AOA IA2

Analysis of
FIBONACCI NUMBERS

Ronak Rathod-16010122156,
Nikhil Patil-16010122136,
Hyder Presswala-16010122151

Introduction:

The Research Paper discusses the growing impact of information and communication technology in the era of globalization, particularly focusing on advancements in algorithms and programming. It mentions two types of programming algorithms: static programming and dynamic programming. The application of these algorithms is illustrated using the example of calculating Fibonacci numbers, which are derived from the sum of two consecutive numbers and have recursive elements.

The Research Paper introduces the research topic of comparing the efficiency of calculating Fibonacci numbers using static programming (recursive algorithm) versus dynamic programming (top-down and bottom-up approaches), aiming to determine the most optimal time efficiency by analyzing the execution time of these algorithms.

Types of PROGRAMMING ALGORITHMS:

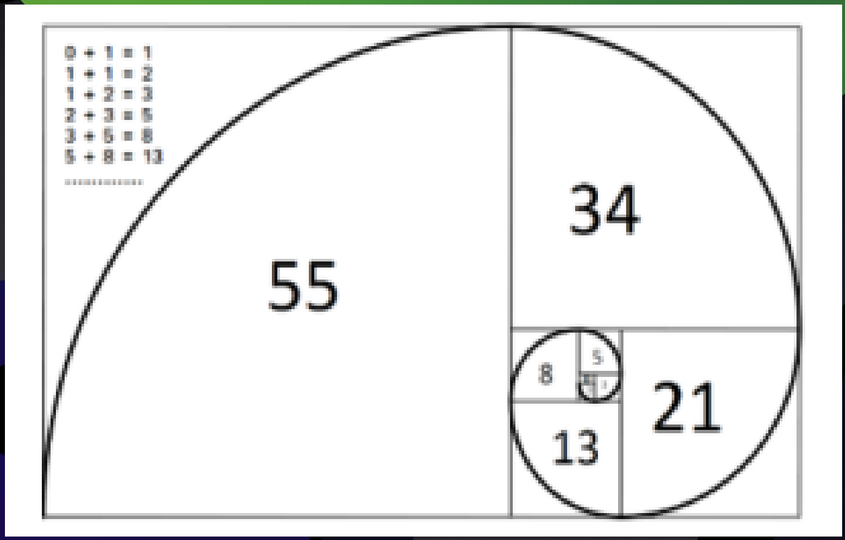
1. Static programming
2. Dynamic Programming
   - 2.1 Top Down
   - 2.2 Bottom Up

What is Fibonacci numbers??

The Fibonacci number series is a sequence of numbers discovered by Leonardo Fibonacci, starting with 1 and 1, where each subsequent term is the sum of the two preceding terms. This series generates a ratio known as the Golden Ratio, approximately 1.618, through dividing consecutive Fibonacci numbers.

The sequence is defined by the recurrence relation Fn = Fn-1 + Fn-2,
with initial values F0 = 0 and F1 = 1.

The Fibonacci sequence includes numbers like 1, 2, 3, 5, 8, 13, 21, and so on, exhibiting a consistent ratio between adjacent terms.



1

Static Programming:

Static programming refers to a programming paradigm or approach where the program's behavior is determined at compile time and remains fixed during execution. In static programming languages, variables, data types, and memory allocation are typically defined and set during the compilation phase. This means that once the program is compiled, the instructions and structure of the program do not change while it is running.

```
function fib(n)
    if n = 0 or n = 1
        return 1
    else
        return fib(n-1) + fib(n-2)
```

1

Static Programming Implementation:

