COMP108
Data Structures and Algorithms

Divide-and-Conquer Algorithms (Part III Fibonacci Numbers)

Professor Prudence Wong

pwong@liverpool.ac.uk

Practice Exam released (revision lecture next week)

2022-23

This week: Tutorials on Wed / Fri (check your timetable)
tutorial submission by Friday
Programming assessment feedback to be released in Week 12

Fibonacci Rabbits

A pair of rabbits, one month old, is too young to reproduce. Suppose that in their second month, and every month thereafter, they produce a new pair.

end of month-0

Fibonacci Rabbits

A pair of rabbits, one month old, is too young to reproduce. Suppose that in their second month, and every month thereafter, they produce a new pair.

end of month-0

end of month-1

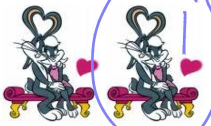
Fibonacci Rabbits

A pair of rabbits, one month old, is too young to reproduce. Suppose that in their second month, and every month thereafter, they produce a new pair.

0

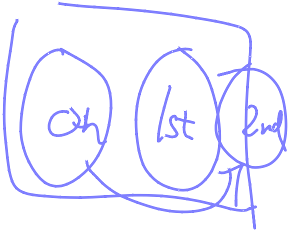
end of
month-0

end of
month-1



1

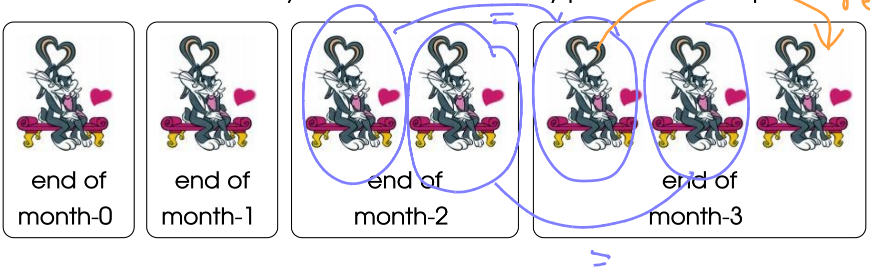
end of
month-2



Fibonacci Rabbits

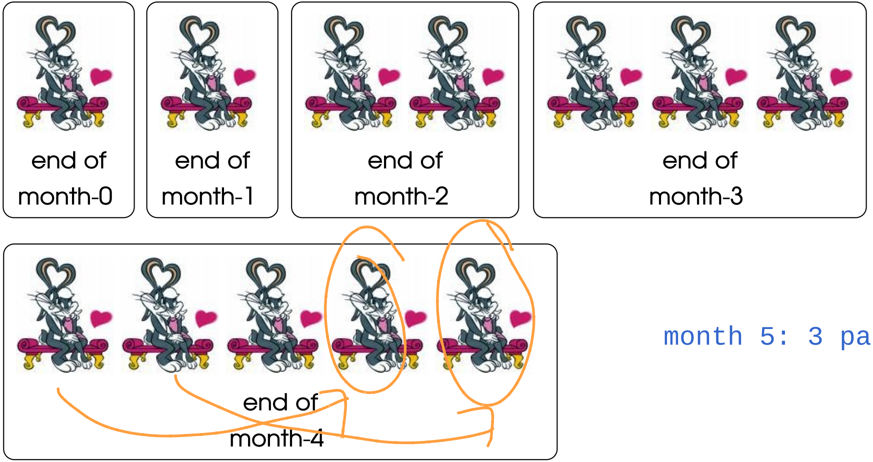
A pair of rabbits, one month old, is too young to reproduce. Suppose that in their second month, and every month thereafter, they produce a new pair.

reproduce



Fibonacci Rabbits

A pair of rabbits, one month old, is too young to reproduce. Suppose that in their second month, and every month thereafter, they produce a new pair.



month 5: 3 pairs to reproduce

Fibonacci Rabbits

A pair of rabbits, one month old, is too young to reproduce. Suppose that in their second month, and every month thereafter, they produce a new pair.


end of month-0


end of month-1


end of month-2


end of month-3


end of month-4

How many at end of month-5, 6, 7 and so on?

