AOA Program Project II

1. Team Members:

Heng Zhou (UFID: 8627-8332)
Main contributions: design and analysis of 6 DP algorithms.

Zhenyu Wang (UFID: 6045-7013):
Main contributions: design and analysis of 2 Brute Force algorithms, comparative study, conclusion.

2. Design and Analysis of Algorithms

TASK 1

Algorithm Design

Design a $\Theta(m^3n^3)$ time Brute Force algorithm for solving Problem 1.

Pseudocode

```
for each plot in town
        while not reaching the boundary of the town
                expand the right-down corner of the square to find the maximal square area with all plots larger than h
return maximal square
```

Analysis

Searching every plot in the town is $\Theta(mn)$.
For every plot, expand its right down corner is $\Theta(\min(m,n))$.
Check whether every square satisfies all plots larger than h is $\Theta(mn)$.
The total time complexity is $\Theta(m^2n^2\min(m,n))$.

TASK 2

Algorithm Design

Design a $\Theta(m^2n^2)$ time algorithm for solving Problem 1.

Pseudocode

```
int[][] globalmax
for each plot i,j in town
```

```
    if p[i][j] >= h
        consider (i,j) as a start point
        find the maxlength of plots that > h
        for each plot (a,b) in maxlength find biggest square = level
        globalmax[i][j] = level
for each plot in globalmax
        return the index of biggest value
```

Analysis

Searching every plot in the town is Θ(mn).
For every plot, check that the longest edge with this point as the upper-left corner is the length of the edge of the square, and check whether every point in the square satisfies all plots larger than h is Θ(mn).
The total time complexity is $\Theta(m^2n^2)$.

TASK 3

Algorithm Design

Design a Θ(mn) time Dynamic Programming algorithm for solving Problem1

Pseudocode

```
mx = 0
if plot >= h
        plot = 1
else
        plot = 0
for each plot
        plots[i][j] = min(plots[i-1][j], plots[i][j-1], plots[i-1][j-1])+1
        mx = max(mx, plots[i][j])
return mx
```

Analysis

Recursive formulation:
for i,j = 0 to n
    plots[i][j] ← min(plots[i-1][j], plots[i][j-1], plots[i-1][j-1])+1
Search each plot for once, so the time complexity is Θ(mn).

TASK 4

Algorithm Design

Give an implementation of ALG4.

Analysis

Recursion[i][j][s] indicates the number of plots except four corners whose value is less than h in the square with (i, j)as the upper-left corner coordinate and s as the side length. After the recursion calculation, find the points with recursion[i][j][s]<k(the number of the points is denoted by a), then find the largest s among all points and return the corresponding (i, j) coordinate.

Recursive formulation:

p[i][j][s] = a + p[i][j][s-1], a is the number of plots that less than h in column(i+s) and row(j+s)