In this we are taking Fibonacci numbers from 1 – 5000 but in range like first 1- 10 then 10 – 20 then 20 -30 so on till 5000.

```python
import time

# Fibonacci function with memoization
def fibonacci(n):
    if n <= 1:
        return n
    else:
        return fibonacci(n-1) + fibonacci(n-2)

# Calculate Fibonacci numbers from 1 to 10 and measure time
def calculate_fibonacci(number):
    start_time = time.perf_counter()
    for i in range(1, number + 1):  # Corrected the range
        fibonacci(i)
    end_time = time.perf_counter()
    time_taken = end_time - start_time
    print(f"Time taken to compute Fibonacci numbers from 1 to {number}:
{time_taken:.6f} seconds")
    return time_taken

# Main function
def main():
    avg_time = 0.0
    for i in range(10, 31):
        time_taken = calculate_fibonacci(i)
        avg_time += time_taken
    avg_time /= 90  # Adjusted the divisor to calculate the average
    print("Average Time is  : {:.6f} seconds".format(avg_time))

if __name__ == "__main__":
    main()
```

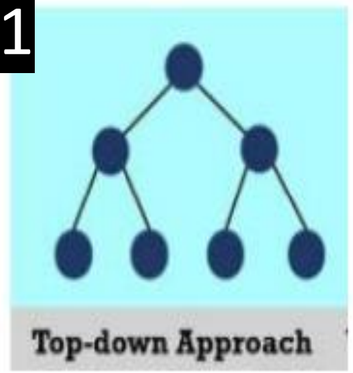
Output:

```
jarvis@JARVIS MINGW64 /1/KJSCE/SY/my_sem_4/AOA
$ python staticFIB.py
Time taken to compute Fibonacci numbers from 1 to 10: 0.000028 seconds
Time taken to compute Fibonacci numbers from 1 to 11: 0.000066 seconds
Time taken to compute Fibonacci numbers from 1 to 12: 0.000091 seconds
Time taken to compute Fibonacci numbers from 1 to 13: 0.000124 seconds
Time taken to compute Fibonacci numbers from 1 to 14: 0.000192 seconds
Time taken to compute Fibonacci numbers from 1 to 15: 0.000290 seconds
Time taken to compute Fibonacci numbers from 1 to 16: 0.000470 seconds
Time taken to compute Fibonacci numbers from 1 to 17: 0.000773 seconds
Time taken to compute Fibonacci numbers from 1 to 18: 0.001255 seconds
Time taken to compute Fibonacci numbers from 1 to 19: 0.002035 seconds
Time taken to compute Fibonacci numbers from 1 to 20: 0.003316 seconds
Time taken to compute Fibonacci numbers from 1 to 21: 0.005492 seconds
Time taken to compute Fibonacci numbers from 1 to 22: 0.009042 seconds
Time taken to compute Fibonacci numbers from 1 to 23: 0.015007 seconds
Time taken to compute Fibonacci numbers from 1 to 24: 0.025639 seconds
Time taken to compute Fibonacci numbers from 1 to 25: 0.038776 seconds
Time taken to compute Fibonacci numbers from 1 to 26: 0.058675 seconds
Time taken to compute Fibonacci numbers from 1 to 27: 0.092646 seconds
Time taken to compute Fibonacci numbers from 1 to 28: 0.143497 seconds
Time taken to compute Fibonacci numbers from 1 to 29: 0.227467 seconds
Time taken to compute Fibonacci numbers from 1 to 30: 0.379586 seconds
Average Time is  : 0.011161 seconds
```

Dynamic Programming:

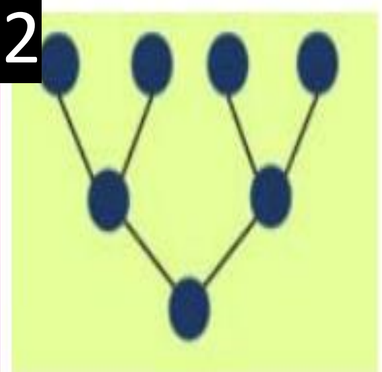
Dynamic programming is a method used in computer science and mathematics to solve complex problems by breaking them down into smaller, overlapping subproblems and solving each subproblem only once, storing the solutions to avoid redundant computations. This approach optimizes efficiency by leveraging previously computed solutions and is characterized by its use of optimal substructure and techniques like memoization or tabulation. Dynamic programming is commonly applied to problems with overlapping subproblems, such as optimization problems, shortest path problems, and sequence alignment problems, and is implemented using either a bottom-up or top-down approach.