Fibonacci Numbers in Nature - Petals on flowers

1 petal:
white calla lily

2 petals:
euphorbia

3 petals:
trillium

5 petals:
columbine

8 petals:
bloodroot

13 petals:
black-eyed susan

21 petals:
shasta daisy

34 petals:
field daisy

Fibonacci Numbers

Fibonacci number $F(n)$

$$F(n) = \begin{cases} 1 & \text{if } n = 0 \text{ or } 1 \\ F(n-1) + F(n-2) & \text{if } n > 1 \end{cases}$$

existing new

recurrence

Fibonacci Numbers

Fibonacci number $F(n)$

$$F(n) = \begin{cases} 1 & \text{if } n = 0 \text{ or } 1 \\ F(n-1) + F(n-2) & \text{if } n > 1 \end{cases}$$

n	0	1	2	3	4	5	6	7	8	9	10
$F(n)$	1	1	2	3	5	8	13	21	34	55	89

Fibonacci Numbers

Fibonacci number $F(n)$

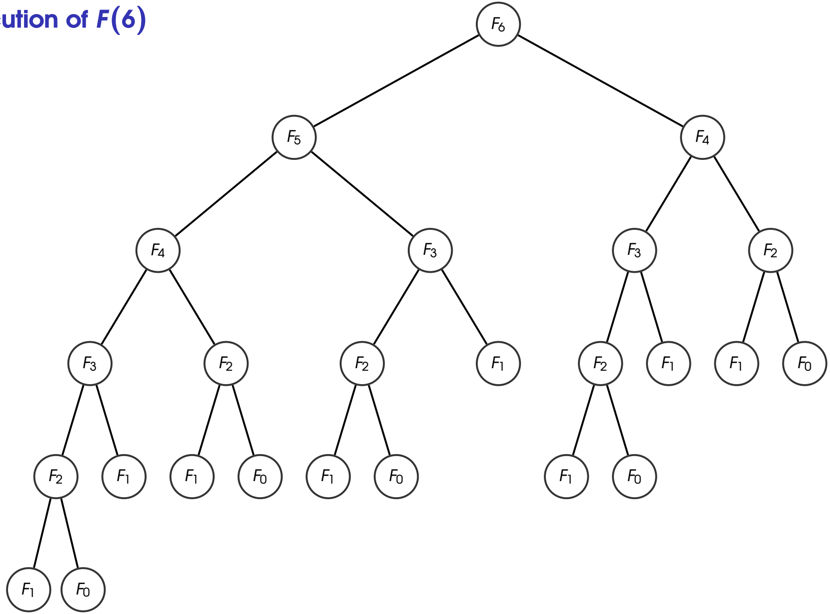
$$F(n) = \begin{cases} 1 & \text{if } n = 0 \text{ or } 1 \\ F(n-1) + F(n-2) & \text{if } n > 1 \end{cases}$$

n	0	1	2	3	4	5	6	7	8	9	10
$F(n)$	1	1	2	3	5	8	13	21	34	55	89

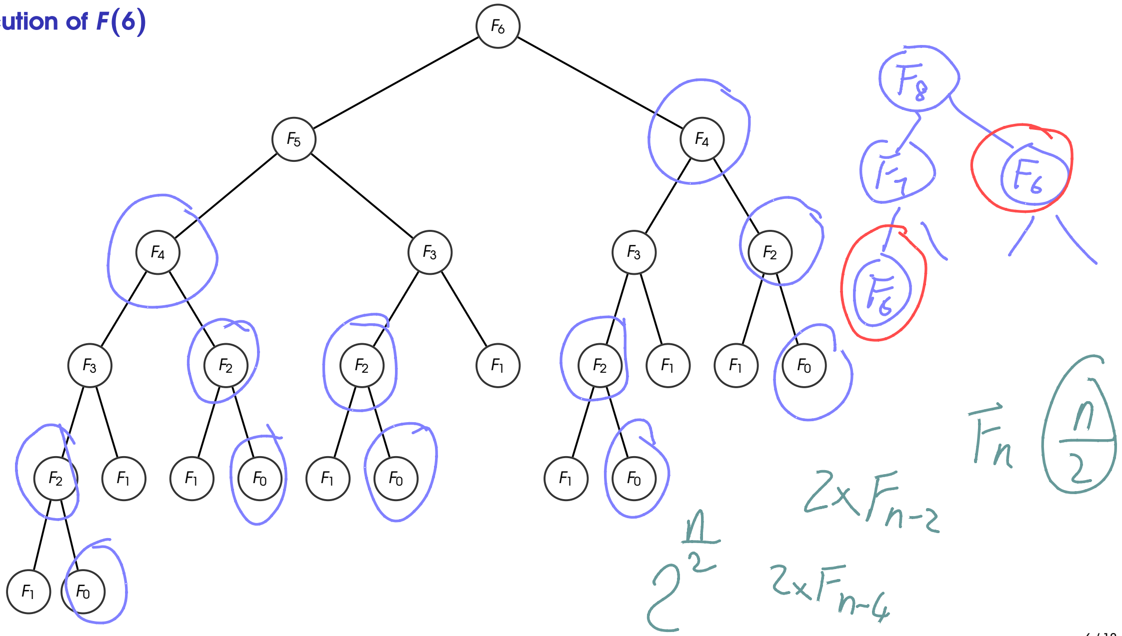
Pseudo code for the recursive algorithm:

```
Algorithm F(n)
    if n == 0 OR n == 1 then
        return 1
    else
    return F(n − 1) + F(n − 2)
```

