Space-time trade-offs

Chapter 7 and a little of 8

Space-time trade offs

Most discussion on algorithm efficiency speaks to run-time, efficiency in CPU utilization. However, especially with cloud computing, memory utilization and data exchange volume must also be considered.

Space-time trade offs

Most discussion on algorithm efficiency speaks to run-time, efficiency in CPU utilization. However, especially with cloud computing, memory utilization and data exchange volume must also be considered.

A tradeoff is a situation where one thing increases and another thing decreases. It is a way to solve a problem in:

- Either in less time and by using more space, or
- In very little space by spending a long amount of time.

Space-time trade offs

Here, space refers to the data storage consumed in performing a given task (RAM, HDD, etc), and time refers to the time consumed in performing a given task (computation time or response time).

Space-time trade-offs

The best Algorithm is that which helps to solve a problem that requires less space in memory and also takes less time to generate the output. But in general, it is not always possible to achieve both of these conditions at the same time.

Types of space-time trade-offs

- Compressed or Uncompressed data
- Re-Rendering or Stored images
- Smaller code or loop unrolling
- Lookup tables or Recalculation

Compressed or Uncompressed data

What is the difference with a compressed file and a uncompressed file?

Compressed or Uncompressed data

What is the difference with a compressed file and a uncompressed file?

- A uncompressed file format has a bigger file size and has better sound/video/file quality.
- A compressed file format has a smaller file size and has poor sound/video/file quality.

Compressed or Uncompressed data

What files can be compressed?

Compressed or Uncompressed data

What files can be compressed?

- Almost all files can be compressed into a smaller file size such as: a .avi compressed into a .mp3 file format so that it's made playable on an MP3 player OR a .doc file compressed into a .zip file.

Compressed or Uncompressed data

What are the effects of compressing a file?

Compressed or Uncompressed data

What are the effects of compressing a file?

- When a file is compressed, it reduces the quality of the audio or the quality of the video (moving image), it reduces the size of the file and it erases ALL unimportant data and CANNOT be recovered.

Compressed or Uncompressed data

A space–time tradeoff can be applied to the problem of **data storage**. If data is stored uncompressed, it takes more space but access takes less time than if the data were stored compressed (since compressing the data reduces the amount of space it takes, but it takes time to run the decompression algorithm).

Compressed or Uncompressed data

A space–time tradeoff can be applied to the problem of **data storage**. If data is stored uncompressed, it takes more space but access takes less time than if the data were stored compressed (since compressing the data reduces the amount of space it takes, but it takes time to run the decompression algorithm).

Depending on the particular instance of the problem, either way is practical. There are also rare instances where it is possible to directly work with compressed data, such as in the case of compressed bitmap indices, where it is faster to work with compression than without compression.

Compressed or Uncompressed data

https://www.youtube.com/watch?v=guo8if4Yxhw

Lossy vs Lossless

What is the difference between lossless and lossy compression?

Lossy vs Lossless

What is the difference between lossless and lossy compression?

Lossless compression is a class of data compression algorithms that allows the original data to be perfectly reconstructed from the compressed data.

Lossy compression, on the other hand, uses inexact approximations by discarding some data from the original file, making it an irreversible compression method.

Lossy	Lossless
Lossy compression is the strategy of extracting data that is not available.	Lossless Compression does not delete information that is not visible.
In Lossy compression, a file in its original state is not preserved or reconstructed.	A file can be returned to its original state when it is in Lossless Compression.
In lossy compression, the accuracy of data is impaired.	Lossless Compression doesn’t sacrifice the accuracy of the content.
The scale of data is reduced by lossy compression.	The size of data is not limited by Lossless Compression.
In Images, Audio , Video, Lossy Compression can be used.	In text, images and sound, Lossless Compression can be used.
Lossy compression provides more potential for data-holding.	Lossless Compression has less ability to retain data
Irreversible compression is often referred to as lossy compression.	Reversible compression is also known as Lossless Compression.

