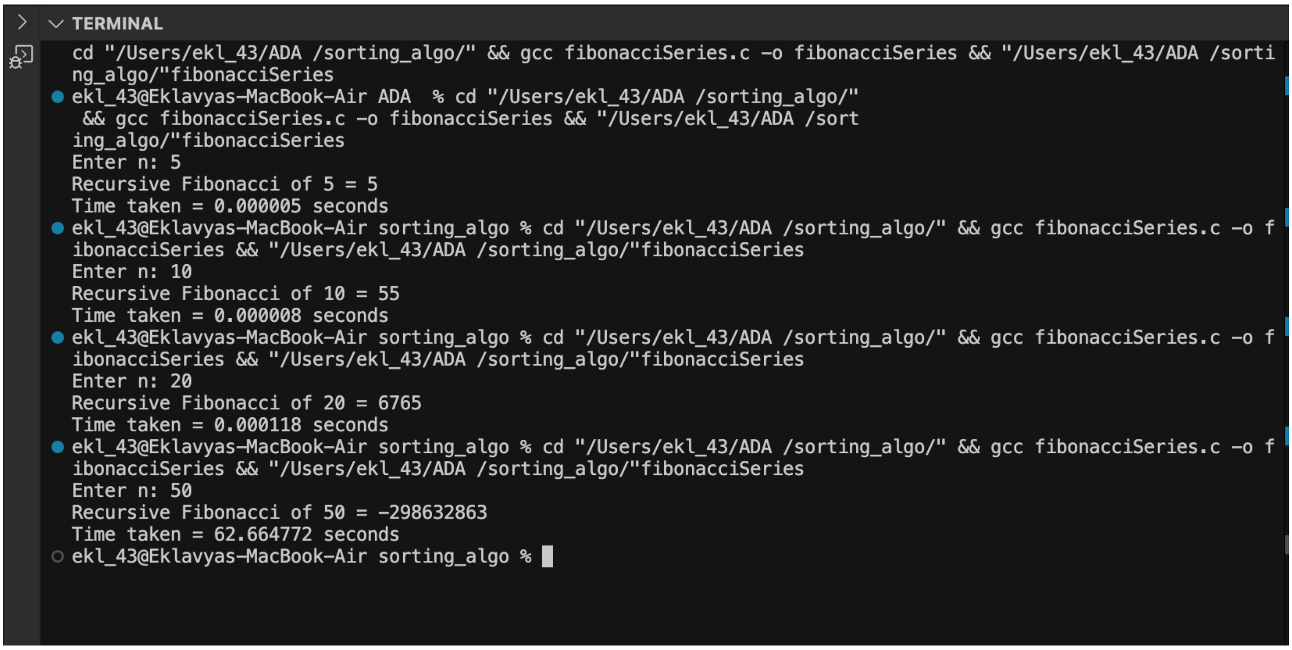
Fibonacci Series

4(a) To implement and analyze different approaches for generating Fibonacci numbers and compare their time and space complexities using **recursive version**

C CODE:

```
#include <stdio.h>
#include <time.h>
// Recursive Fibonacci
int fib_recursive(int n) {
if (n <= 1) return n;
return fib_recursive(n – 1) + fib_recursive(n – 2);
}
int main() {
int n;
printf("Enter n: ");
scanf("%d", &n);
clock_t start = clock();
int result = fib_recursive(n);
clock_t end = clock();
double time_taken = ((double)(end – start)) / CLOCKS_PER_SEC;
printf("Recursive Fibonacci of %d = %d\n", n, result);
printf("Time taken = %f seconds\n", time_taken);
return 0;
```

Output:



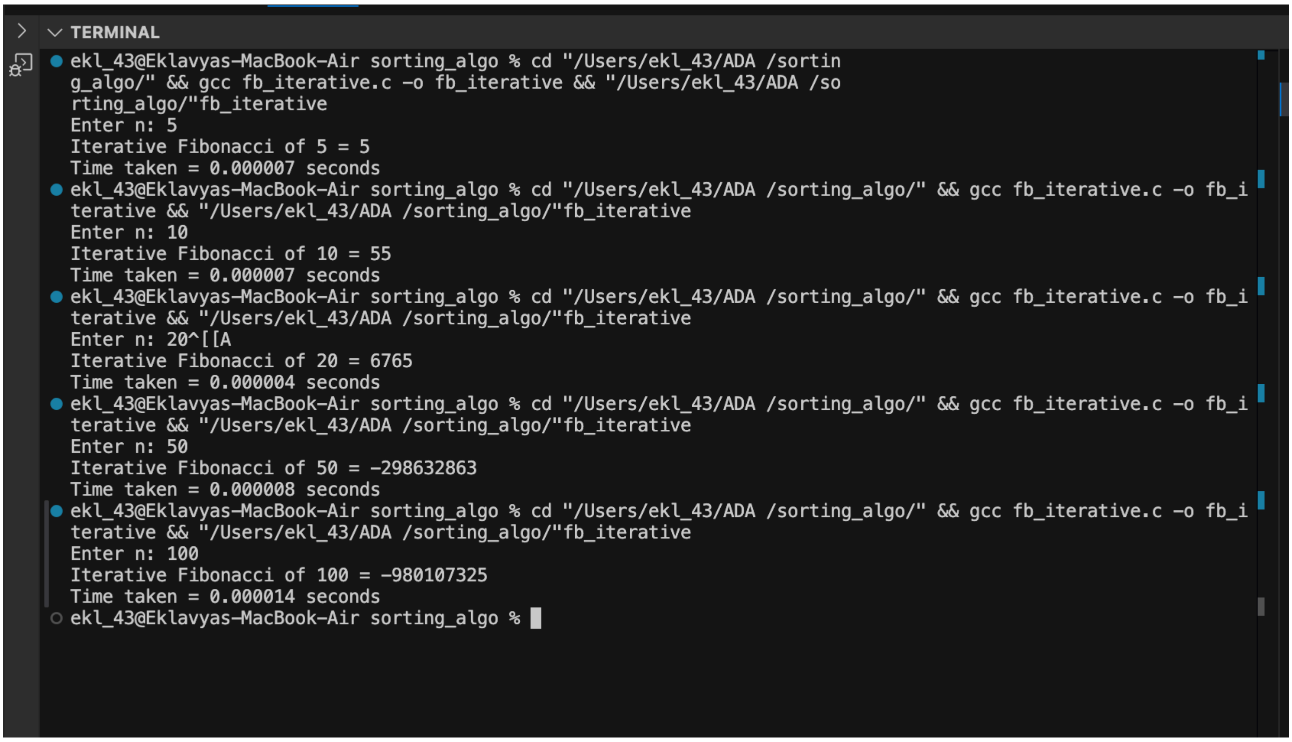
```
  cd "/Users/ekl_43/ADA /sorting_algo/" && gcc fibonacciSeries.c -o fibonacciSeries && "/Users/ekl_43/ADA /sorti
  ng_algo/"fibonacciSeries
● ekl_43@Eklavyas-MacBook-Air ADA  % cd "/Users/ekl_43/ADA /sorting_algo/"
   && gcc fibonacciSeries.c -o fibonacciSeries && "/Users/ekl_43/ADA /sort
  ing_algo/"fibonacciSeries
  Enter n: 5
  Recursive Fibonacci of 5 = 5
  Time taken = 0.000005 seconds
● ekl_43@Eklavyas-MacBook-Air sorting_algo % cd "/Users/ekl_43/ADA /sorting_algo/" && gcc fibonacciSeries.c -o f
  ibonacciSeries && "/Users/ekl_43/ADA /sorting_algo/"fibonacciSeries
  Enter n: 10
  Recursive Fibonacci of 10 = 55
  Time taken = 0.000008 seconds
● ekl_43@Eklavyas-MacBook-Air sorting_algo % cd "/Users/ekl_43/ADA /sorting_algo/" && gcc fibonacciSeries.c -o f
  ibonacciSeries && "/Users/ekl_43/ADA /sorting_algo/"fibonacciSeries
  Enter n: 20
  Recursive Fibonacci of 20 = 6765
  Time taken = 0.000118 seconds
● ekl_43@Eklavyas-MacBook-Air sorting_algo % cd "/Users/ekl_43/ADA /sorting_algo/" && gcc fibonacciSeries.c -o f
  ibonacciSeries && "/Users/ekl_43/ADA /sorting_algo/"fibonacciSeries
  Enter n: 50
  Recursive Fibonacci of 50 = -298632863
  Time taken = 62.664772 seconds
○ ekl_43@Eklavyas-MacBook-Air sorting_algo % █
```