Execution of $F(6)$



Execution of $F(6)$



Time Complexity

To analyze the time complexity of calculating Fibonacci numbers, we use a mathematical tool called **recurrence**.

A recurrence is an equation or inequality that describes a function in terms of its value on smaller inputs.

To solve a recurrence is to derive asymptotic bounds on the solution

Time Complexity

To analyze the time complexity of calculating Fibonacci numbers, we use a mathematical tool called **recurrence**.

Let $T(n)$ denote the time to calculate Fibonacci number $F(n)$.

A recurrence is an equation or inequality that describes a function in terms of its value on smaller inputs.

To solve a recurrence is to derive asymptotic bounds on the solution

Time Complexity

To analyze the time complexity of calculating Fibonacci numbers, we use a mathematical tool called **recurrence**.

Let $T(n)$ denote the time to calculate Fibonacci number $F(n)$.

$$T(n) = \begin{cases} 1 & \text{if } n = 0 \text{ or } 1 \\ T(n-1) + T(n-2) + 1 & \text{if } n > 1 \end{cases}$$

A recurrence is an equation or inequality that describes a function in terms of its value on smaller inputs.

To solve a recurrence is to derive asymptotic bounds on the solution

Time Complexity

To analyze the time complexity of calculating Fibonacci numbers, we use a mathematical tool called **recurrence**.

Let $T(n)$ denote the time to calculate Fibonacci number $F(n)$.

$$T(n) = \begin{cases} 1 & \text{if } n = 0 \text{ or } 1 \\ T(n-1) + T(n-2) + 1 & \text{if } n > 1 \end{cases}$$

- $T(n-1)$: time to calculate $F(n-1)$,

A recurrence is an equation or inequality that describes a function in terms of its value on smaller inputs.

To solve a recurrence is to derive asymptotic bounds on the solution

Time Complexity

To analyze the time complexity of calculating Fibonacci numbers, we use a mathematical tool called **recurrence**.

Let $T(n)$ denote the time to calculate Fibonacci number $F(n)$.

$$T(n) = \begin{cases} 1 & \text{if } n = 0 \text{ or } 1 \\ T(n-1) + T(n-2) + 1 & \text{if } n > 1 \end{cases}$$

- $T(n-1)$: time to calculate $F(n-1)$,
- $T(n-2)$: time to calculate $F(n-2)$,

A recurrence is an equation or inequality that describes a function in terms of its value on smaller inputs.

To solve a recurrence is to derive asymptotic bounds on the solution

Time Complexity

To analyze the time complexity of calculating Fibonacci numbers, we use a mathematical tool called **recurrence**.

Let $T(n)$ denote the time to calculate Fibonacci number $F(n)$.

$$T(n) = \begin{cases} 1 & \text{if } n = 0 \text{ or } 1 \\ T(n-1) + T(n-2) + 1 & \text{if } n > 1 \end{cases}$$

- $T(n-1)$: time to calculate $F(n-1)$,
- $T(n-2)$: time to calculate $F(n-2)$,
- plus 1: time to add $F(n-1)$ and $F(n-2)$

A recurrence is an equation or inequality that describes a function in terms of its value on smaller inputs.
To solve a recurrence is to derive asymptotic bounds on the solution

Time Complexity

To analyze the time complexity of calculating Fibonacci numbers, we use a mathematical tool called **recurrence**.

Let $T(n)$ denote the time to calculate Fibonacci number $F(n)$.

$$T(n) = \begin{cases} 1 & \text{if } n = 0 \text{ or } 1 \\ T(n-1) + T(n-2) + 1 & \text{if } n > 1 \end{cases}$$

- $T(n-1)$: time to calculate $F(n-1)$,
- $T(n-2)$: time to calculate $F(n-2)$,
- plus 1: time to add $F(n-1)$ and $F(n-2)$
- When n is 0 or 1, there is no need to divide and the only time needed is to return the number 1.

A recurrence is an equation or inequality that describes a function in terms of its value on smaller inputs.