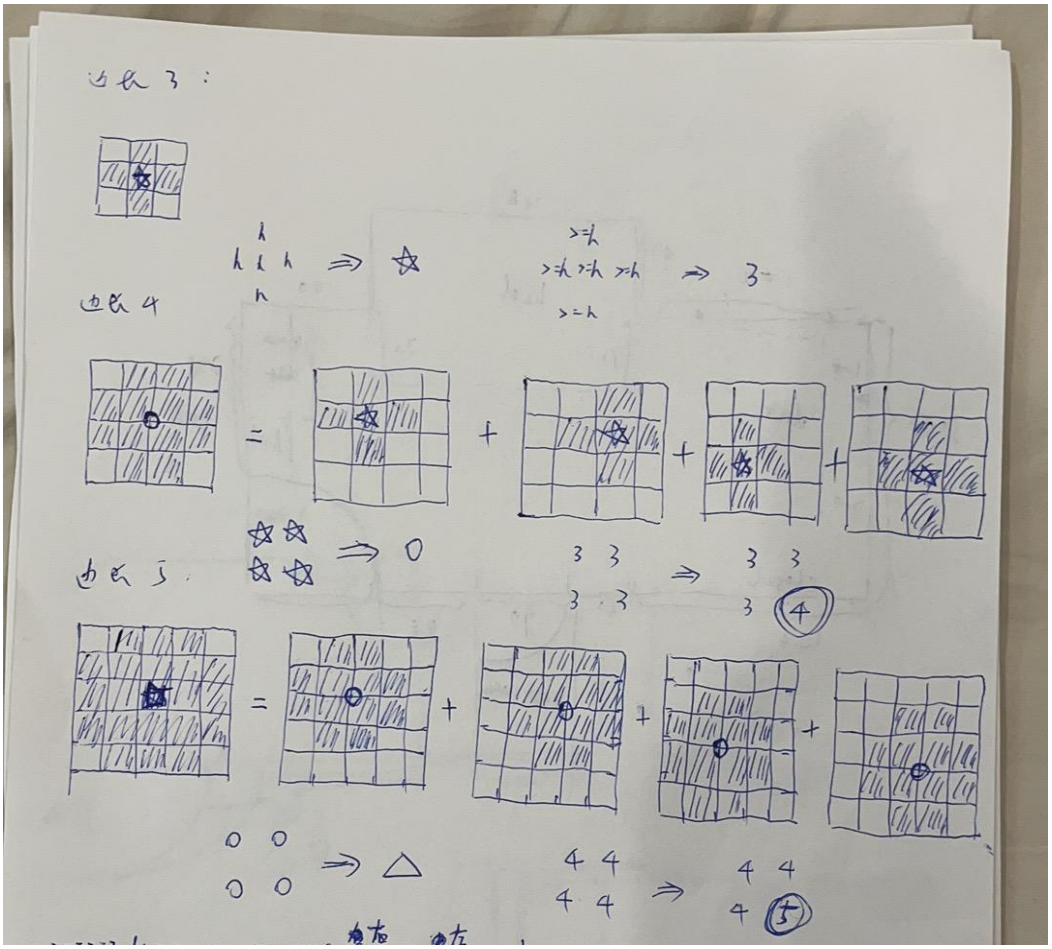
Searching every plot in the town is Θ(mn). With every plot, we have to increase s from 0 to n.So, the total time complexity is Θ(m·n^2).

TASK 5A

Algorithm Design

Give a recursive implementation of ALG5 using Memoization

Analysis



The first step is to find a plus sign with length of 3. If we got exactly four same plus signs, we can combine them into a bigger one. We need to mark all the elements to see if they can be used as the center of a plus sign. In the second step, we use recursion to calculate whether the centroid of a certain plus sign can form a larger plus sign with other plus signs.
In this process, there will be a lot of repeated calculations because recursion is the top-down method, so we need a new storage space to keep the values that have been calculated. In the process of recursion calculation, when we get a new calculation result, we save it in the array, and when we do the next calculation, we first determine whether the value to be calculated is in the memo array, if it is, then we call it directly, if not, then we calculate it.

Recursive formulation:
p[i][j] = 0 if i < 1 || j < 1
p[i][j] = 1+ min{p[i-1][j], p[i][j-1], p[i-1][j-1]} if (i >= 1 && j>=1)

Each plot is searched at one time for determine whether it is a centroid of a certain plus sign, so the complexity is Θ(mn). And time complexity of the recursion is also Θ(mn) because we have Memoization to repeated calculations.
So the overall time complexity is Θ(mn).

