#InfyTQ Data Structures Using JAVA/Python PART-2

Notes: -

Hashing & Hash Table: -

A bank wants to store and retrieve the details of its 1 million customers. Suppose the key that identifies each customer record is the customer name followed by the 11 digit account number. How many comparisons will be required to identify a customer record given the key?

Hashing is the process of transforming a set of characters (key) into a shorter fixed length integer value. This shorter fixed length integer value which represents the original set of characters (key) is known as **hash value** or **hash**. A **hash function** will be used to generate the hash value from the original set of characters (key). Various algorithms may be used to arrive at the hash function.

Suppose we have a key-value pair as shown below. Here key is the three letter abbreviation of country names and value is the corresponding country name.

Input(Key)	Value
ISR	Israel
PER	Peru
IND	India
FJI	Fiji
CAN	Canada

Suppose we want to store the country names based on its key using the below hash function, let's understand how it works.

$h(k) = \sum_{i=0}^{n} (ord(k[i])) \% 5$, where n=2 as there are 3 characters in the input key of this example and ord(k[i]) is a python function which returns the ASCII value of character at the ith position in k.

Hash values corresponding to each key can be generated as shown below:

Key(Input)	Value	Hash(Output)			
ISR	Israel	(ord("I")+ord("S")+ ord("R"))%5	(73+83+82)%5	238%5	3
PER	Peru	(ord("P")+ ord("E") + ord("R"))%5	(80+69+82)%5	231%5	1
IND	India	(ord("I")+ord("N")+ ord("D"))%5	(73+78+68)%5	219%5	4
FJI	Fiji	(ord("F")+ord("J") +ord("I"))%5	(70+74+73)%5	217%5	2
CAN	Canada	(ord("C")+ord("A") +ord("N"))%5	(67+65+78)%5	210%5	0

$h(k) = \sum_{i=0}^{n} (ord(k[i])) \% 5$, where n=2 as there are 3 characters in the input key of this example

1. Hash function will always generate the same hash value (output) for a given key (input).
2. Keys have to be unique.
3. A given key will have only one value in the key-value pair.

Suppose now we have added SWE also to the list.

Key(Input)	Value	Hash(Output)			
ISR	Israel	(ord("I")+ord("S")+ ord("R"))%5	(73+83+82)%5	238%5	3
PER	Peru	(ord("P")+ ord("E") + ord("R"))%5	(80+69+82)%5	231%5	1
IND	India	(ord("I")+ord("N")+ ord("D"))%5	(73+78+68)%5	219%5	4
FJI	Fiji	(ord("F")+ord("J") +ord("I"))%5	(70+74+73)%5	217%5	2
CAN	Canada	(ord("C")+ord("A") +ord("N"))%5	(67+65+78)%5	210%5	0
SWE	Sweden	(ord("S")+ord("W") +ord("E"))%5	(83+87+69)%5	239%5	4