4(b) To implement and analyze different approaches for generating Fibonacci numbers and compare their time and space complexities using **iterative version**

C CODE:

```
#include <stdio.h>
#include <time.h>
// Iterative Fibonacci
int fib_iterative(int n) {
if (n <= 1) return n;
int a = 0, b = 1, c;
for (int i = 2; i <= n; i++) {
c = a + b;
a = b;
b = c;
}
return b;
}
int main() {
int n;
printf("Enter n: ");
scanf("%d", &n);
clock_t start = clock();
int result = fib_iterative(n);
clock_t end = clock();
double time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;
printf("Iterative Fibonacci of %d = %d\n", n, result);
printf("Time taken = %f seconds\n", time_taken);
return 0;
```

Output:



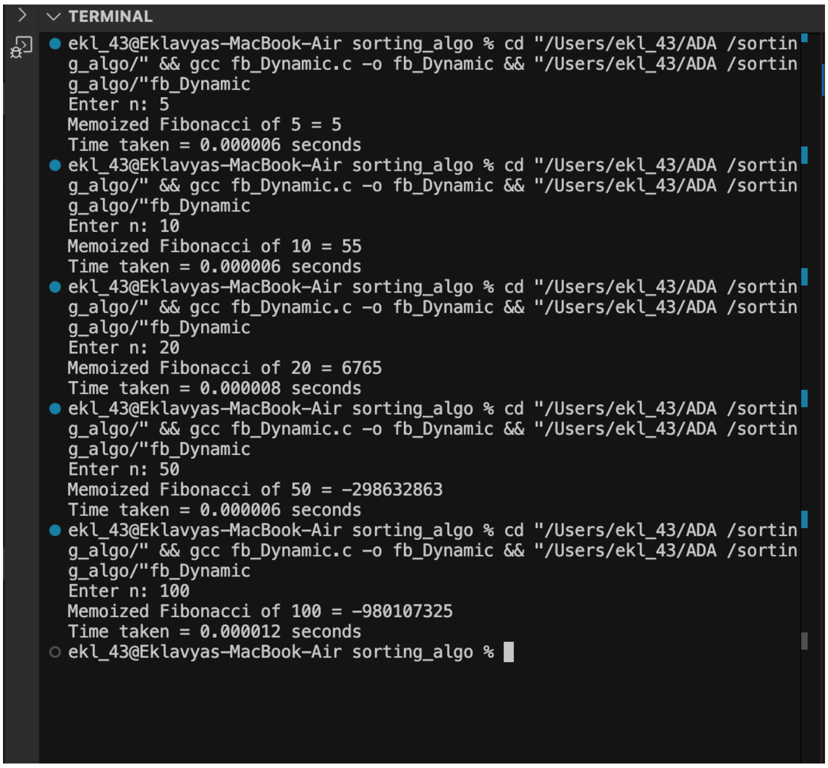
```
● ekl_43@Eklavyas-MacBook-Air sorting_algo % cd "/Users/ekl_43/ADA /sortin
  g_algo/" && gcc fb_iterative.c -o fb_iterative && "/Users/ekl_43/ADA /so
  rting_algo/"fb_iterative
  Enter n: 5
  Iterative Fibonacci of 5 = 5
  Time taken = 0.000007 seconds
● ekl_43@Eklavyas-MacBook-Air sorting_algo % cd "/Users/ekl_43/ADA /sorting_algo/" && gcc fb_iterative.c -o fb_i
  terative && "/Users/ekl_43/ADA /sorting_algo/"fb_iterative
  Enter n: 10
  Iterative Fibonacci of 10 = 55
  Time taken = 0.000007 seconds
● ekl_43@Eklavyas-MacBook-Air sorting_algo % cd "/Users/ekl_43/ADA /sorting_algo/" && gcc fb_iterative.c -o fb_i
  terative && "/Users/ekl_43/ADA /sorting_algo/"fb_iterative
  Enter n: 20^[[A
  Iterative Fibonacci of 20 = 6765
  Time taken = 0.000004 seconds
● ekl_43@Eklavyas-MacBook-Air sorting_algo % cd "/Users/ekl_43/ADA /sorting_algo/" && gcc fb_iterative.c -o fb_i
  terative && "/Users/ekl_43/ADA /sorting_algo/"fb_iterative
  Enter n: 50
  Iterative Fibonacci of 50 = -298632863
  Time taken = 0.000008 seconds
● ekl_43@Eklavyas-MacBook-Air sorting_algo % cd "/Users/ekl_43/ADA /sorting_algo/" && gcc fb_iterative.c -o fb_i
  terative && "/Users/ekl_43/ADA /sorting_algo/"fb_iterative
  Enter n: 100
  Iterative Fibonacci of 100 = -980107325
  Time taken = 0.000014 seconds
○ ekl_43@Eklavyas-MacBook-Air sorting_algo % █
```