2.1




```
if n = 0 then
   return 0
else if n = 1 or n = 2 then
   return 1
else
   fib[1] = 1
   fib[2] = 1
   for i = 3 to n do
        fib[i] = fib[i-1] + fib[i-2]
   endfor
   return fib[i]
endif
```

2.2




```
prevF ← 0
currF ← 1
for i←2 to n do
   newF ← prevF + currF
   prevF ← currF
   currF ← newF
endfor
return currF
```

2.1

Top-Down approach:

```python
import time

def fibonacci(n):
    if n <= 1:
        return n
    fib = [0] * (n + 1)
    fib[1] = 1
    for i in range(2, n + 1):
        fib[i] = fib[i - 1] + fib[i - 2]
    return fib[n]

def calculate_fibonacci(number):
    start_time = time.perf_counter()
    for i in range(1, number + 1):
        fibonacci(i)
    end_time = time.perf_counter()
    time_taken = end_time - start_time
    print(f"Time taken to compute Fibonacci numbers from 1 to {number}:
{time_taken:.6f} seconds")
    return time_taken

# Main function
def main():
    avg_time = 0.0
    for i in range(10, 31):
        time_taken = calculate_fibonacci(i)
        avg_time += time_taken
    avg_time /= 90  # Adjusted the divisor to calculate the average
    print("Average Time is  : {:.6f} seconds".format(avg_time))

if __name__ == "__main__":
    main()
```

Output:

```
jarvis@JARVIS MINGW64 /1/KJSCE/SY/my_sem_4/AOA
$ python staticFIB.py
Time taken to compute Fibonacci numbers from 1 to 10: 0.000012 seconds
Time taken to compute Fibonacci numbers from 1 to 11: 0.000008 seconds
Time taken to compute Fibonacci numbers from 1 to 12: 0.000011 seconds
Time taken to compute Fibonacci numbers from 1 to 13: 0.000031 seconds
Time taken to compute Fibonacci numbers from 1 to 14: 0.000011 seconds
Time taken to compute Fibonacci numbers from 1 to 15: 0.000011 seconds
Time taken to compute Fibonacci numbers from 1 to 16: 0.000012 seconds
Time taken to compute Fibonacci numbers from 1 to 17: 0.000028 seconds
Time taken to compute Fibonacci numbers from 1 to 18: 0.000014 seconds
Time taken to compute Fibonacci numbers from 1 to 19: 0.000017 seconds
Time taken to compute Fibonacci numbers from 1 to 20: 0.000018 seconds
Time taken to compute Fibonacci numbers from 1 to 21: 0.000018 seconds
Time taken to compute Fibonacci numbers from 1 to 22: 0.000020 seconds
Time taken to compute Fibonacci numbers from 1 to 23: 0.000030 seconds
Time taken to compute Fibonacci numbers from 1 to 24: 0.000025 seconds
Time taken to compute Fibonacci numbers from 1 to 25: 0.000033 seconds
Time taken to compute Fibonacci numbers from 1 to 26: 0.000026 seconds
Time taken to compute Fibonacci numbers from 1 to 27: 0.000028 seconds
Time taken to compute Fibonacci numbers from 1 to 28: 0.000037 seconds
Time taken to compute Fibonacci numbers from 1 to 29: 0.000039 seconds
Time taken to compute Fibonacci numbers from 1 to 30: 0.000041 seconds
Average Time is  : 0.000005 seconds
```

2.2

Bottom-Up approach:

```python
import time

# Fibonacci function with memoization
def fibonacci(n, memo={}):
    if n in memo:
        return memo[n]
    if n <= 1:
        return n
    memo[n] = fibonacci(n-1, memo) + fibonacci(n-2, memo)
    return memo[n]

# Calculate Fibonacci numbers from 1 to 10 and measure time
def calculate_fibonacci(number):
    start_time = time.perf_counter()
    for i in range(1, number + 1):
        fibonacci(i)
    end_time = time.perf_counter()
    time_taken = end_time - start_time
    print(f"Time taken to compute Fibonacci numbers from 1 to {number}:
{time_taken:.6f} seconds")
    return time_taken

# Main function
def main():
    avg_time = 0.0
    for i in range(10, 31):
        time_taken = calculate_fibonacci(i)
        avg_time += time_taken
    avg_time /= 90  # Adjusted the divisor to calculate the average
    print("Average Time is  : {:.6f} seconds".format(avg_time))

if __name__ == "__main__":
```