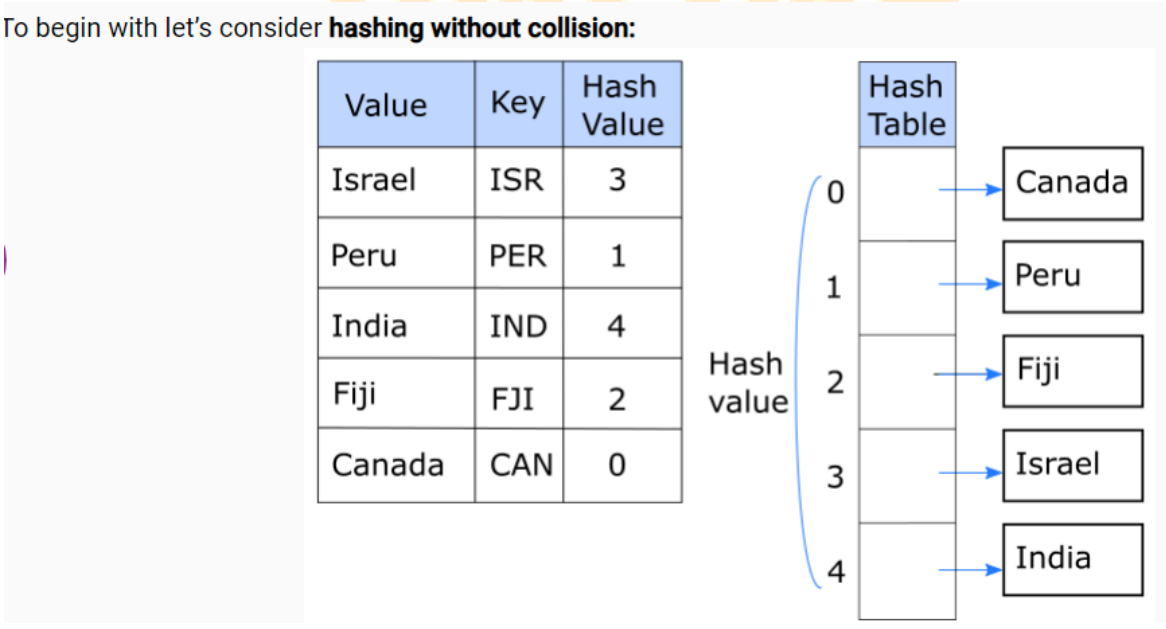
Two keys (IND and SWE) have generated the same hash value. That means hash function may compute same hash value for multiple keys and this is known as **collision** in hashing. This occurs because the number of possibilities in input (key) is much greater than the number of possibilities in the output (hash value).

In this example, three letter abbreviation exists for all the countries in the world whereas the hash value can be only between 0 – 4.

Collisions are inevitable, however number of collisions depends on the goodness of the hash function.

#Hash Table: -

Hash table is a data structure that helps to map the keys to its value. It is primarily an array which stores the reference to the actual value based on the hash. In the hash table, hash refers to its index and the number of elements in the hash table is based on the hash function. Thus hash table can be searched very quickly for the actual value based on the hash obtained from its key.



Here, we observe that since hash function takes a mod of 5, only possible values for hash are 0 to 4, hence the hash table can have only 5 elements.

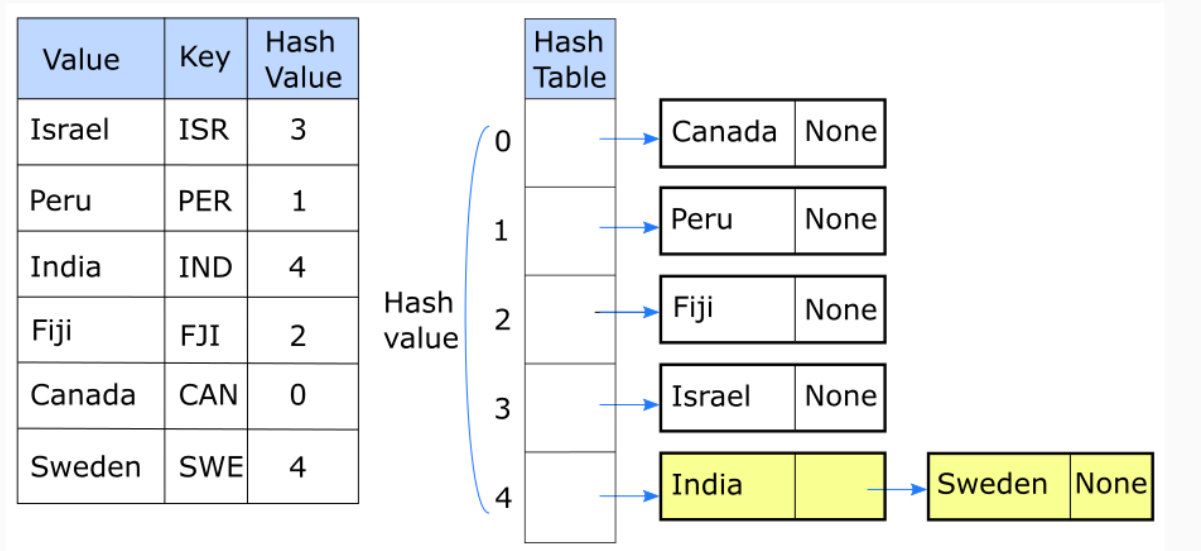
Hashing with collision:

We have already seen that hash function may compute same hash value for multiple keys resulting in collision.

Let's understand how this is handled.

One of the techniques that can be used for handling collision is known as **separate chaining**. In this case, instead of hash table containing a reference to one value, it will maintain a reference to a linked list. When more than one key maps to the same hash, its values are added as nodes to the corresponding linked list.

Observe the linked list maintained for each hash and how values are stored in case of collision (IND,SWE).



Python:

Python dictionary is implemented internally as a hash table.

`dict={key1:value1, key2:value2,….keyn:value n}` <----- **Implemented internally as hash table**

`dict.get(key1)` <---------- **Implicitly implements the get() operation of hash table**

`dict.update({keym:valuem})` <----- **Implicitly implements the put() operation of hash table**

#Algorithms: -

It is step by step process performing some operations(or actions).

It takes I/P list & Process Step by Step by step method for solving a problem & gives the o/p as a list.

There are two criteria for judging Algorithm: -

1. Correctness.
2. Efficiency

Check Diagram from Handwritten Notes: -

#Searching: - From Handwritten Notes

#Sorting: - From Handwritten Notes

1. Merge Sort:-

Merge sort is one of the most prominent divide-and-conquer sorting algorithms in the modern era. It can be used to sort the values in any traversable data structure such as a list.

It divides the input array into two halves, calls itself for the two halves, and then merges the two sorted halves. **The merge() function** is used for merging two halves. The merge(arr, l, m, r) is a key process that assumes that arr[l..m] and arr[m+1..r] are sorted and merges the two sorted sub-arrays into one

```
MergeSort(arr[], l,  r)
If r > l
     1. Find the middle point to divide the array into
two halves:
             middle m = l+ (r-l)/2
     2. Call mergeSort for first half:
             Call mergeSort(arr, l, m)
     3. Call mergeSort for second half:
             Call mergeSort(arr, m+1, r)
     4. Merge the two halves sorted in step 2 and 3:
             Call merge(arr, l, m, r)
```

#Quick Sort In Python:-

Quicksort is a popular sorting algorithm and is often used, right alongside Merge Sort. It's a good example of an efficient sorting algorithm, with an average complexity of O(nlogn)

Quicksort

The basic version of the algorithm does the following:

Divide the collection in two (roughly) equal parts by taking a pseudo-random element and using it as a pivot.

Elements smaller than the pivot get moved to the left of the pivot, and elements larger than the pivot to the right of it.

This process is repeated for the collection to the left of the pivot, as well as for the array of elements to the right of the pivot until the whole array is sorted.

#Algorithm Technique: -

1. Greedy Approach.

2. Travelling Salesman Problem

3. Dynamic Programming

From Handwritten Notes

#Analysis of Algorithm: -

Analysis of algorithms is all about estimating the amount of computing resources needed to execute a given algorithm before implementing it as a program. It is also referred as **algorithm complexity.**

The most common resource that is estimated using analysis of algorithms is the computation time i.e., how fast the algorithm performs and what affects its run time.

From Notes: -

Step Count: - Analysis of Algorithm: -

Let's find the step-count of linear search algorithm:

Assumption is that each operation takes 1 unit of time.