TASK 5B

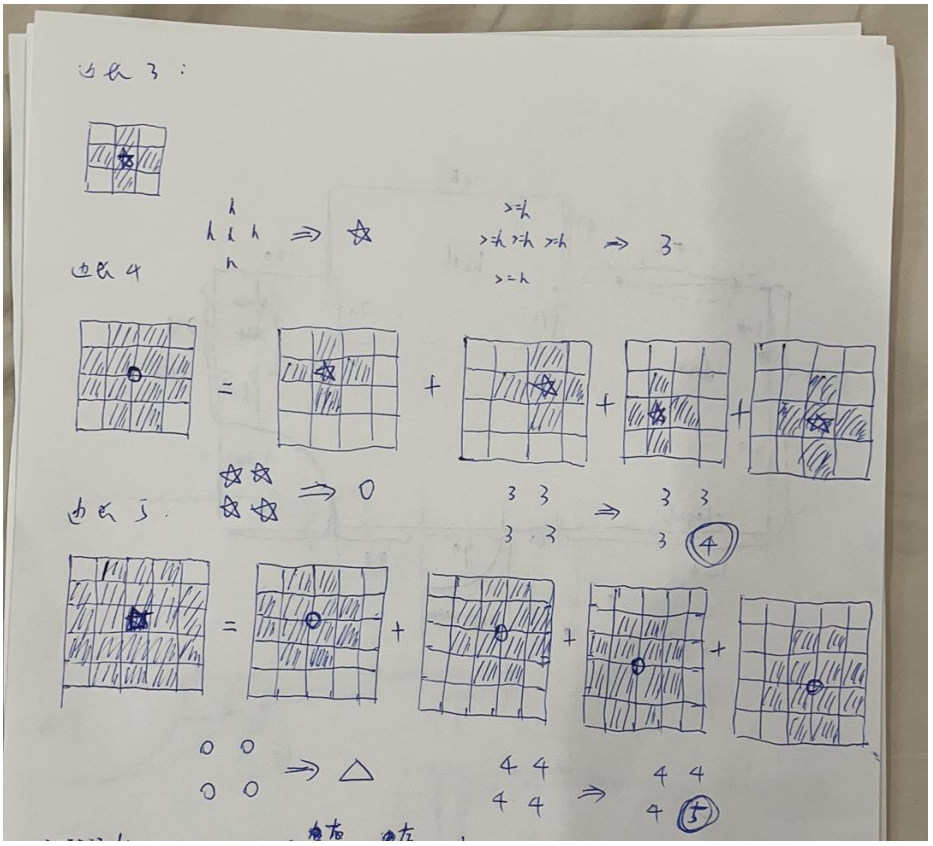
Algorithm Design

Give an iterative BottomUp implementation of Alg5.
Alg5 Design a Θ(mn) time Dynamic Programming algorithm for solving Problem2

Pseudocode

```
int[][] dp = new int[m][n];
int[][] dirs = new int[][]{{-1, 0}, {1, 0}, {0, 1}, {0, -1}};
for (int i = 1; i < m - 1; i++) {
	for (int j = 1; j < n - 1; j++) {
		boolean allNotLessThanN = true;
		for (int[] dir : dirs) {
			int ii = i + dir[0];
			int jj = j + dir[1];
			if (plots[ii][jj] < h || plots[i][j] < h) {
				allNotLessThanN = false;
			}
		}
		if (allNotLessThanN) {
			dp[i][j] = 3;
		}
	}
}
for (int i = 1; i < m; i++) {
	for (int j = 1; j < n; j++) {
		if (dp[i][j] == dp[i-1][j] && dp[i][j] == dp[i][j-1] && dp[i][j] == dp[i-1][j-1] && dp[i][j]
!= 0) {
			dp[i][j]++;
		}
		mx = Math.max(mx, dp[i][j]);
	}
}
return mx
```

Analysis



The first step is to find a plus sign with length of 3. If we got exactly four same plus signs, we can combine them into a bigger one.

Recursive formulation:

for i,j = 0 to n

plots[i][j] ← dp[i][j]++;

if (dp[i][j] == dp[i-1][j] && dp[i][j] == dp[i][j-1] && dp[i][j] == dp[i-1][j-1] && dp[i][j] != 0)

Each plot is searched at one time, so the complexity is Θ(mn).

The for loop in four directions doesn't impact time complexity, since 4 iterations are a constant.

So the overall time complexity is Θ(mn).

TASK 6

Algorithm Design

Give an implementation of Alg6.

Alg6 Design a $\Theta(m^3n^3)$ time Brute Force algorithm for solving Problem3

Pseudocode

for each plot in town

while not reaching the boundary of the town

expand the right-down corner of the square to find the maximal square area with up to k plots that are less than h
return maximal square

Analysis

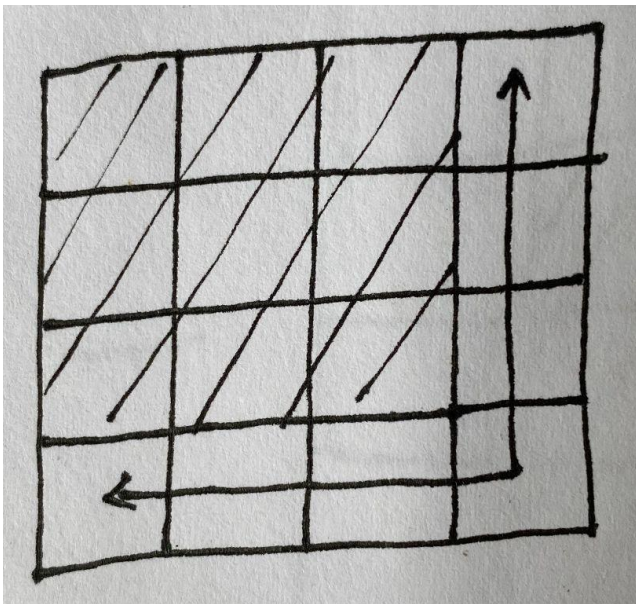
Searching every plot in the town is Θ(mn).
For every plot, expand its right down corner is Θ(min(m,n)).
Check whether every square satisfies that up to k plots are less than h is Θ(mn).
The total time complexity is $\Theta(m^2n^2\min(m,n))$.

