CUCKOO HASHING VS. UNIVERSAL HASHING

CUCKOO HASHING

Cuckoo hashing is a scheme in computer programming for resolving hash collisions of values of hash functions in a table, with worst-case constant lookup time. The main feature of this method is that two hash functions are used instead of one so that each element to be inserted will have two positions where it can be inserted. When its position is filled in one hash function, the existing element is displaced and inserted in the second hash table while the new one is inserted in its place.

ALGORITHM-INSERTION

```
Insert in heap1(x):
procedure
pos=func1(x)
if(heap1[pos]=empty)
        insert x in heap1[pos]
else
        prev=heap1[pos]
        insert x In heap1[pos]
        call insert in heap2 for prev
Insert in heap2(x):
procedure
pos=func2(x)
if(heap2[pos]=empty)
        insert x in heap2[pos]
else
        prev=heap2[pos]
        insert x in heap2[pos]
        call insert in heap1 for prev
```

Insertion in cuckoo hashing uses a greedy algorithm and is started by checking if the key's hash position in the first hash function is unoccupied. If it is empty, the key is inserted there. Else, the old key is replaced by the new key and the function for inserting the old key in second hash table is called. This process goes on till it enters an infinite loop or till the load factor of one of the hash tables becomes greater than 50%. In such cases, the hashing using those particular functions is inefficient and hash functions are changed and rehashing is done. The insert function for cuckoo hashing can be proved to execute in expected constant time as long as the load factor is maintained below 50%. This can be proved by creating an undirected graph with the key value as the edge weight and hash values of two hash functions as the end vertices. If this graph is a pseudoforest with at most one cycle in each of its connected components, it is said to complete in expected constant time.

ALGORITHM-LOOKUP:

```
search(x)
        if $T_1[h_1(x)]==x$ {found, return;}
        else if $T_2[h_2(x)]==x$ {found, return;}
        else {not found, return;}
```

Lookup or search function of Cuckoo Hashing is a constant time function even for worst case because all that is needed to be done is to check the corresponding positions in both the hash tables for the key to be searched for. If both these positions are empty, it is implied that the required key is not found in the hash tables.

THEORETICAL BOUNDS:

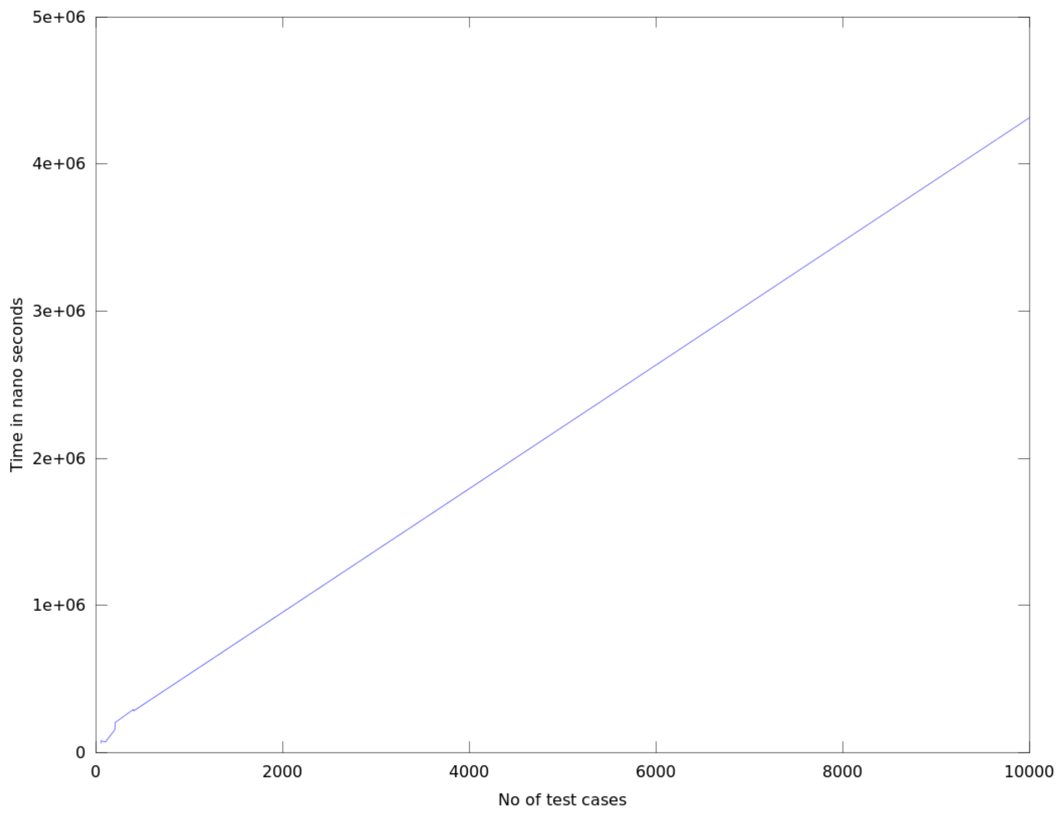
- If the hash tables have size r, we enforce that no more than r^2 insertions are performed without changing the hash functions.
- With probability $O(1/n^2)$ the hash functions have some unspecified behaviour(i.e., we should expect the worst possible).
- Otherwise, the hash functions behave exactly as if they had been picked from a $(1, n^\delta)$-universal family.

