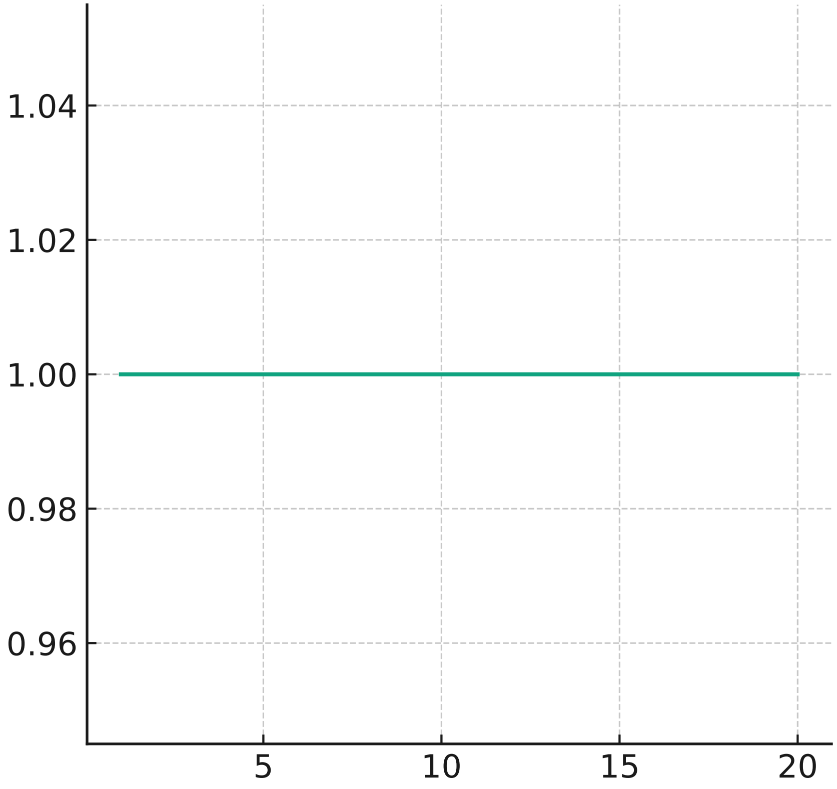
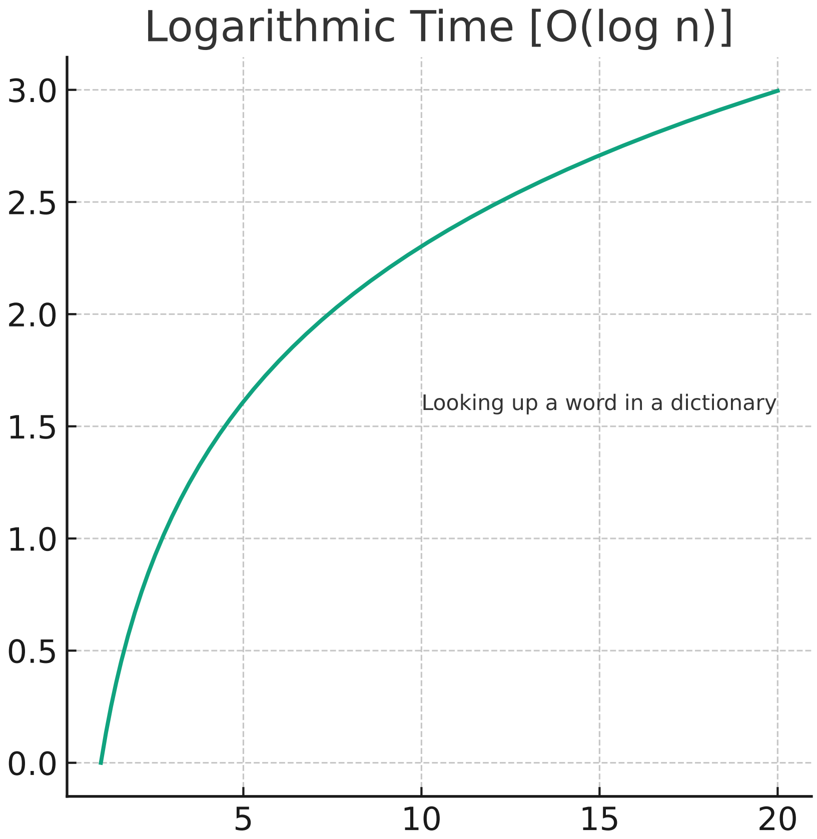
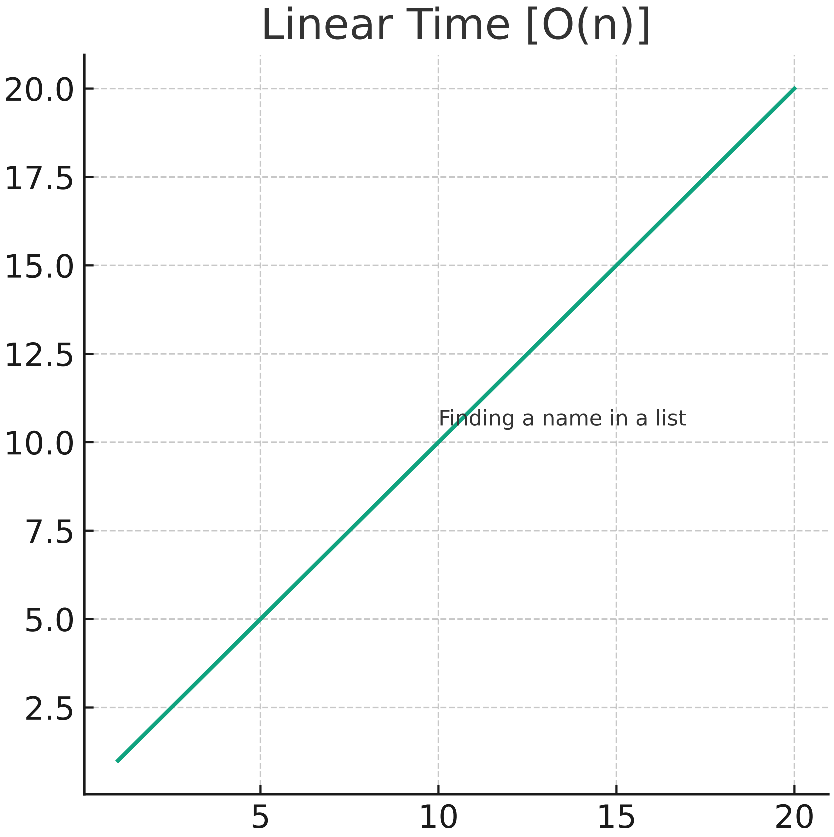