4(c) To implement and analyze different approaches for generating Fibonacci numbers and compare their time and space complexities using **memoization approach(Dynamic programming)**

C CODE:

```
#include <stdio.h>
#include <time.h>
#define MAX 10000
int memo[MAX];
// Memoization (Top-Down DP)
int fib_memo(int n) {
if (memo[n] != -1) return memo[n];
if (n <= 1) memo[n] = n;
else memo[n] = fib_memo(n-1) + fib_memo(n-2);
return memo[n];
}
int main() {
int n;printf("Enter n: ");
scanf("%d", &n);
for (int i = 0; i < MAX; i++) memo[i] = -1; // init
clock_t start = clock();
int result = fib_memo(n);
clock_t end = clock();
double time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;
printf("Memoized Fibonacci of %d = %d\n", n, result);
printf("Time taken = %f seconds\n", time_taken);
return 0;
}
```

Output:



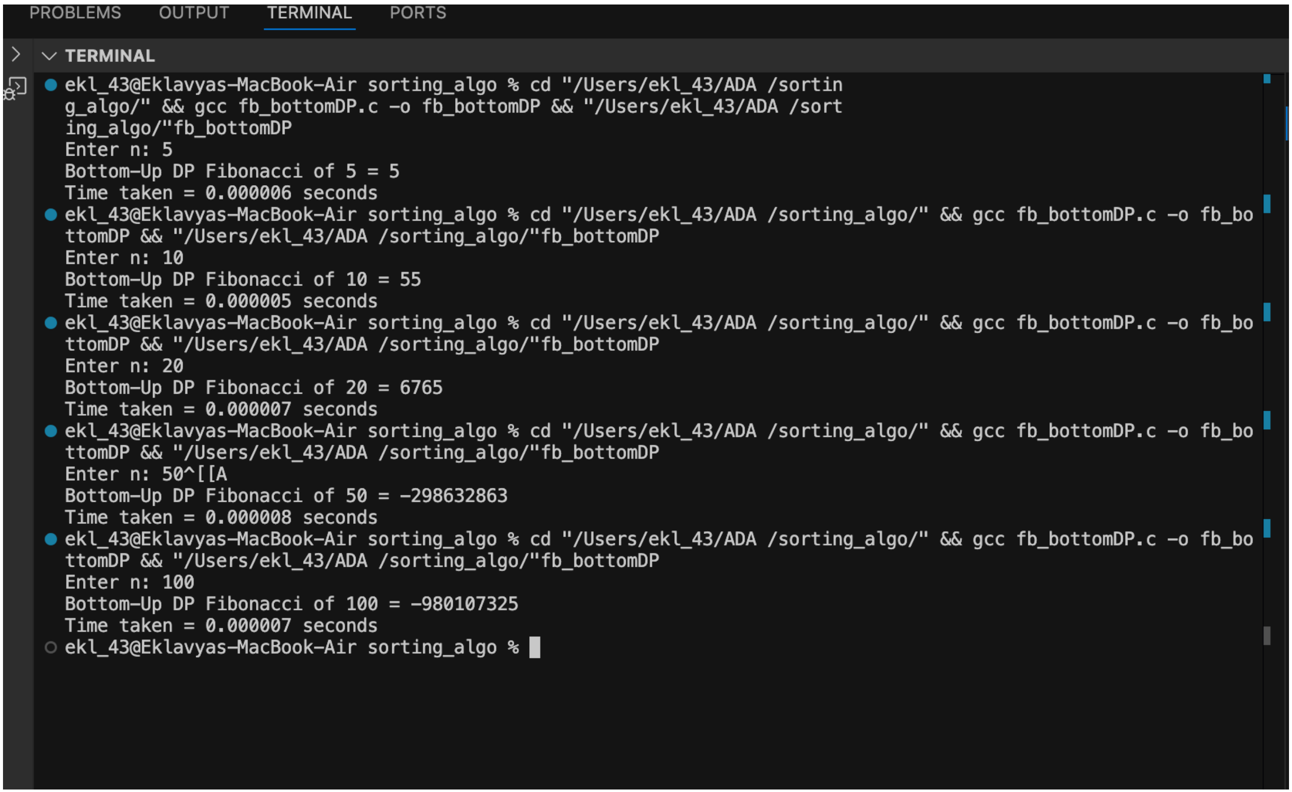
```
● ekl_43@Eklavyas-MacBook-Air sorting_algo % cd "/Users/ekl_43/ADA /sortin
  g_algo/" && gcc fb_Dynamic.c -o fb_Dynamic && "/Users/ekl_43/ADA /sortin
  g_algo/"fb_Dynamic
  Enter n: 5
  Memoized Fibonacci of 5 = 5
  Time taken = 0.000006 seconds
● ekl_43@Eklavyas-MacBook-Air sorting_algo % cd "/Users/ekl_43/ADA /sortin