Lossy

Advantages:

It is supported with very limited file sizes and a tone of facilities, plugins, and applications.

Disadvantages:

With a higher compression ratio, output degrades. Unable to bring back the original after compressing

Lossless

Advantages:

No content reduction, minor reductions in the size of image files.

Disadvantages:

Bigger files than if lossy compression were to be used

Re-Rendering or Stored images

Storing only the source of a vector image and rendering it as a bitmap image every time the page is requested would be trading time for space; more time used, but less space.

Rendering the image when the page is changed and storing the rendered images would be trading space for time; more space used, but less time.

This technique is more generally known as caching.

Caching simply explained

https://www.youtube.com/watch?v=6FyXURRVmR0

Smaller code or loop unrolling

Larger code size can be traded for higher program speed when applying loop unrolling. This technique makes the code longer for each iteration of a loop, but saves the computation time required for jumping back to the beginning of the loop at the end of each iteration.

Smaller code or loop unrolling

Loop unrolling is a loop transformation technique that helps to optimize the execution time of a program. We basically remove or reduce iterations. Loop unrolling increases the program’s speed by eliminating loop control instruction and loop test instructions.

Smaller code or loop unrolling

```
// This program does not use loop unrolling.
#include<stdio.h>
        int main(void)
{
                for (int i=0; i<5; i++)
                   printf("Hello\n"); //print hello 5 times
                return 0;
}
```

Smaller code or loop unrolling

```
// This program uses loop unrolling.

#include<stdio.h>

        int main(void)

{

        // unrolled the for loop in program 1

        printf("Hello\n");

        printf("Hello\n");

         printf("Hello\n");

         printf("Hello\n");

         printf("Hello\n");

                return 0;

}
```

Smaller code or loop unrolling

```
// This program uses loop unrolling.

#include<stdio.h>

        int main(void)

{

        // unrolled the for loop in program 1

        printf("Hello\n");

        printf("Hello\n");

         printf("Hello\n");

         printf("Hello\n");

         printf("Hello\n");

                return 0;

}
```

Output:

```
Hello
Hello
Hello
Hello
Hello
```

Smaller code or loop unrolling

Program 2 is more efficient than program 1 because in program 1 there is a need to check the value of i and increment the value of i every time round the loop. So small loops like this or loops where there is fixed number of iterations are involved can be unrolled completely to reduce the loop overhead.

Advantages of loop unrolling

- Increases program efficiency.
- Reduces loop overhead.
- If statements in loop are not dependent on each other, they can be executed in parallel.

Disadvantages of loop unrolling

- Increased program code size, which can be undesirable.
- Possible increased usage of register in a single iteration to store temporary variables which may reduce performance.
- Apart from very small and simple codes, unrolled loops that contain branches are even slower than recursions.

Smaller code or loop unrolling

https://www.youtube.com/watch?v=AKYuP3vpdlg

https://www.youtube.com/watch?v=6_5Dfak8qfw

Lookup Tables or Recalculation

The most common situation is an algorithm involving a lookup table: an implementation can include the entire table, which reduces computing time, but increases the amount of memory needed, or it can compute table entries as needed, increasing computing time, but reducing memory requirements.

Lookup Tables or Recalculation

The answers to some questions for every possible value can be written down. One way of solving this problem is to write down the entire lookup table, which will let you find answers very quickly but will use a lot of space. Another way is to calculate the answers without writing down anything, which uses very little space, but might take a long time.

Define: Lookup table

A lookup table is an array of data that maps input values to output values, thereby approximating a mathematical function. Given a set of input values, a lookup operation retrieves the corresponding output values from the table.

Example: Fibonacci

In mathematical terms, the sequence Fn of the Fibonacci Numbers is defined by the recurrence relation:

Example: Fibonacci

In mathematical terms, the sequence Fn of the Fibonacci Numbers is defined by the recurrence relation:

Fn = Fn – 1 + Fn – 2,

where, F0 = 0 and F1 = 1.

The Fibonacci numbers are the numbers in the following integer sequence.