Output:

```
jarvis@JARVIS MINGW64 /1/KJSCE/SY/my_sem_4/AOA
$ python staticFIB.py
Time taken to compute Fibonacci numbers from 1 to 10: 0.000008 seconds
Time taken to compute Fibonacci numbers from 1 to 11: 0.000003 seconds
Time taken to compute Fibonacci numbers from 1 to 12: 0.000003 seconds
Time taken to compute Fibonacci numbers from 1 to 13: 0.000002 seconds
Time taken to compute Fibonacci numbers from 1 to 14: 0.000002 seconds
Time taken to compute Fibonacci numbers from 1 to 15: 0.000002 seconds
Time taken to compute Fibonacci numbers from 1 to 16: 0.000002 seconds
Time taken to compute Fibonacci numbers from 1 to 17: 0.000002 seconds
Time taken to compute Fibonacci numbers from 1 to 18: 0.000002 seconds
Time taken to compute Fibonacci numbers from 1 to 19: 0.000002 seconds
Time taken to compute Fibonacci numbers from 1 to 20: 0.000002 seconds
Time taken to compute Fibonacci numbers from 1 to 21: 0.000002 seconds
Time taken to compute Fibonacci numbers from 1 to 22: 0.000002 seconds
Time taken to compute Fibonacci numbers from 1 to 23: 0.000008 seconds
Time taken to compute Fibonacci numbers from 1 to 24: 0.000003 seconds
Time taken to compute Fibonacci numbers from 1 to 25: 0.000003 seconds
Time taken to compute Fibonacci numbers from 1 to 26: 0.000003 seconds
Time taken to compute Fibonacci numbers from 1 to 27: 0.000013 seconds
Time taken to compute Fibonacci numbers from 1 to 28: 0.000003 seconds
Time taken to compute Fibonacci numbers from 1 to 29: 0.000003 seconds
Time taken to compute Fibonacci numbers from 1 to 30: 0.000003 seconds
Average Time is  : 0.000001 seconds
```

Now as we can see both approaches of dynamic programming are faster than static programming

Now to find which one of the above approaches of dynamic programming is faster we will increase n