To solve a recurrence is to derive asymptotic bounds on the solution

Recurrence for Merge Sort

O(1)

Let $T(n)$ denote the time complexity of running merge sort on n numbers.

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2 \times T\left(\frac{n}{2}\right) + n & \text{if } n > 1 \end{cases}$$

- $T(\frac{n}{2})$: time to recursively sort one half
- 2 times : there are two halves
- plus n: to merge two sorted halves
- When n is 1, there is no need to divide and the only time needed is to return the number itself

Recurrence for Finding Sum/Max/Min

Let $T(n)$ denote the time complexity of finding sum/max/min on n numbers.

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2 \times T\left(\frac{n}{2}\right) + 1 & \text{if } n > 1 \end{cases}$$

- $T(\frac{n}{2})$: time to recursively search one half
- 2 times : there are two halves
- plus 1: only one addition or one comparison to combine the two answers
- When n is 1, there is no need to divide and the only time needed is to return the number itself

Solving Recurrence

Solving recurrence for calculating Fibonacci numbers

Suppose n is even.

$$T(n) = T(n-1) + T(n-2) + 1$$

$$T(n-2) + T(n-3) + 1$$

Solving recurrence for calculating Fibonacci numbers

Suppose n is even.

$$\begin{aligned} T(n) &= T(n-1) + T(n-2) + 1 \\ &= (T(n-2) + T(n-3) + 1) + T(n-2) + 1 \end{aligned}$$

Solving recurrence for calculating Fibonacci numbers

Suppose n is even.

$$
\begin{aligned}
T(n) &= T(n-1) + T(n-2) + 1 \\
&= (T(n-2) + \underbrace{T(n-3) + 1)}_{} + T(n-2) \underbrace{+ 1}_{} \\
&> 2 \times T(n-2)
\end{aligned}
$$

ignore

by >

Solving recurrence for calculating Fibonacci numbers

Suppose n is even.

$$
\begin{aligned}
T(n) &= T(n-1) + T(n-2) + 1 \\
&= (T(n-2) + T(n-3) + 1) + T(n-2) + 1 \\
&> 2 \times T(n-2) \\
&> 2 \times (2 \times T(n-4))
\end{aligned}
$$

Solving recurrence for calculating Fibonacci numbers

Suppose n is even.

$$
\begin{aligned}
T(n) &= T(n-1) + T(n-2) + 1 \\
&= (T(n-2) + T(n-3) + 1) + T(n-2) + 1 \\
&> 2 \times T(n-2) \\
&> 2 \times (2 \times T(n-4)) = 2^2 \times T(n-4)
\end{aligned}
$$

Solving recurrence for calculating Fibonacci numbers

Suppose n is even.

$$
\begin{aligned}
T(n) &= T(n-1) + T(n-2) + 1 \\
&= (T(n-2) + T(n-3) + 1) + T(n-2) + 1 \\
&> 2 \times T(n-2) \\
&> 2 \times (2 \times T(n-4)) = 2^2 \times T(n-4) \\
&> 2^2 \times (2 \times T(n-6))
\end{aligned}
$$

Solving recurrence for calculating Fibonacci numbers

Suppose n is even.

$$
\begin{aligned}
T(n) &= T(n-1) + T(n-2) + 1 \\
&= (T(n-2) + T(n-3) + 1) + T(n-2) + 1 \\
&> 2 \times T(n-2) \\
&> 2 \times (2 \times T(n-4)) = 2^2 \times T(n-4) \\
&> 2^2 \times (2 \times T(n-6)) = 2^3 \times T(n-6)
\end{aligned}
$$

$2^4 \times T(n-8)$

$2^5 \times T(n-10)$ $\qquad k = \frac{n}{2}$

$2^k \times T(n-2k) \ldots$ $\qquad n - 2 \times \frac{n}{2} = 0$

Solving recurrence for calculating Fibonacci numbers

Suppose n is even.

$$
\begin{aligned}
T(n) &= T(n-1) + T(n-2) + 1 \\
&= (T(n-2) + T(n-3) + 1) + T(n-2) + 1 \\
&> 2 \times T(n-2) \\
&> 2 \times (2 \times T(n-4)) = 2^2 \times T(n-4) \\
&> 2^2 \times (2 \times T(n-6)) = 2^3 \times T(n-6) \\
&> 2^3 \times (2 \times T(n-8))
\end{aligned}
$$

Solving recurrence for calculating Fibonacci numbers

Suppose n is even.