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ……..

Check your understanding

Given n = 2, F[2] =

Check your understanding

Given n = 2, F[2] = F[1] + F[0] = 0 + 1 = 1

Check your understanding

Given n = 2, F[2] = F[1] + F[0] = 0 + 1 = 1

Given n = 3, F[3] =

Check your understanding

Given n = 2, F[2] = F[1] + F[0] = 0 + 1 = 1

Given n = 3, F[3] = F[2] + F[1] = 1+ 2= 3

Define: Recursion

What is recursion?

Define: Recursion

What is recursion?

Recursion is the process in which:

- a function calls itself until the base cases are reached
- complex situations will be traced recursively and become simpler and simpler
- the whole structure of the process is tree like
- values are not stored until the final stage is reached

Example: Fibonacci

Take a couple moments to write some code for the Fibonacci sequence

Fn = Fn – 1 + Fn – 2,

where, F0 = 0 and F1 = 1.

Your code should look something like this:

```
# Function to find Nth Fibonacci term
def Fibonacci(N:int):
    # Base Case
    if (N < 2):
        return N

    # Recursively computing the term
    # using recurrence relation
    return Fibonacci(N - 1) + Fibonacci(N - 2)
```

Time Complexity

Take a moment to think of what the time complexity of this code would be?

Time Complexity

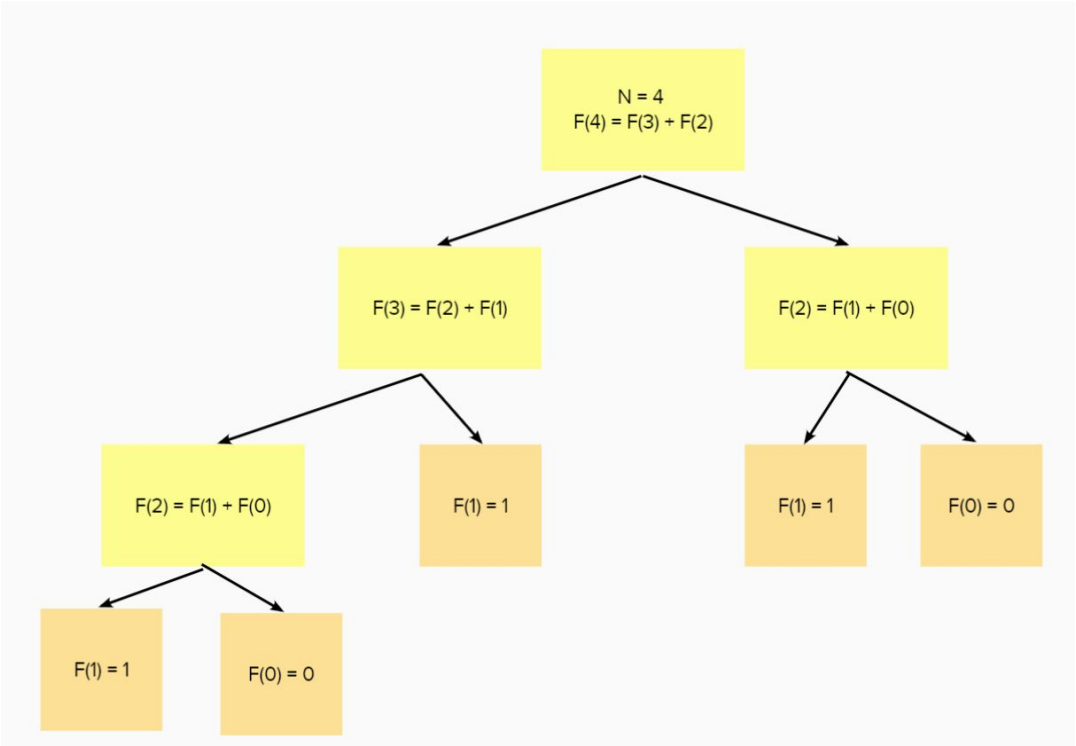
Take a moment to think of what the time complexity of this code would be?