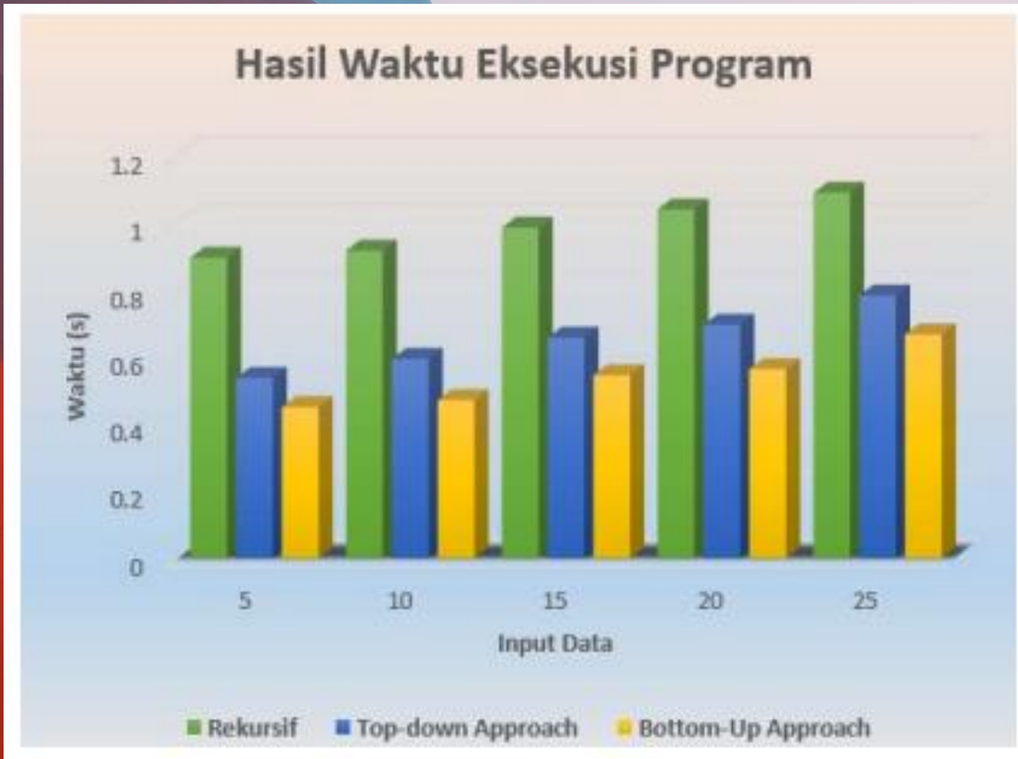
Top Down For 1 – 1000

```
 Time taken to compute Fibonacci numbers from 1 to 992: 0.039152 seconds
 Time taken to compute Fibonacci numbers from 1 to 993: 0.041263 seconds
 Time taken to compute Fibonacci numbers from 1 to 994: 0.036640 seconds
 Time taken to compute Fibonacci numbers from 1 to 995: 0.043155 seconds
 Time taken to compute Fibonacci numbers from 1 to 996: 0.039393 seconds
 Time taken to compute Fibonacci numbers from 1 to 997: 0.042430 seconds
 Time taken to compute Fibonacci numbers from 1 to 998: 0.041446 seconds
 Time taken to compute Fibonacci numbers from 1 to 999: 0.042171 seconds
 Average Time is  : 0.164908 seconds
```

Bottom-Up

```
Time taken to compute Fibonacci numbers from 1 to 992: 0.000101 seconds
Time taken to compute Fibonacci numbers from 1 to 993: 0.000121 seconds
Time taken to compute Fibonacci numbers from 1 to 994: 0.000115 seconds
Time taken to compute Fibonacci numbers from 1 to 995: 0.000108 seconds
Time taken to compute Fibonacci numbers from 1 to 996: 0.000127 seconds
Time taken to compute Fibonacci numbers from 1 to 997: 0.000104 seconds
Time taken to compute Fibonacci numbers from 1 to 998: 0.000103 seconds
Time taken to compute Fibonacci numbers from 1 to 999: 0.000104 seconds
Average Time is  : 0.001106 seconds
```

Here we can see that Bottom up approach wins.



Results:

Result table Fibonacci numbers and execution time dynamic programming Top-down approach

Fib to-	Fibonacci Summary	Execution Time (ms)
15	610	658.00
20	6765	697.00
25	75025	784.00
Average		654.40

Result table Fibonacci numbers and execution time dynamic programming Bottom-up approach

Fib to-	Fibonacci Summary Result	Execution Time (ms)
5	5	451.00
10	55	472.00
15	610	546.00
20	6765	564.00
25	75025	668.00
Average		540.20

Result table Fibonacci numbers and execution time static programming

Fib to-	Fibonacci Summary Result	Execution Time (ms)
5	5	897.00
10	55	918.00
15	610	967.00
20	6765	1041.00
25	75025	1091.00
Average		982.80

Conclusion:

The research compares the efficiency of Fibonacci number calculation algorithms, specifically static programming and dynamic programming algorithms. Dynamic programming is found to be more time-efficient than static programming due to its ability to avoid repetitive calculations. However, within dynamic programming, the bottom-up approach proves to be more efficient than the top-down approach, despite both having the same complexity of O(n). This efficiency difference is attributed to the bottom-up approach's use of traversal methods for quicker calculations compared to the recursive method in the top-down approach, which incurs higher processor workload due to context switching. Overall, the research recommends utilizing dynamic programming, particularly the bottom-up approach, for optimal time efficiency in calculating Fibonacci numbers.


Thank
You