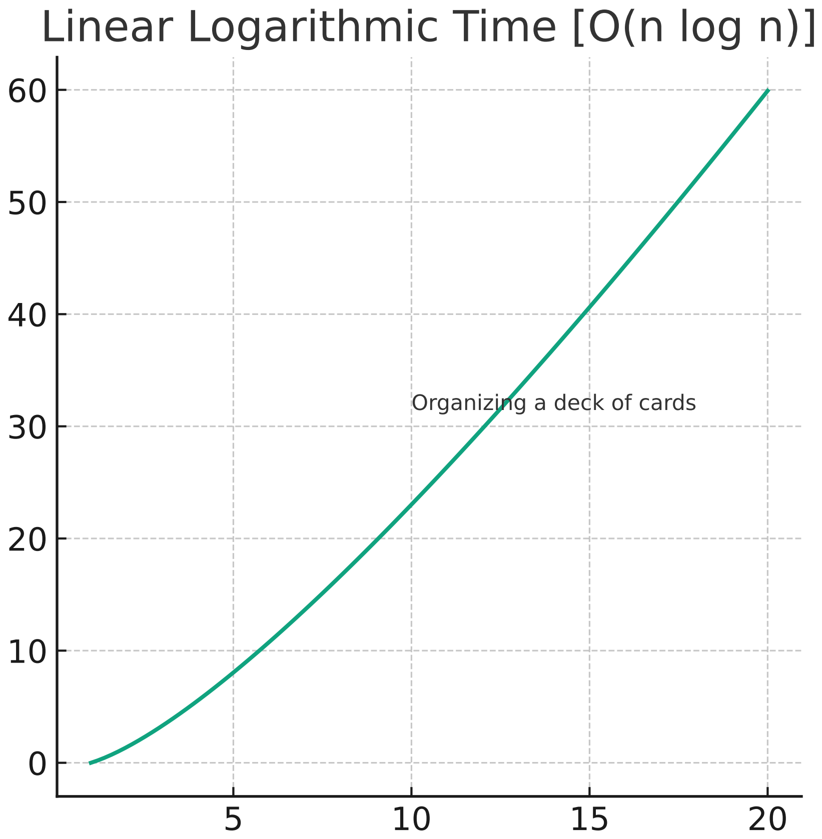
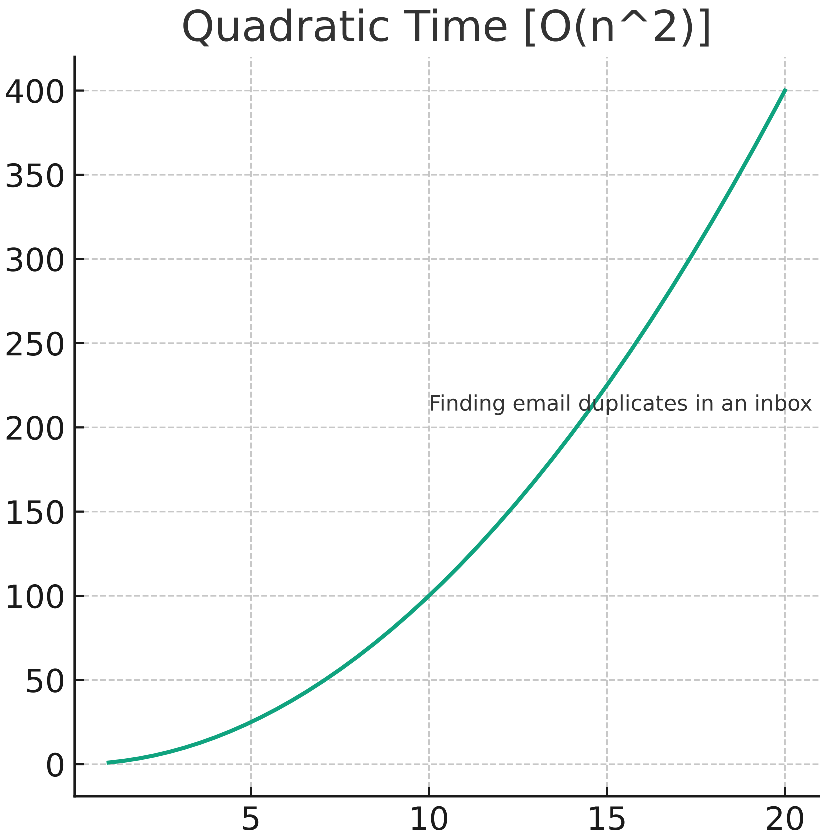
Constant Time [O(1)]