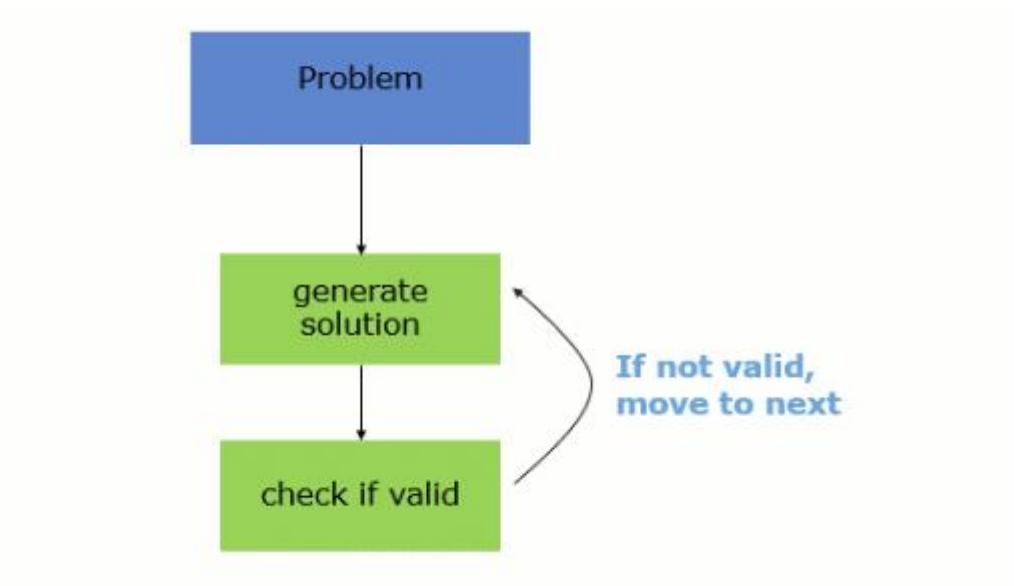
```
linear_search (list,num,n)
                                                                  Steps
1. for (i=0, i<n, i=i+1)              = 1 + (n+1) + 2(n)   = 3n+2
2.     if (num == list[i]) then       = n + n              = 2n
3.        display("Element found")    = 1                  = 1
4.        return true                 = 1                  = 1
5.     end-if                         = 0                  = 0
6. end-for                            = 0                  = 0
7. return false                       = 1                  = 1
```

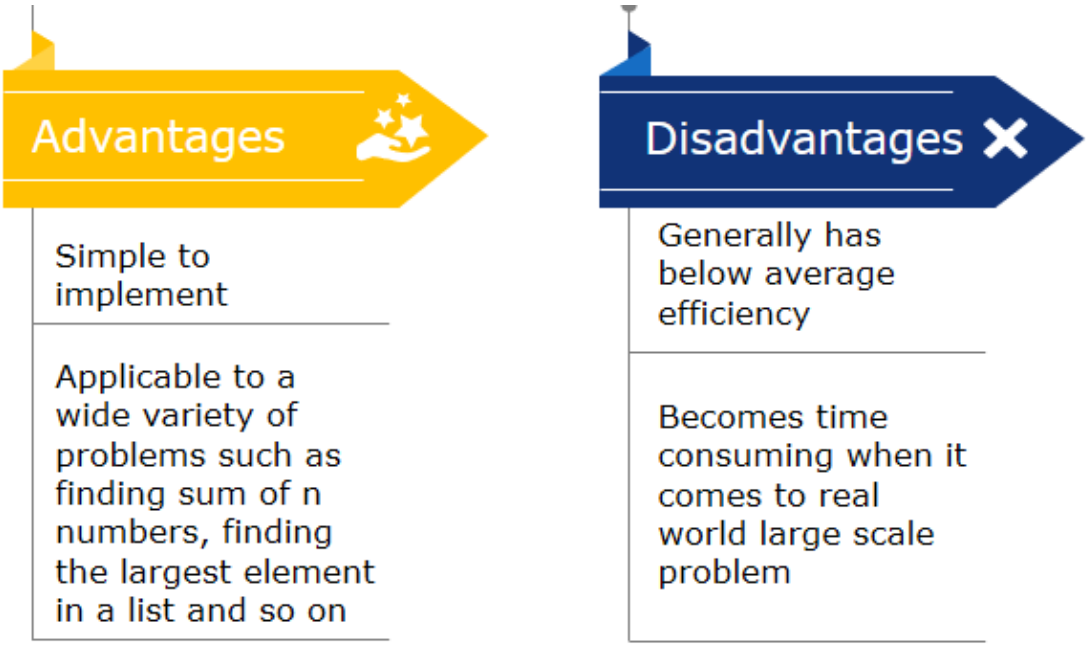
Algorithm	Time Complexity
Binary Search	O(logn)
Linear Search	O(n)
Quick Sort	$O(10^2)$
Merge Sort	O(nlogn)
Bubble Sort	$O(n^2)$
Selection Sort	$O(n^2)$
To break a numeric password of n digits (Brute-Force)	$O(10^n)$
Traveling Salesman Problem (Brute-Force	O(n!)

