.length \geqslant 1$, then $L \leqslant H$. If $L < H$, the integer division rounds down, it becomes smaller on every loop iteration. So the loop finally terminates. If $L == H$, then the loop terminates and return NIL. So both conditions will terminate.

4.Guess the time function T(n) of the recursive algorithm using recursion tree.

The recurrence relation $T(n) = T(\frac{n}{2}) + C$

Step 1: $T(n) = T(\frac{n}{2}) + C$

Step 2: $T(\frac{n}{2}) = T(\frac{n}{2^2}) + C$

Step 3: $T(\frac{n}{2^2}) = T(\frac{n}{2^3}) + C$

.

.

.

Step k: $T(\frac{n}{2^{k-1}}) = T(\frac{n}{2^k}) + C$

Add all the step equations, $T(n) = T(\frac{n}{2^k}) + kC$

Since $\frac{n}{2^k} = 1$, $n = 2^k$, i.e., $logn = k$

Then $T(n) = T(1) + logn$, i.e., So $T(n) = O(logn)$

5.Prove the above guess using substitution method.

Since $T(n) = O(logn)$, $T(n) = T(\frac{n}{2}) + n$

$$\begin{aligned} T(n) &\leqslant c\log(\frac{n}{2}) + n \\ &\leqslant c(\log n - \log 2) + n \\ &\leqslant c\log n - c + n \\ &\leqslant c\log n \end{aligned}$$

6.Confirm the result using the master method.

Because $T(n) = T(\frac{n}{2}) + c$, then $T(n) = T(\frac{n}{2}) + f(n)$.

According to Master theorem:

$$\begin{aligned} T(n) &= \Theta(\log n * n^{\log_2^1}) \\ &= \Theta(\log n) \\ &= O(n) \end{aligned}$$

B. Programming problem

Code the Insertion Sort and Merge Sort algorithms given in the textbook and compare their time complexity via testing.

1. Code both algorithms in the same program and try to be as similar to the pseudocode in the textbook as possible.

Answer: The code is in the uploaded cpp file with the input/output screen captures.

2. Add proper indicators of time cost into the program (such as counters for certain operations) — you need to explain, in the report, why you think these indicators are proper for the job.

Answer: The comparison and assignment counters are added in the code ,which used as the indicators of time cost. Since comparison and assignment are the most cost operations in the calculation process.

3.Decide the size and number of testing cases, so that there are enough data to show the difference between the two algorithms on time complexity, and at the same time the test does not run too long.

Answer:In the test session, in order to show the difference between the two algorithms on time complexity, and the length of the array to be sorted in each set of test data can be set. Arrays of length 5, 10, 50, 100, 500, 1000, 5000 are tested respectively. Each length of the array is generated in ten sets, and each number in the array is randomly generated in the range of [0,100], and the result is obtained by calculating the average of the 10 sets of data.

When the length of random array is 5000:

```
ave_merge_sort_counter: 302385
ave_insert_sort_counter: 12354150
```

It is obvious that the time cost difference between the two algorithms.

4.Use random number generator to generate testing cases and use them on both algorithms, and use sample runs to show that the program works properly in the report.

Answer:

a.The sample when array length is **5**:

```
Please input the length of array:
5
Random generate array:
71 72 22 72 76
Insertion sorted array:
22 71 72 72 76
Merge sorted array:
22 71 72 72 76

total test_cases_num: 10
every_list_length: 5

ave_merge_sort_time: 5 ms
ave_merge_sort_counter: 54
ave_insert_sort_time: 1 ms
ave_insert_sort_counter: 19
```

b.The sample when array length is **10**:

```
Please input the length of array:
10
Random generate array:
7 79 22 38 46 11 63 18 59 77
Insertion sorted array:
7 11 18 22 38 46 59 63 77 79
Merge sorted array:
7 11 18 22 38 46 59 63 77 79

otal test_cases_num: 10
every_list_length: 10

ave_merge_sort_time: 7 ms
ave_merge_sort_counter: 161
ave_insert_sort_time: 2 ms
ave_insert_sort_counter: 62
```

5. Collect and display the testing results to show the difference of the two algorithms on time complexity.

Answer: Table 1 lists seven sets of test data.