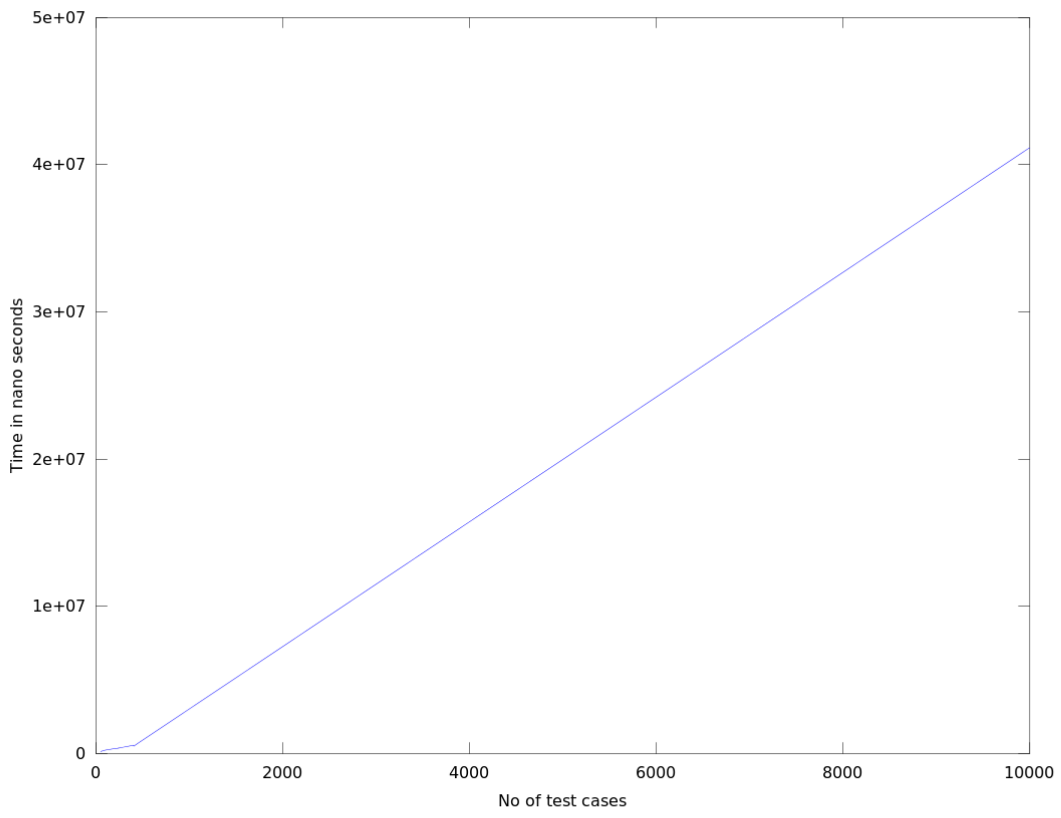
- The lookup call preceding the insertion loop ensures robustness if the key to be inserted is already in the dictionary. A slightly faster implementation can be obtained if this is known not to occur.
- WORST CASE : The worst case for lookup function occurs when a key is searched before its insertion while for insert function it occurs when an infinite loop is encountered or when number of insertions called is greater than the square of the hash table size.