  g_algo/" && gcc fb_Dynamic.c -o fb_Dynamic && "/Users/ekl_43/ADA /sortin
  g_algo/"fb_Dynamic
  Enter n: 10
  Memoized Fibonacci of 10 = 55
  Time taken = 0.000006 seconds
● ekl_43@Eklavyas-MacBook-Air sorting_algo % cd "/Users/ekl_43/ADA /sortin
  g_algo/" && gcc fb_Dynamic.c -o fb_Dynamic && "/Users/ekl_43/ADA /sortin
  g_algo/"fb_Dynamic
  Enter n: 20
  Memoized Fibonacci of 20 = 6765
  Time taken = 0.000008 seconds
● ekl_43@Eklavyas-MacBook-Air sorting_algo % cd "/Users/ekl_43/ADA /sortin
  g_algo/" && gcc fb_Dynamic.c -o fb_Dynamic && "/Users/ekl_43/ADA /sortin
  g_algo/"fb_Dynamic
  Enter n: 50
  Memoized Fibonacci of 50 = -298632863
  Time taken = 0.000006 seconds
● ekl_43@Eklavyas-MacBook-Air sorting_algo % cd "/Users/ekl_43/ADA /sortin
  g_algo/" && gcc fb_Dynamic.c -o fb_Dynamic && "/Users/ekl_43/ADA /sortin
  g_algo/"fb_Dynamic
  Enter n: 100
  Memoized Fibonacci of 100 = -980107325
  Time taken = 0.000012 seconds
○ ekl_43@Eklavyas-MacBook-Air sorting_algo % █
```

4(d) To implement and analyze different approaches for generating Fibonacci numbers and compare their time and space complexities using **Bottom Up Approach (Dynamic Programming)**