TASK 7A

Algorithm Design

Give a recursive implementation of ALG7 using Memoization

Analysis



Recursion[i][j][s] indicates the number of plots whose value is less than h in the square with (i, j)as the upper-left corner coordinate and s as the side length. After the recursion calculation, find the points with recursion[i][j][s]<k(the number of the points is denoted by a), then find the largest s among all points and return the corresponding (i, j) coordinate.

Recursive formulation:

p[i][j][s] = a + p[i][j][s-1], a is the number of plots that less than h in column(i+s) and row(j+s)

TASK 7B

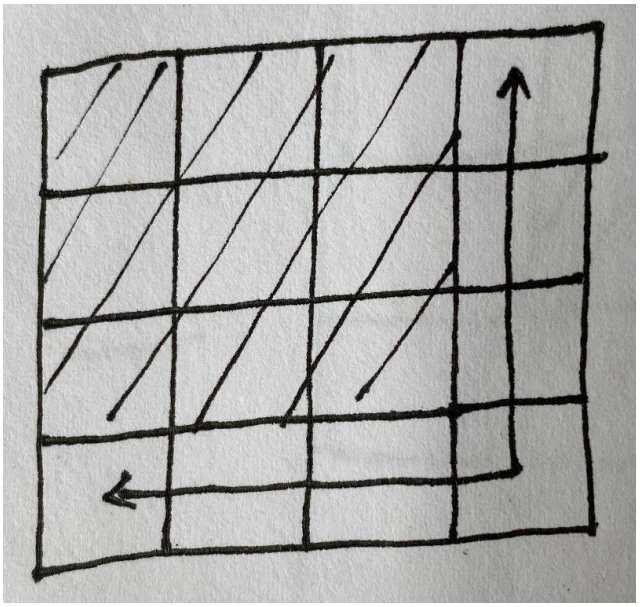
Algorithm Design

Give an iterative BottomUp implementation of Alg7.
Alg7 Design a Θ(mnk) time Dynamic Programming algorithm for solving Problem3

Pseudocode

```
for (int i = 0; i < m; i++) {
	for (int j = 0; j < n; j++) {
		int dir = 1;
		int twoDim = 0;
		if (dp[i][j] == 1) {
			history[i][j][1] = 1;
		}
		while (i - dir >= 0 && j - dir >= 0) {
			twoDim += dp[i-dir][j] + dp[i][j-dir];
			if (history[i-1][j-1][dir] + twoDim + dp[i][j] <= k) {
				history[i][j][dir+1] = history[i-1][j-1][dir] + twoDim + dp[i][j];
				dir++;
				if (dir > mx) {
					mx = dir;
					x2 = i;
					y2 = j;
					x1 = i - dir + 1;
					y1 = j - dir + 1;
				}
			}
			else
				break;
		}
	}
}
return x1 + " " + y1 + " " + x2 + " " + y2;
}
```