$$
\begin{aligned}
T(n) &= T(n-1) + T(n-2) + 1 \\
&= (T(n-2) + T(n-3) + 1) + T(n-2) + 1 \\
&> 2 \times T(n-2) \\
&> 2 \times (2 \times T(n-4)) = 2^2 \times T(n-4) \\
&> 2^2 \times (2 \times T(n-6)) = 2^3 \times T(n-6) \\
&> 2^3 \times (2 \times T(n-8)) = 2^4 \times T(n-2 \times 4)
\end{aligned}
$$

Solving recurrence for calculating Fibonacci numbers

Suppose n is even.

$$
\begin{aligned}
T(n) &= T(n-1) + T(n-2) + 1 \\
&= (T(n-2) + T(n-3) + 1) + T(n-2) + 1 \\
&> 2 \times T(n-2) \\
&> 2 \times (2 \times T(n-4)) = 2^2 \times T(n-4) \\
&> 2^2 \times (2 \times T(n-6)) = 2^3 \times T(n-6) \\
&> 2^3 \times (2 \times T(n-8)) = 2^4 \times T(n-2 \times 4) \\
&\vdots \\
&> 2^k \times T(n-2k) \\
&\vdots
\end{aligned}
$$

Solving recurrence for calculating Fibonacci numbers

Suppose n is even.

$$
\begin{aligned}
T(n) &= T(n-1) + T(n-2) + 1 \\
&= (T(n-2) + T(n-3) + 1) + T(n-2) + 1 \\
&> 2 \times T(n-2) \\
&> 2 \times (2 \times T(n-4)) = 2^2 \times T(n-4) \\
&> 2^2 \times (2 \times T(n-6)) = 2^3 \times T(n-6) \\
&> 2^3 \times (2 \times T(n-8)) = 2^4 \times T(n-2 \times 4) \\
&\vdots \\
&> 2^k \times T(n-2k) \\
&\vdots \\
&> 2^{\frac{n}{2}} \times T(0)
\end{aligned}
$$

Solving recurrence for calculating Fibonacci numbers

Suppose n is even.

$$
\begin{aligned}
T(n) &= T(n-1) + T(n-2) + 1 \\
&= (T(n-2) + T(n-3) + 1) + T(n-2) + 1 \\
&> 2 \times T(n-2) \\
&> 2 \times (2 \times T(n-4)) = 2^2 \times T(n-4) \\
&> 2^2 \times (2 \times T(n-6)) = 2^3 \times T(n-6) \\
&> 2^3 \times (2 \times T(n-8)) = 2^4 \times T(n-2 \times 4) \\
&\vdots \\
&> 2^k \times T(n-2k) \\
&\vdots \\
&> 2^{\frac{n}{2}} \times T(0) \\
&= 2^{\frac{n}{2}} \qquad \texttt{because T(0) is 1}
\end{aligned}
$$

Solving recurrence for calculating Fibonacci numbers

Suppose n is even.

$$
\begin{aligned}
T(n) &= T(n-1) + T(n-2) + 1 \\
&= (T(n-2) + T(n-3) + 1) + T(n-2) + 1 \\
&> 2 \times T(n-2) \\
&> 2 \times (2 \times T(n-4)) = 2^2 \times T(n-4) \\
&> 2^2 \times (2 \times T(n-6)) = 2^3 \times T(n-6) \\
&> 2^3 \times (2 \times T(n-8)) = 2^4 \times T(n-2 \times 4) \\
&\vdots \\
&> 2^k \times T(n-2k) \\
&\vdots \\
&> 2^{\frac{n}{2}} \times T(0) \\
&= 2^{\frac{n}{2}}
\end{aligned}
$$

Suppose n is odd. $T(n) > 2^{\frac{n-1}{2}} \times T(1)$

Solving recurrence for calculating Fibonacci numbers

Suppose n is even.

$$\begin{aligned}
T(n) &= T(n-1) + T(n-2) + 1 \\
&= (T(n-2) + T(n-3) + 1) + T(n-2) + 1 \\
&> 2 \times T(n-2) \\
&> 2 \times (2 \times T(n-4)) = 2^2 \times T(n-4) \\
&> 2^2 \times (2 \times T(n-6)) = 2^3 \times T(n-6) \\
&> 2^3 \times (2 \times T(n-8)) = 2^4 \times T(n-2 \times 4) \\
&\vdots \\
&> 2^k \times T(n-2k) \\
&\vdots \\
&> 2^{\frac{n}{2}} \times T(0) \\
&= 2^{\frac{n}{2}}
\end{aligned}$$

