# DSA Basics

## Some factors to consider choosing data structure?

* **What type of data needs to be stored**

It might be a possibility that a certain data structure can be the best fit for data.

* **Cost of operations:**

If we want to minimize the cost for the operations for the most frequently performed operations. For example, we have a simple list on which we must perform the search operation; then, we can create an array in which elements are stored in sorted order to perform the **binary search**. The binary search works very fast for the simple list as it divides the search space into half.

* **Memory usage**

Sometimes, we want a data structure that utilizes less memory.

* Linked list

1. **How to find if linked list has leaf?**
2. **How to find if linked list has infinite loop?**

* Recursion Imp

1. **What is recursion?**

A function or method calling itself is called recursion.

1. **What is the main disadvantage of recursive function?**

If recursive function is not implemented properly then there is chance of infinite loop

1. **What are the 2 properties of recursion or what is the guideline to write a recursive function?**

**Condition**: There should be at least one condition, so when the condition is met the function stop calls itself.

**Progressive approach:** Each time recursive call is made it should comes closer to condition.

1. **Why recursion is required Even though we can achieve same by iterating?**
2. Recursion makes program more readable.
3. Recursion is more efficient the iteration because it is fast.
4. **What is the disadvantage of recursion?**

Space complexity is more in case of recursion than iteration because each time function is called system needs to store activation record.

So, we need to consider time and space while implementing recursion.

# Algorithm Basic

## What is Time Complexity?

<https://www.geeksforgeeks.org/understanding-time-complexity-simple-examples/>

Time Complexity of algorithm/code is not equal to the actual time required to execute a particular code but the number of times a statement executes

Examples to calculate time complexity:

**O(1)**

#include <stdio.h>

int main()

{

printf("Hello World");

}

**O(N)**

#include <stdio.h>

void main()

{

int i, n = 8;

for (i = 1; i <= n; i++) {

printf("Hello Word !!!\n");

}

}

## How to Compare Algorithms?

To compare algorithms, let us define a few objective measures:

* **Execution times**: Not a good measure as execution times are specific to a particular computer.
* **A number of statements executed**: Not a good measure, since the number of statements varies with the programming language as well as the style of the individual programmer.
* **Ideal solution**: Let us assume that we express the running time of a given algorithm as a function of the input size n (i.e., f(n)) and compare these different functions corresponding to running times. This kind of comparison is independent of machine time, programming style, etc.

## What is logarithm?

<https://www.youtube.com/watch?v=M4ubFru2O80>

Log2(8) means that what we need to power 2 to get 8. That is 23

Log10(100) mean that what we need to power 10 to get to 100. That is 1010

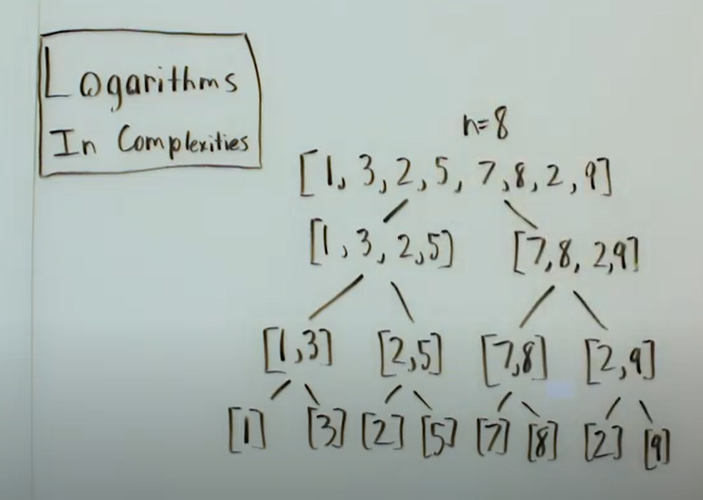
When log is used in computer science, we need to assume that it log2

When log appears in calculus, we need to assume it is log10

Say we have an array [1,3,2,5,7,8,2,9]

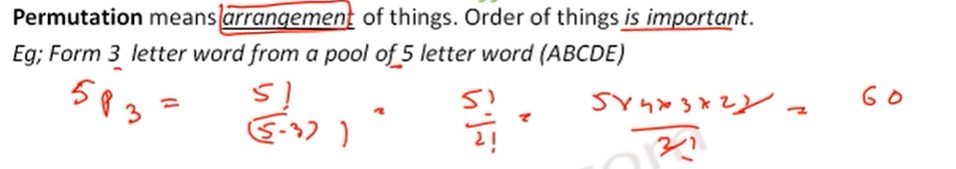
How many times we cut this in half?