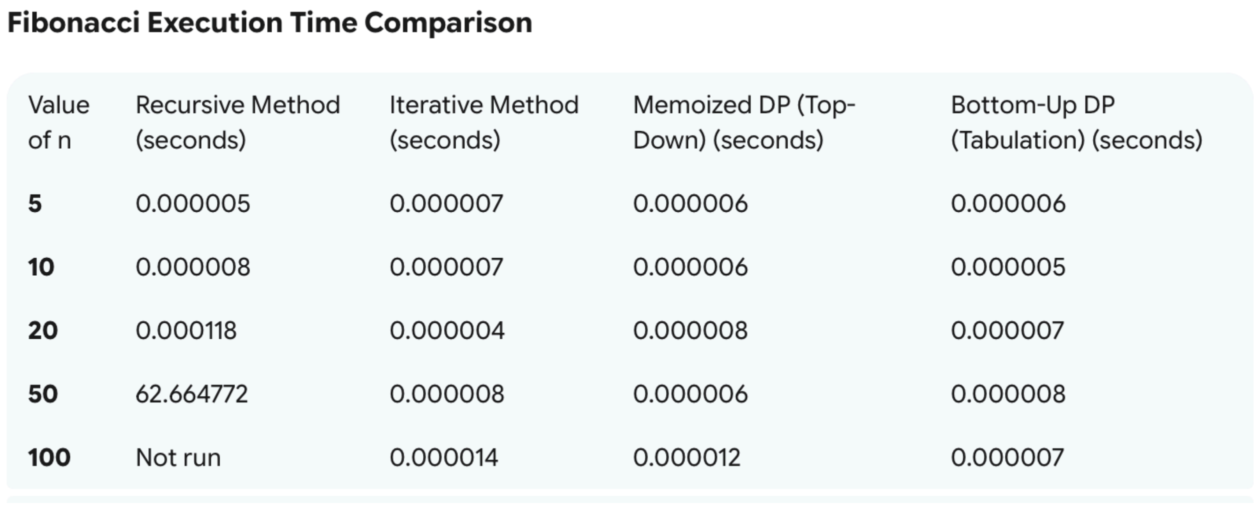
C CODE:

```
#include <stdio.h>
#include <time.h>
// Bottom-Up DP
int fib_bottomup(int n) {
if (n <= 1) return n;
int dp[n+1];
dp[0] = 0; dp[1] = 1;
for (int i = 2; i <= n; i++)
dp[i] = dp[i-1] + dp[i-2];
return dp[n];
}
int main() {
int n;
printf("Enter n: ");
scanf("%d", &n);
clock_t start = clock();
int result = fib_bottomup(n);
clock_t end = clock();
double time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;
printf("Bottom-Up DP Fibonacci of %d = %d\n", n, result);
printf("Time taken = %f seconds\n", time_taken);
return 0;
}
```

Output:

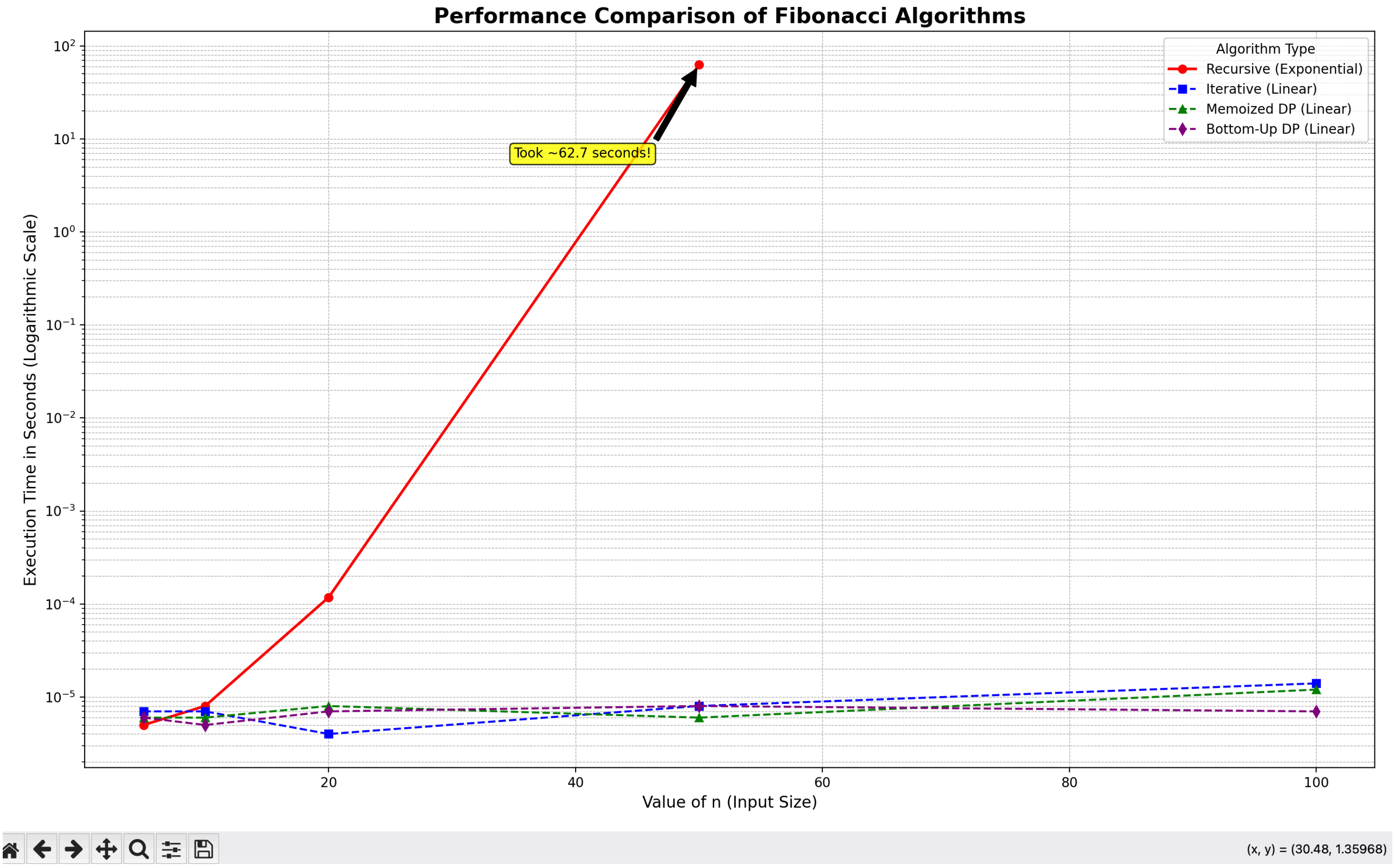


```
● ekl_43@Eklavyas-MacBook-Air sorting_algo % cd "/Users/ekl_43/ADA /sortin
  g_algo/" && gcc fb_bottomDP.c -o fb_bottomDP && "/Users/ekl_43/ADA /sort
  ing_algo/"fb_bottomDP
  Enter n: 5
  Bottom-Up DP Fibonacci of 5 = 5
  Time taken = 0.000006 seconds
● ekl_43@Eklavyas-MacBook-Air sorting_algo % cd "/Users/ekl_43/ADA /sorting_algo/" && gcc fb_bottomDP.c -o fb_bo
  ttomDP && "/Users/ekl_43/ADA /sorting_algo/"fb_bottomDP
  Enter n: 10
  Bottom-Up DP Fibonacci of 10 = 55
  Time taken = 0.000005 seconds
● ekl_43@Eklavyas-MacBook-Air sorting_algo % cd "/Users/ekl_43/ADA /sorting_algo/" && gcc fb_bottomDP.c -o fb_bo
  ttomDP && "/Users/ekl_43/ADA /sorting_algo/"fb_bottomDP
  Enter n: 20
  Bottom-Up DP Fibonacci of 20 = 6765
  Time taken = 0.000007 seconds
● ekl_43@Eklavyas-MacBook-Air sorting_algo % cd "/Users/ekl_43/ADA /sorting_algo/" && gcc fb_bottomDP.c -o fb_bo
  ttomDP && "/Users/ekl_43/ADA /sorting_algo/"fb_bottomDP
  Enter n: 50^[[A
  Bottom-Up DP Fibonacci of 50 = -298632863
  Time taken = 0.000008 seconds
● ekl_43@Eklavyas-MacBook-Air sorting_algo % cd "/Users/ekl_43/ADA /sorting_algo/" && gcc fb_bottomDP.c -o fb_bo
  ttomDP && "/Users/ekl_43/ADA /sorting_algo/"fb_bottomDP
  Enter n: 100
  Bottom-Up DP Fibonacci of 100 = -980107325
  Time taken = 0.000007 seconds
○ ekl_43@Eklavyas-MacBook-Air sorting_algo % 
```