Suppose n is odd. $T(n) > 2^{\frac{n-1}{2}} \times T(1) = 2^{\frac{n-1}{2}}$

Solving recurrence for calculating Fibonacci numbers

Suppose n is even.

$$
\begin{aligned}
T(n) &= T(n-1) + T(n-2) + 1 \\
&= (T(n-2) + T(n-3) + 1) + T(n-2) + 1 \\
&> 2 \times T(n-2) \\
&> 2 \times (2 \times T(n-4)) = 2^2 \times T(n-4) \\
&> 2^2 \times (2 \times T(n-6)) = 2^3 \times T(n-6) \\
&> 2^3 \times (2 \times T(n-8)) = 2^4 \times T(n-2 \times 4) \\
&\vdots \\
&> 2^k \times T(n-2k) \\
&\vdots \\
&> 2^{\frac{n}{2}} \times T(0) \\
&= 2^{\frac{n}{2}}
\end{aligned}
$$

Suppose n is odd. $\quad T(n) > 2^{\frac{n-1}{2}} \times T(1) = 2^{\frac{n-1}{2}}$

exponential!

Solving recurrence for Merge Sort

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2 \times T\left(\frac{n}{2}\right) + n & \text{if } n > 1 \end{cases}$$

Solving the recurrence

$$T(n) \quad = \quad 2 \times T\left(\frac{n}{2}\right) + n$$

Solving recurrence for Merge Sort

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2 \times T\left(\frac{n}{2}\right) + n & \text{if } n > 1 \end{cases}$$

Solving the recurrence

$$\begin{aligned} T(n) &= 2 \times T\left(\frac{n}{2}\right) + n \\ &= 2 \times \left(2 \times T\left(\frac{n}{2^2}\right) + \frac{n}{2}\right) + n \end{aligned}$$

Solving recurrence for Merge Sort

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2 \times T\left(\frac{n}{2}\right) + n & \text{if } n > 1 \end{cases}$$

Solving the recurrence

$$\begin{aligned} T(n) &= 2 \times T\left(\frac{n}{2}\right) + n \\ &= 2 \times \left(2 \times T\left(\frac{n}{2^2}\right) + \frac{n}{2}\right) + n = 2^2 \times T\left(\frac{n}{2^2}\right) + 2n \end{aligned}$$

Solving recurrence for Merge Sort

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2 \times T\left(\frac{n}{2}\right) + n & \text{if } n > 1 \end{cases}$$

Solving the recurrence

$$\begin{aligned} T(n) &= 2 \times T\left(\frac{n}{2}\right) + n \\ &= 2 \times (2 \times T\left(\frac{n}{2^2}\right) + \frac{n}{2}) + n = 2^2 \times T\left(\frac{n}{2^2}\right) + 2n \\ &= 2^2 \times (2 \times T\left(\frac{n}{2^3}\right) + \frac{n}{2^2}) + 2n \end{aligned}$$

Solving recurrence for Merge Sort

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2 \times T\left(\frac{n}{2}\right) + n & \text{if } n > 1 \end{cases}$$

Solving the recurrence

$$\begin{aligned} T(n) &= 2 \times T\left(\frac{n}{2}\right) + n \\ &= 2 \times \left(2 \times T\left(\frac{n}{2^2}\right) + \frac{n}{2}\right) + n = 2^2 \times T\left(\frac{n}{2^2}\right) + 2n \\ &= 2^2 \times \left(2 \times T\left(\frac{n}{2^3}\right) + \frac{n}{2^2}\right) + 2n = 2^3 \times T\left(\frac{n}{2^3}\right) + 3n \\ &\vdots \end{aligned}$$

Solving recurrence for Merge Sort

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2 \times T\left(\frac{n}{2}\right) + n & \text{if } n > 1 \end{cases}$$

Solving the recurrence

$$\begin{aligned} T(n) &= 2 \times T\left(\frac{n}{2}\right) + n \\ &= 2 \times (2 \times T\left(\frac{n}{2^2}\right) + \frac{n}{2}) + n = 2^2 \times T\left(\frac{n}{2^2}\right) + 2n \\ &= 2^2 \times (2 \times T\left(\frac{n}{2^3}\right) + \frac{n}{2^2}) + 2n = 2^3 \times T\left(\frac{n}{2^3}\right) + 3n \\ &\vdots \\ &= 2^{\log n} \times T\left(\frac{n}{2^{\log n}}\right) + (\log n) \times n \end{aligned}$$

Solving recurrence for Merge Sort

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2 \times T\left(\frac{n}{2}\right) + n & \text{if } n > 1 \end{cases}$$