Answer: $O(2^n)$

The time complexity of the above implementation is exponential due to multiple calculations of the same subproblems again and again. every node will split into two sub-branches.

Time Complexity

To optimize the previous approach, we can use Dynamic Programming to reduce the complexity by memoization of the overlapping subproblems as shown below in the recursion tree:



In our recursive code..

Looking at the tree on the previous slide, in our recursion code, we would start from the very top where F[4] = F[3] + F[2]. And then we would try to find the value of F[3] and F[2]. Eventually, we would reach the base case where F[0] = 0 and F[1] = 1, we could simply sum it up from the bottom to the top and obtain F[4] = 3.

Dynamic Programming

Dynamic Programming is mainly an optimization over plain recursion. Wherever we see a recursive solution that has repeated calls for same inputs, we can optimize it using Dynamic Programming.

The idea is to simply store the results of subproblems, so that we do not have to re-compute them when needed later. This simple optimization reduces time complexities from exponential to polynomial.

Dynamic Programming

For example, if we write a simple, recursive solution for Fibonacci Numbers, we get exponential time complexity and if we optimize it by storing solutions of subproblems, time complexity reduces to linear.

We already wrote some code that had exponential time complexity. Now let’s work on writing code using the idea of dynamic programming.

How to write dynamic programming script

Dynamic programming and recursion are very similar

1. Both recursion and dynamic programming are starting with the base case where we initialize the start.
2. After we wrote the base case, we will try to find any patterns followed by the problem's logic flow. Once we find it, we are basically done.
3. The main difference is that, for recursion, we do not store any intermediate values whereas dynamic programming do utilize that.

Hint!

First, code for your best case like we did in the recursive script. In this case, the base case would be F[0] = 0 and F[1] = 1. You can explicitly write these two conditions under your if statement.

Hint!

First, code for your best case like we did in the recursive script. In this case, the base case would be F[0] = 0 and F[1] = 1. You can explicitly write these two conditions under your if statement.

Next, you will need to create an empty dynamic programming array to store all the intermediate and temporary results in order for faster computing. For example, to calculate F[4], you would first calculate F[2] and F[3] and store their value in a list that you created ahead of time.

Example: Fibonacci

Take a couple moments to write some code for the Fibonacci sequence without using recursion

Your code should look something like this:

```
# Function to find Nth Fibonacci term
def Fibonacci(N):
    f=[0]*(N + 2)

    # 0th and 1st number of the
    # series are 0 and 1
    f[0] = 0
    f[1] = 1

    # Iterate over the range [2, N]
    for i in range(2,N+1) :

        # Add the previous 2 numbers
        # in the series and store it
        f[i] = f[i - 1] + f[i - 2]


    # Return Nth Fibonacci Number
    return f[N]
```

Time Complexity

Take a moment to think of what the time complexity of this code would be?