DATA SETS:

No of inputs	Insert Time (ns)	Find Time(ns)
50	145755	64970
100	237027	72065
200	343111	154432
400	566244	293630
1000	41136168	4314044

GRAPHS ON THE BASIS OF DATA:





UNIVERSAL HASHING

Universal hashing is a hash algorithm that refers to a family of hash functions with a certain mathematical property and selects a function from it randomly. A low number of collisions in expectation are guaranteed even if the data is chosen by an adversary. The applications of Universal Hashing in Computer Science are varied from implementations of hash tables, randomized algorithms, and cryptography.

ALGORITHM:

```
Insert(x)
    Calculate hash value for key
    If table is empty
        Insert in table at corresponding index
    Else
        Implement linear probing
```

```
Find(x)
    Calculate hash value for x
    If present at that index
        Print “present”
    Else
        While(index<table size)
            Check if x is at index++
        If(index==table size)
            Print(not present)
```

hash function is chosen from the universal set of hash functions and the input is passed through it. The hash function used is

$$(ax+by)\%tablesize$$

(where a and b are chosen randomly)

Table size is fixed and equals 106. This ensures less number of collisions. The key is inserted in the hash table at the index calculated by the hash function. Collisions, which when occur are resolved using the Linear Probing technique. Thus table is used efficiently.

THEORETICAL BOUNDS:

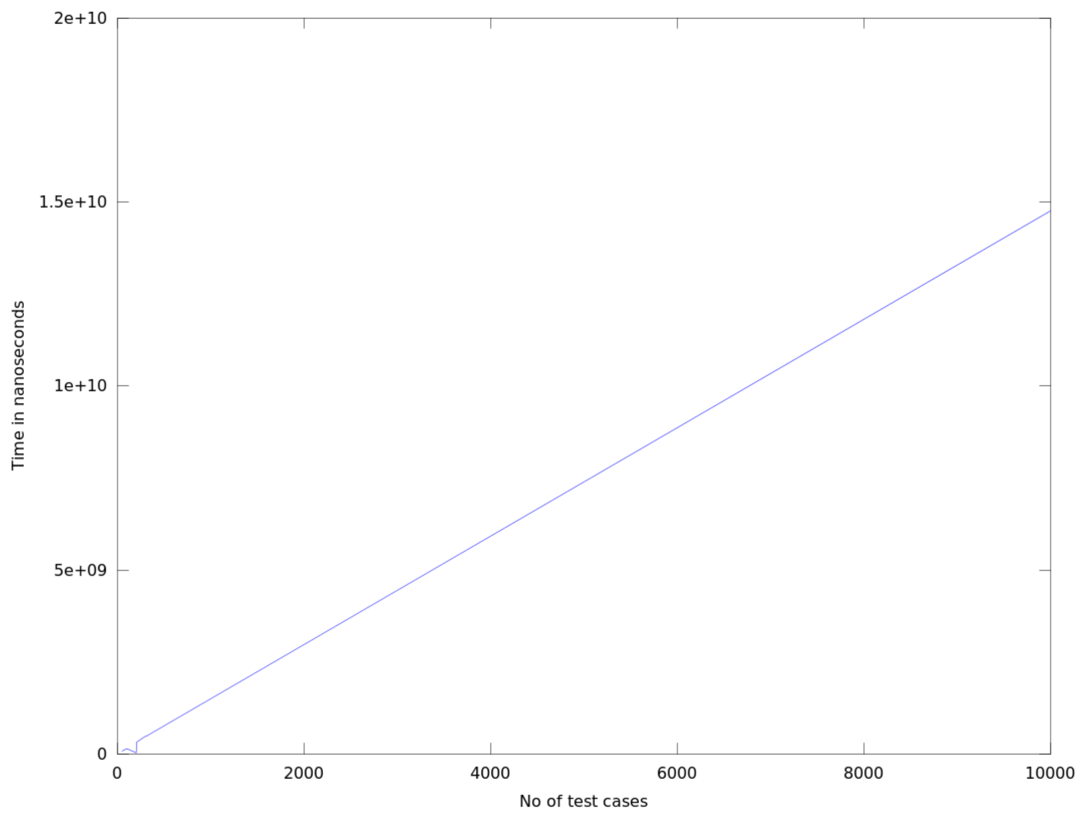
- Collisions occur only when same keys are inserted again. This gives rise to a worst case scenario.
- Similarly, when number of insertions is greater than the size of the table, collisions occur because Linear Probing is implemented.