Solving the recurrence

$$\begin{aligned}
T(n) &= 2 \times T\left(\frac{n}{2}\right) + n \\
&= 2 \times \left(2 \times T\left(\frac{n}{2^2}\right) + \frac{n}{2}\right) + n = 2^2 \times T\left(\frac{n}{2^2}\right) + 2n \\
&= 2^2 \times \left(2 \times T\left(\frac{n}{2^3}\right) + \frac{n}{2^2}\right) + 2n = 2^3 \times T\left(\frac{n}{2^3}\right) + 3n \\
&\vdots \\
&= 2^{\log n} \times T\left(\frac{n}{2^{\log n}}\right) + (\log n) \times n \\
&= 2^{\log n} \times T(1) + (\log n) \times n
\end{aligned}$$

Solving recurrence for Merge Sort

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2 \times T\left(\frac{n}{2}\right) + n & \text{if } n > 1 \end{cases}$$

Solving the recurrence

$$\begin{aligned}
T(n) &= 2 \times T\left(\frac{n}{2}\right) + n \\
&= 2 \times \left(2 \times T\left(\frac{n}{2^2}\right) + \frac{n}{2}\right) + n = 2^2 \times T\left(\frac{n}{2^2}\right) + 2n \\
&= 2^2 \times \left(2 \times T\left(\frac{n}{2^3}\right) + \frac{n}{2^2}\right) + 2n = 2^3 \times T\left(\frac{n}{2^3}\right) + 3n \\
&\vdots \\
&= 2^{\log n} \times T\left(\frac{n}{2^{\log n}}\right) + (\log n) \times n \\
&= 2^{\log n} \times T(1) + (\log n) \times n \\
&= n + n \log n = O(n \log n)
\end{aligned}$$

Solving Recurrence for Finding Sum/Max/Min

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2 \times T\left(\frac{n}{2}\right) + 1 & \text{if } n > 1 \end{cases}$$

Solving the recurrence

$$T(n) = 2 \times \color{red}{T\left(\frac{n}{2}\right)} + 1$$

Solving Recurrence for Finding Sum/Max/Min

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2 \times T\left(\frac{n}{2}\right) + 1 & \text{if } n > 1 \end{cases}$$

Solving the recurrence

$$\begin{aligned} T(n) &= 2 \times T\left(\frac{n}{2}\right) + 1 \\ &= 2 \times \left(2 \times T\left(\frac{n}{2^2}\right) + 1\right) + 1 \end{aligned}$$

Solving Recurrence for Finding Sum/Max/Min

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2 \times T\left(\frac{n}{2}\right) + 1 & \text{if } n > 1 \end{cases}$$

Solving the recurrence

$$\begin{aligned} T(n) &= 2 \times T\left(\frac{n}{2}\right) + 1 \\ &= 2 \times \left(2 \times T\left(\frac{n}{2^2}\right) + 1\right) + 1 = 2^2 \times T\left(\frac{n}{2^2}\right) + (2^2 - 1) \end{aligned}$$

Solving Recurrence for Finding Sum/Max/Min

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2 \times T\left(\frac{n}{2}\right) + 1 & \text{if } n > 1 \end{cases}$$

Solving the recurrence

$$\begin{aligned} T(n) &= 2 \times T\left(\frac{n}{2}\right) + 1 \\ &= 2 \times \left(2 \times T\left(\frac{n}{2^2}\right) + 1\right) + 1 = 2^2 \times T\left(\frac{n}{2^2}\right) + (2^2 - 1) \\ &= 2^2 \times \left(2 \times T\left(\frac{n}{2^3}\right) + 1\right) + (2^2 - 1) \end{aligned}$$

Solving Recurrence for Finding Sum/Max/Min

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2 \times T\left(\frac{n}{2}\right) + 1 & \text{if } n > 1 \end{cases}$$

Solving the recurrence

$$\begin{aligned} T(n) &= 2 \times T\left(\frac{n}{2}\right) + 1 \\ &= 2 \times \left(2 \times T\left(\frac{n}{2^2}\right) + 1\right) + 1 = 2^2 \times T\left(\frac{n}{2^2}\right) + (2^2 - 1) \\ &= 2^2 \times \left(2 \times T\left(\frac{n}{2^3}\right) + 1\right) + (2^2 - 1) = 2^3 \times T\left(\frac{n}{2^3}\right) + (2^3 - 1) \\ &\vdots \end{aligned}$$

Solving Recurrence for Finding Sum/Max/Min

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2 \times T\left(\frac{n}{2}\right) + 1 & \text{if } n > 1 \end{cases}$$