Table 1: The counters of different array length

Length	Merge_Counters	Insertion_Counters
5	54	19
10	161	62
50	1370	1344
100	3237	5034
500	21806	120626
1000	48608	485402
5000	302385	12354150

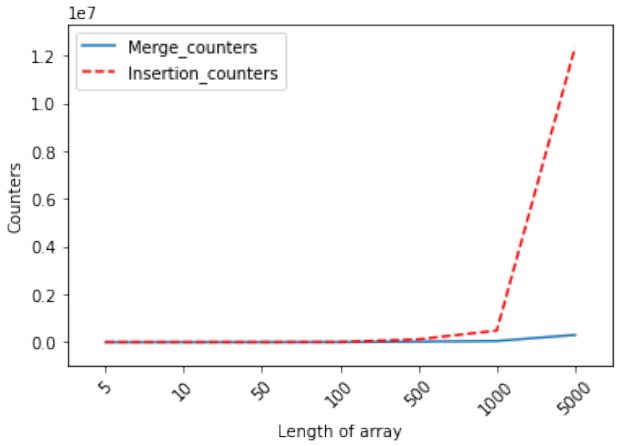

Figure 1: The plot of Length and Counters.

Figure 1 show the counters of Merge and Insertion with different array length. According to the table and figure, it is clear that when the lengths are 5, 10, 50, the insertion sort works better then merge, i.e., the insertion sort work better when the array length is short. When the array length becomes larger, its is clear that merge sort becomes better than insertion sort.

6. Compare the testing results with the theoretical analysis conclusions on the algorithms.

Answer: The time complexity of the Merge sort is $O(nlgn)$, and the time complexity of the Insertion sort is $O(n^2)$. Both of them grow with n. However, the growth of $O(nlgn)$ is slower than the growth of $O(n^2)$, i.e., as the length of array becomes larger, the time complexity of Insertion sort becomes higher and much larger than Merge.

7.(Optional)Measure the actual time (e.g., in seconds) used by the program, compare the results with the above ones, and analyze their difference.

Answer: Add the time counter in the code, and the result of the time can be found in table 2, the plot of time counter and length can be found in figure 2.

Comparing the above results, it can be found from the plot that the overall trend is the

Table 2: The time and counters of different array length

Length	Merge_Counters	Merge Time(ms)	Insertion Counters	Insertion Time(ms)
5	54	5	19	1
10	161	7	62	2
50	1370	9	1344	2
100	3237	10	5034	7
500	21806	61	120626	188
1000	48608	132	485402	747
5000	302385	868	12354150	18672

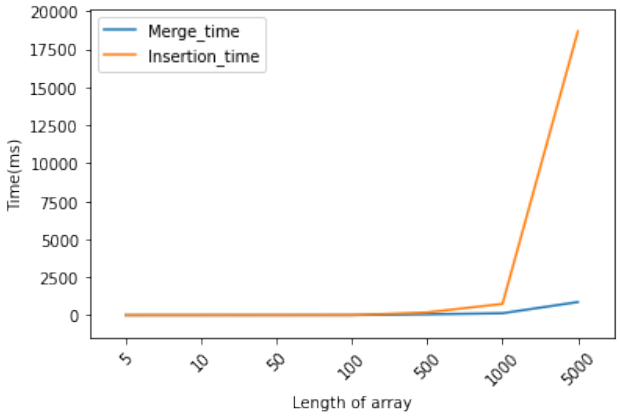

Figure 2: The plot of Length and time.

same for both, with Insertion smaller when array length is small and Merge smaller when array length is large. However, a careful observation at the two tables show that Insertion only starts to become more inefficient than Merge when array length is 500. Based on the analysis of the code, it is inferred that the calculation of time is more accurate because the calculation of counter is missing part of the assignment process. But both methods can show the overall time complexity trend of these two sort algorithms.

Reference

1. Cormen, Thomas H., et al. Introduction to algorithms. MIT press, 2009.
2. https://en.wikipedia.org/wiki/Binary$_search_algorithm$
3. https://www.youtube.com/watch?v=uEUXGcc2VXM