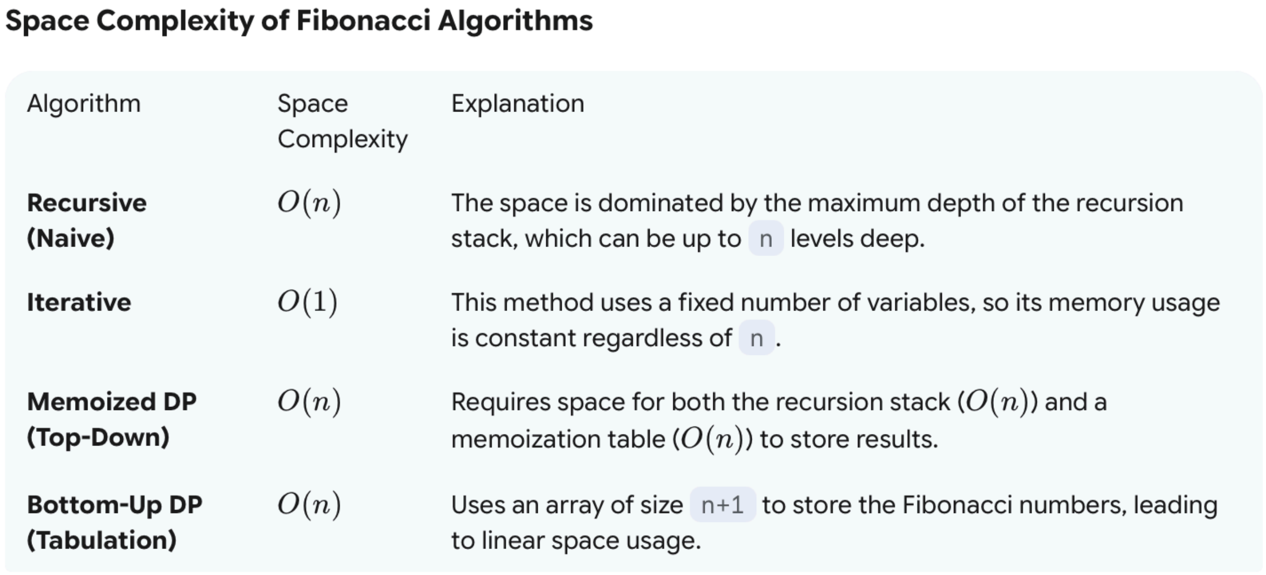
Fibonacci Execution Time Comparison

Value of n	Recursive Method (seconds)	Iterative Method (seconds)	Memoized DP (Top-Down) (seconds)	Bottom-Up DP (Tabulation) (seconds)
5	0.000005	0.000007	0.000006	0.000006
10	0.000008	0.000007	0.000006	0.000005
20	0.000118	0.000004	0.000008	0.000007
50	62.664772	0.000008	0.000006	0.000008
100	Not run	0.000014	0.000012	0.000007



Performance Comparison of Fibonacci Algorithms

Took ~62.7 seconds!



Space Complexity of Fibonacci Algorithms

Algorithm	Space Complexity	Explanation
Recursive (Naive)	$O(n)$	The space is dominated by the maximum depth of the recursion stack, which can be up to `n` levels deep.
Iterative	$O(1)$	This method uses a fixed number of variables, so its memory usage is constant regardless of `n`.
Memoized DP (Top-Down)	$O(n)$	Requires space for both the recursion stack ($O(n)$) and a memoization table ($O(n)$) to store results.
Bottom-Up DP (Tabulation)	$O(n)$	Uses an array of size `n+1` to store the Fibonacci numbers, leading to linear space usage.

CONCLUSION:

Recursive Approach:

- Simple and elegant, but **very slow for large `n`** (**O(2^n)**).
- **High space usage (O(n))** due to recursion stack.
- Practical only for small values of `n`.

Iterative Approach:

- Fast (O(n) time) and memory-efficient (O(1) space).
- Ideal for computing a **single Fibonacci number**.

- Memoization (Top-Down DP):

○ Reduces time complexity to O(n) using memoization.
○ **Space complexity O(n)** for memo array + recursion stack.
○ Useful when reusing previously computed Fibonacci numbers.

- Bottom-Up DP:

○ O(n) time and O(n) space, can be optimized to O(1) space.
○ Avoids recursion stack, **best for large `n`**.

Key Takeaways:

- Recursive method is impractical for large inputs.
- Iterative and optimized Bottom-Up DP are **fastest and most memory-efficient**.
- Memoization is a good balance if you want **recursion with efficiency**.