Time Complexity

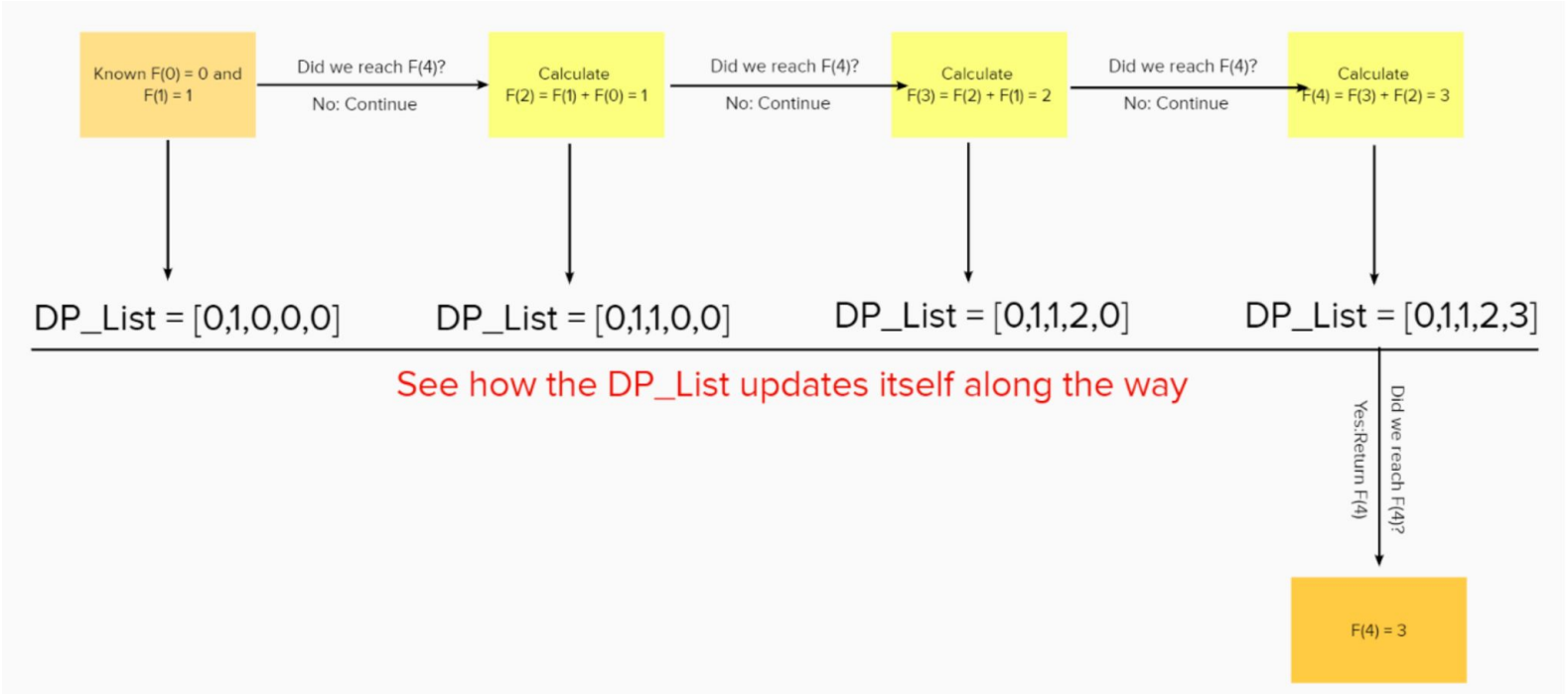
Take a moment to think of what the time complexity of this code would be?

Answer: O(n)

The time complexity of the above implementation is linear by using an auxiliary space for storing the overlapping subproblems states so that it can be used further when required. As shown on the next slide, the dynamic programming procedure chart is linear.