Analysis

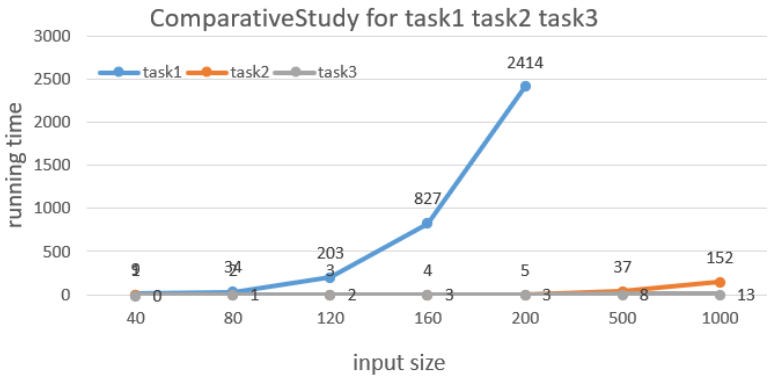


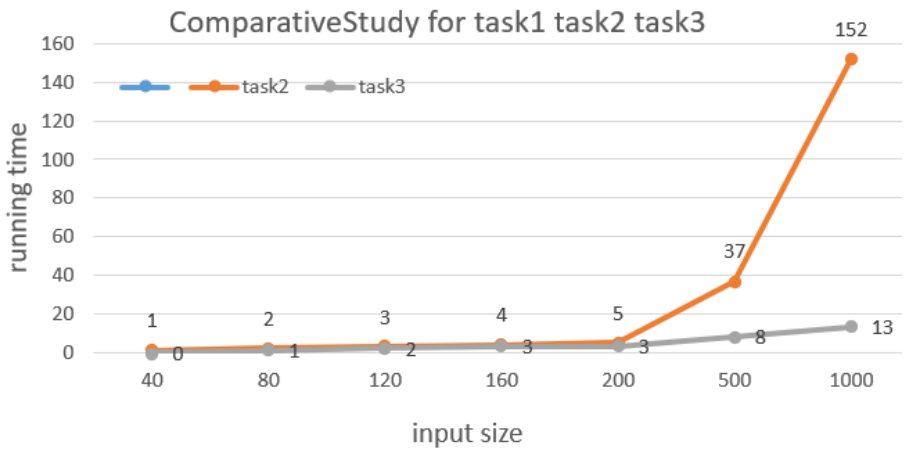
Expand the square while keeping up to k plots that are less than h, using a 3D array to remember the number of plots less than h in each plot for different square sizes.
Recursive formulation:
memo[i][j][dir+1] = memo[i-1][j-1][dir] + twoDim + dp[i][j];
Searching every plot in the town is Θ(mn).
For every plot, expand its right down corner is Θ(min(m,n)).
The total time complexity is Θ(m·n·min(m,n)).

3. Comparative Study

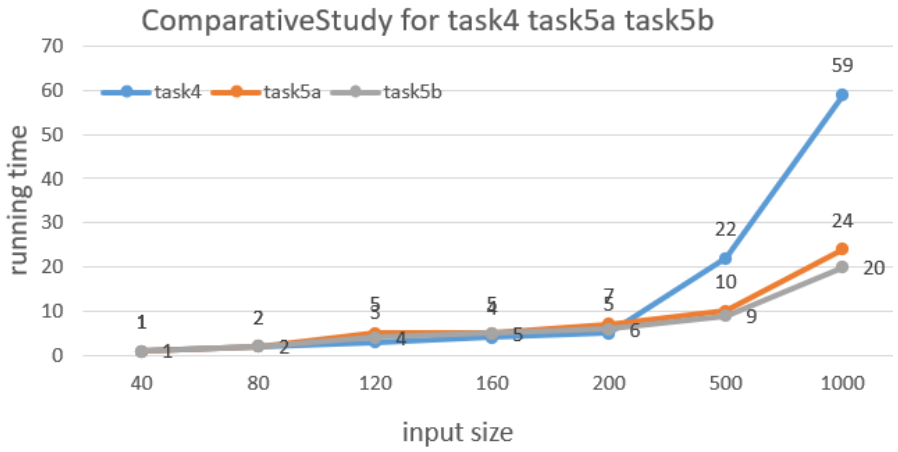
Set input size m=n=40, 80, 120, 160, 200, 500, 1000. h=50 and k=10.
Due to the explosive growth nature of prime numbers, I could not set very large input sizes in task1 and task6, which are limited by the performance of the computer, and if I set a larger inputsize, the computation time would be in hours.