Solving the recurrence

$$\begin{aligned}
T(n) &= 2 \times T\left(\frac{n}{2}\right) + 1 \\
&= 2 \times \left(2 \times T\left(\frac{n}{2^2}\right) + 1\right) + 1 = 2^2 \times T\left(\frac{n}{2^2}\right) + (2^2 - 1) \\
&= 2^2 \times \left(2 \times T\left(\frac{n}{2^3}\right) + 1\right) + (2^2 - 1) = 2^3 \times T\left(\frac{n}{2^3}\right) + (2^3 - 1) \\
&\vdots \\
&= 2^{\log n} \times T\left(\frac{n}{2^{\log n}}\right) + (2^{\log n} - 1)
\end{aligned}$$

Solving Recurrence for Finding Sum/Max/Min

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2 \times T\left(\frac{n}{2}\right) + 1 & \text{if } n > 1 \end{cases}$$

Solving the recurrence

$$\begin{aligned}
T(n) &= 2 \times T\left(\frac{n}{2}\right) + 1 \\
&= 2 \times \left(2 \times T\left(\frac{n}{2^2}\right) + 1\right) + 1 = 2^2 \times T\left(\frac{n}{2^2}\right) + (2^2 - 1) \\
&= 2^2 \times \left(2 \times T\left(\frac{n}{2^3}\right) + 1\right) + (2^2 - 1) = 2^3 \times T\left(\frac{n}{2^3}\right) + (2^3 - 1) \\
&\vdots \\
&= 2^{\log n} \times T\left(\frac{n}{2^{\log n}}\right) + (2^{\log n} - 1) \\
&= n \times T(1) + (n - 1)
\end{aligned}$$

Solving Recurrence for Finding Sum/Max/Min

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2 \times T\left(\frac{n}{2}\right) + 1 & \text{if } n > 1 \end{cases}$$

Solving the recurrence

$$\begin{aligned}
T(n) &= 2 \times T\left(\frac{n}{2}\right) + 1 \\
&= 2 \times \left(2 \times T\left(\frac{n}{2^2}\right) + 1\right) + 1 = 2^2 \times T\left(\frac{n}{2^2}\right) + (2^2 - 1) \\
&= 2^2 \times \left(2 \times T\left(\frac{n}{2^3}\right) + 1\right) + (2^2 - 1) = 2^3 \times T\left(\frac{n}{2^3}\right) + (2^3 - 1) \\
&\vdots \\
&= 2^{\log n} \times T\left(\frac{n}{2^{\log n}}\right) + (2^{\log n} - 1) \\
&= n \times T(1) + (n - 1) \\
&= 2n - 1 = O(n)
\end{aligned}$$

Summary of recurrence and order of growth

$T(n) = 2 \times T(\frac{n}{2}) + 1$	$T(n)$ is $O(n)$
$T(n) = 2 \times T(\frac{n}{2}) + n$	$T(n)$ is $O(n \log n)$
$T(n) = 2 \times T(n-1) + 1$	$T(n)$ is $O(2^n)$

Summary of recurrence and order of growth

$4 \times T(\frac{n}{4}) + 1$

$T(n) = 2 \times T(\frac{n}{2}) + 1$ $T(n)$ is $O(n)$

$T(n) = 2 \times T(\frac{n}{2}) + n$ $T(n)$ is $O(n \log n)$

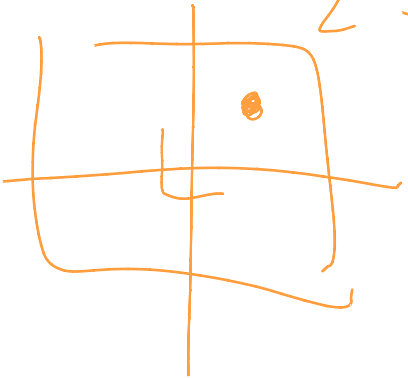
$T(n) = 2 \times T(n-1) + 1$ $T(n)$ is $O(2^n)$

`binary search` $T(n) = T(\frac{n}{2}) + 1$ $T(n)$ is $O(\log n)$

$1 \times T(\frac{n}{2})$

What about Triomino Puzzle

$2^n \to 2^{n-1}$



Summary

Summary: Fibonacci Numbers

Next: Dynamic Programming Algorithms

```
This week: lectures + tutorials
Next week: revision lectures (Tue & Thu) + tutorials
Week 12: feedback on assignment and Q&A (only Thu lecture)
```

For note taking