Time Complexity

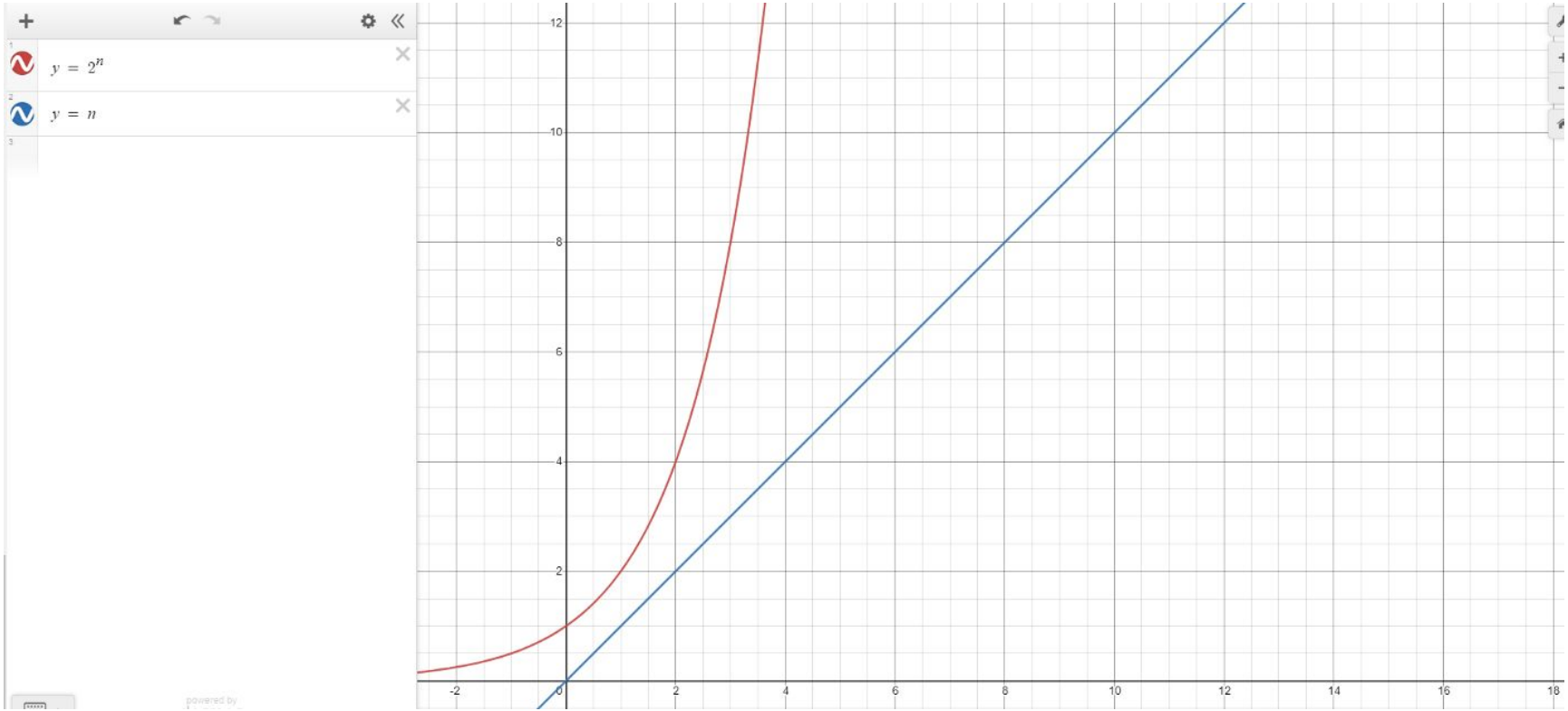
See how we reduced the complexity by using Dynamic Programming as shown below in the recursion tree:



In our non-recursive code..

We start from the very left where F[0] =0 and F[1] = 1. And then we will try to find the value of F[4], we will find the value of F[3] and F[2] first as well as store their value into the dp_list. Eventually, when we reach the right side where F[4] = 3, we can return the final result.

A visualization of the time-complexity comparison



Space complexity

Think about the space complexity for the recursive code vs the non-recursive code.

Space complexity

Think about the space complexity for the recursive code vs the non-recursive code.

For recursion, space complexity would be O(n) since the depth of the tree will be proportional to the size of n.

For dynamic programming: the space complexity would be O(n) since we need to store all intermediate values into our dp_list. So the space we need is the same as n given.

Compare

Features\Algorithm	Recursion	Dynamic Programming
Coding Logic	Base Case + Pattern	Base Case + Pattern
Store Intermediate Result?	No	Yes
Procedure Plot	Tree Like	Linear
O (Time Complexity)	O(2^n)	O(n)
O (Space Complexity)	O(n)	O(n)

Homework problem

Given n friends, each one can remain single or can be paired up with some other friend. Each friend can be paired only once. Find out the total number of ways in which friends can remain single or can be paired up.

Homework problem explanation

N = 3

Output = 4

{1} {2} {3} : all single

{1 2} {3}: 1 + 2 are paired, 3 is single

{1} {2 3}: 2 + 3 are paired, 1 is single

{1 3} {2}: 1 + 3 are paired, 2 is single

Note: {1 2} and {2 1} are the same