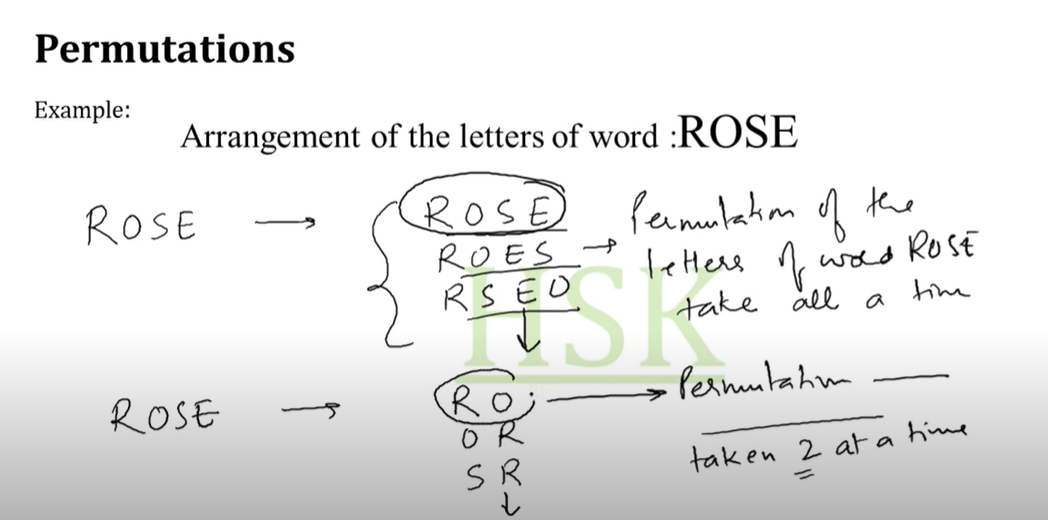
There were 8 elements and we had to cut it 3 times

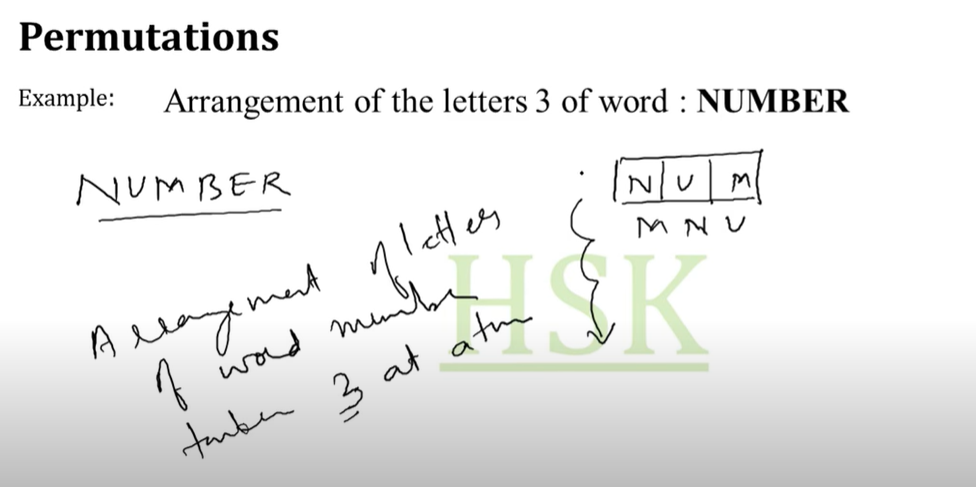


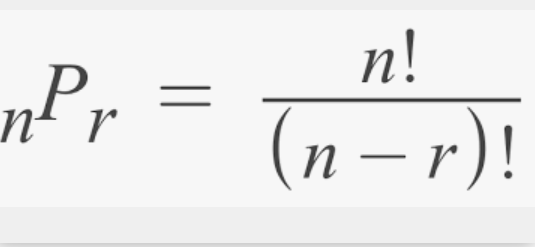
## What is permutation(ಕ್ರಮಪಲ್ಲಟನೆ)?

<https://www.youtube.com/watch?v=PLvb2jYcjeY>









## What is combination (ಒಟ್ಟುಗೊಡುವುದು)?