Task 1 2 3

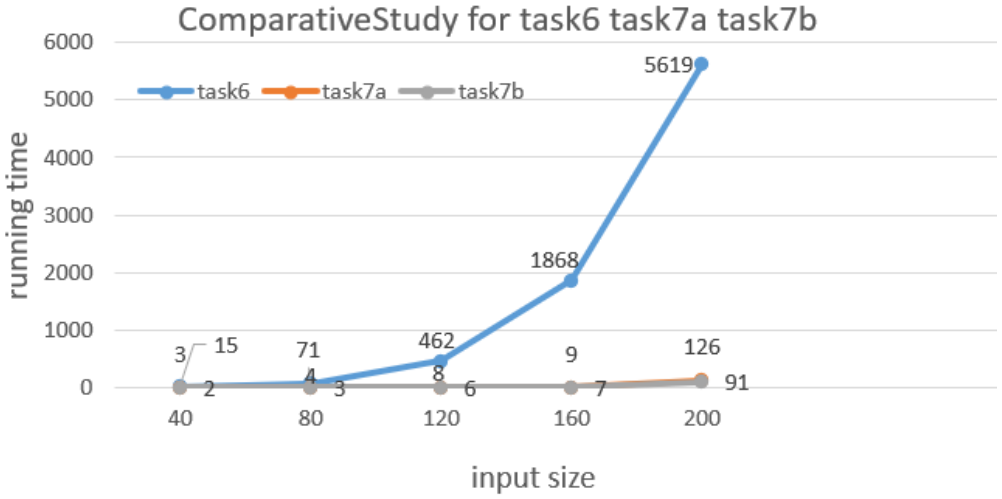


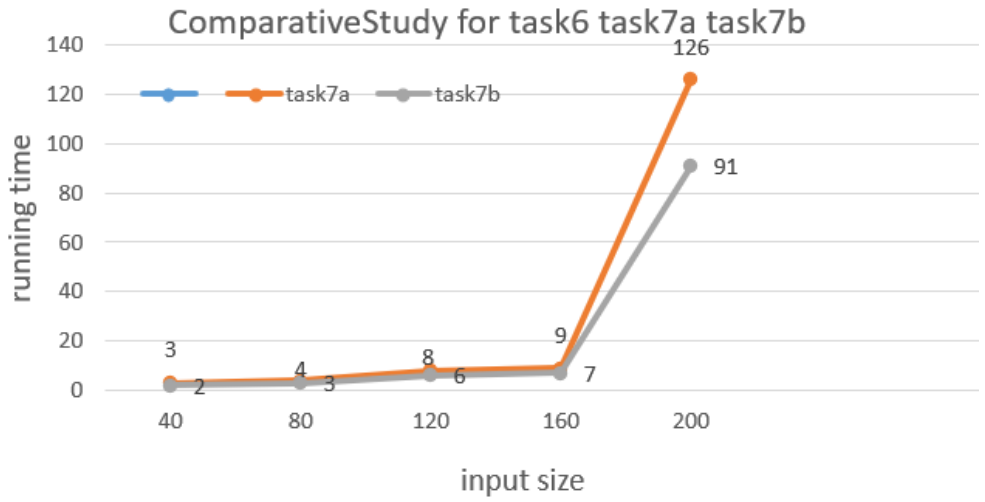


Task 4 5a 5b



Taks 6 7a 7b





4. Conclusion

It is a better way to summarize the different questions 1, 2 and 3 separately.
For question 1:
Algorithm 1, Algorithm 2, and Algorithm 3 are relatively straightforward algorithms with relatively few constraints to consider. However, in the implementation of Algorithm 2, we went through a more complex process to get the final code because there were many constraints to be taken into account to ensure that some instances were not missed without using the DP algorithm.

For question 2:
Both Algorithm 4 and Algorithm 5 use the DP approach. Interestingly, our group worked out the implementation of Algorithm 5 first, which has a much smaller time complexity. We had some difficulties in the implementation of Algorithm 4. We initially had trouble understanding why the time complexity of Algorithm 4 needed to be the square of n, because we had already written simpler ways to write it. When considering the implementation of algorithms with higher complexity, we had to consider ways of handling the data that seemed "worse."

For question 3:
In problem 2, we can use the relationship of position to solve the four corners cleverly. Unlike the first two problems, problem 3 requires more consideration of "number", which forces us to keep an eye on all the data in a square during each iteration, and whether all but k plots satisfy the h requirement.
In addition, when testing, I noticed that if the input size exceeds a certain size, say m=300 and n=300, then the exception "out of stack" is reported.

Overall, this project assignment allowed our group members to deepen our understanding of Dynamic Programming and to understand the importance of considering the time complexity of an algorithm during comparative study, which can significantly affect the efficiency of an algorithm and thus the computational resource overhead.