Brute Force Algorithm:

This is the most basic and simplest type of algorithm. A Brute Force Algorithm is the straightforward approach to a problem i.e., the first approach that comes to our mind on seeing the problem. More technically it is just like iterating every possibility available to solve that problem.
For Example: If there is a lock of **4-digit** PIN. The digits to be chosen from **0-9** then the brute force will be trying all possible combinations one by one
like **0001**, **0002**, **0003**, **0004**, and so on until we get the right PIN. In the worst case, it will take **10,000 tries** to find the right combination.

It is a straightforward approach for solving a problem. It is an exhaustive and trial-and-error technique which evaluates every possible outcome to find a solution.



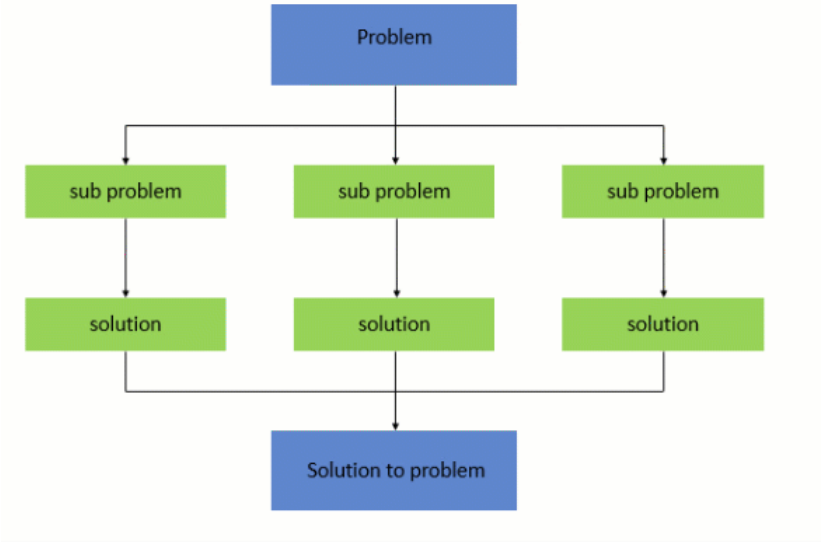


Divide & Conquer Algorithm: -

This technique involves breaking the problem into smaller parts, solving these smaller problems and then combining them to form the solution of the original problem.

It consists of the following steps:

- Divide the original problem into sub-problems
- Conquer or solve the sub-problems individually
- Combine the solutions to get the solution for the original problem



Solves difficult problems with ease

Supports parallelism and can easily be processed by parallel processing systems

Uses recursion which makes it slower

May require extra space due to usage of stacks for the sub problems

ONLINE
LEARNING