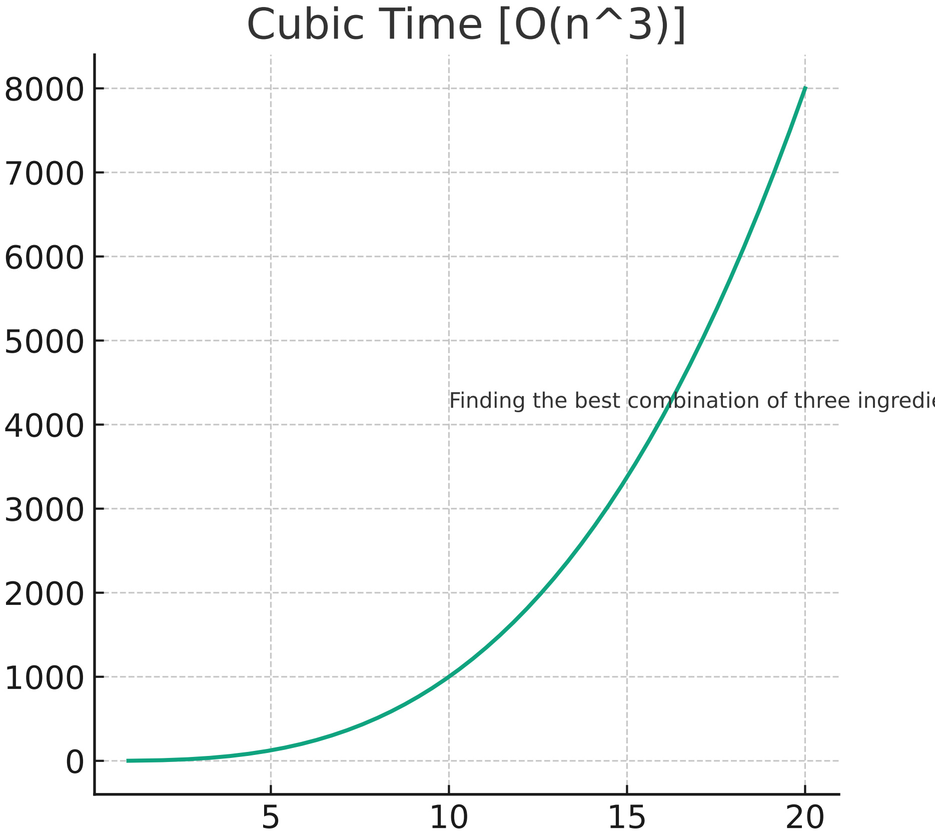


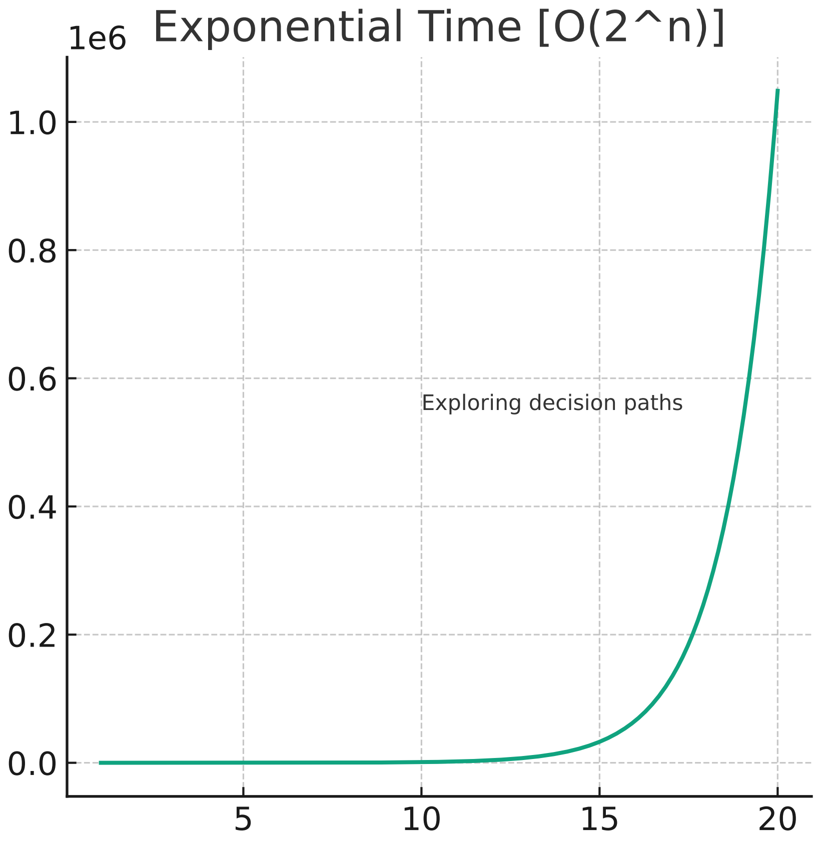

Logarithmic Time [O(log n)]
Looking up a word in a dictionary
3.0
2.5
2.0
1.5
1.0
0.5
0.0
5
10
15
20


Linear Time [O(n)]
Finding a name in a list
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
5
10
15
20


Linear Logarithmic Time [O(n log n)]
Organizing a deck of cards
0
10
20
30
40
50
60
5
10
15
20


Quadratic Time [O(n^2)]
Finding email duplicates in an inbox
400
350
300
250
200
150
100
50
0
5
10
15
20


Cubic Time [O(n^3)]
8000
7000
6000
5000
4000
3000
2000
1000
0
5
10
15
20
Finding the best combination of three ingredie



Exponential Time [O(2^n)]
1e6
1.0
0.8
0.6
0.4
0.2
0.0
5
10
15
20
Exploring decision paths

Constant Time [O(1)]

Imagine you have a book and you want to read the first page. No matter how many pages are in the book, you can directly go to the first page. That's what Constant Time is like - super quick and doesn't depend on the size of the data.

Logarithmic Time [O(log n)]

Think of looking up a word in a dictionary. You don't go through each word one by one. You open it in the middle, see if the word is before or after that point, and then repeat the process in that half of the book. This reduces the area you have to search each time quickly.

Linear Time [O(n)]

Imagine you have a list of names and you're looking for a specific one by checking each name one by one. The time it takes to find the name will depend on where it's located in the list or how long the list is.

Linear Logarithmic Time [O(n log n)]

This one is like organizing a deck of playing cards. You'd divide and conquer, but you also have to go through each card at least once. Sorting algorithms like quicksort or mergesort often have this time complexity.

Quadratic Time [O(n^2)]

Imagine you have to compare each email in your inbox with every other email to find duplicates. The more emails you have, the longer it'll take. Suppose you're organizing a round-robin tournament where each player has to play against every other player. The number of games required would be proportional to the square of the number of players.

Cubic Time [O(n^3)]

This is similar to Quadratic Time, but it grows even faster. Cubic Time algorithms are often seen in tasks involving three nested loops. Imagine you're trying to find the best combination of three different types of ingredients to make a perfect dish. You'd need to try every possible combination of all three ingredients together, which can take a lot of time as the number of different ingredients increases.

Exponential Time [O(2^n)]

These are tasks that double in complexity with each additional element. Imagine a puzzle where each piece you add doubles the number of possible combinations. Think of a decision-making process where each decision you make leads to two more decisions, and so on. The number of possible decision paths will grow exponentially.