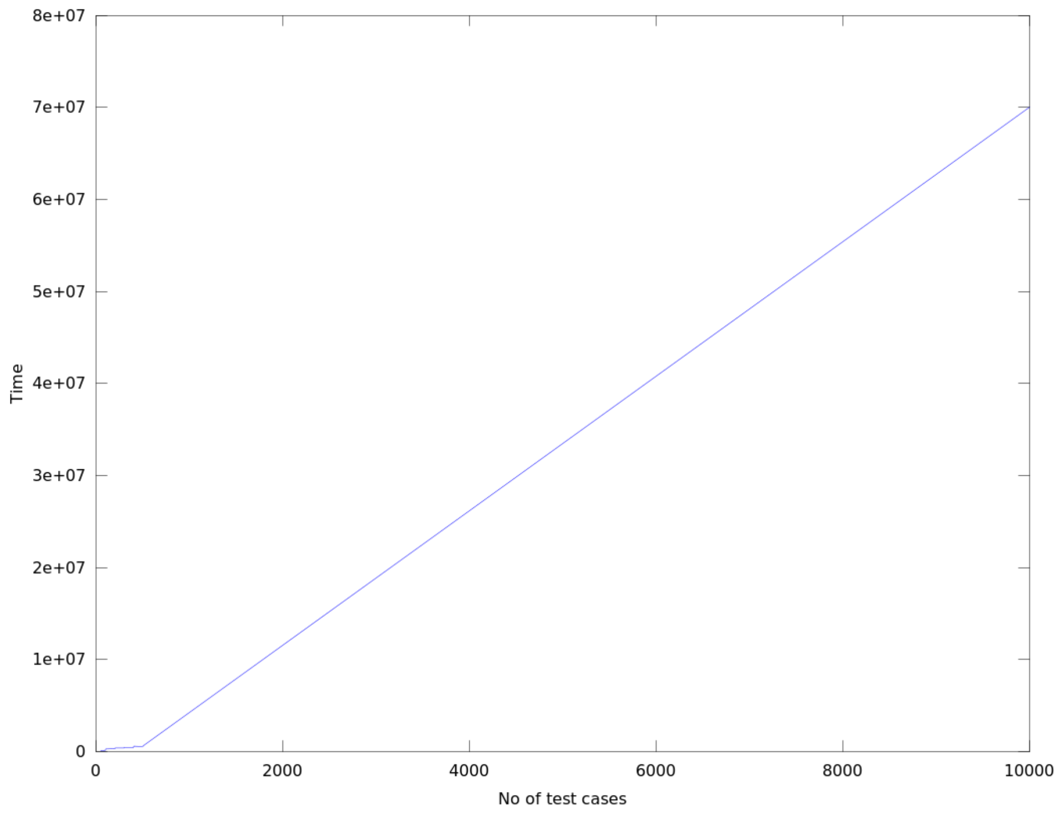
DATA SETS:

No of inputs	Insert Time(ns)	Find time(ns)
50	107242	79230749
100	151534	149286867

200	308606	336824738
400	448699	486591280
1000	7009431	14754567044

GRAPHS FOR DATA SET:





DIFFERENCES BETWEEN CUCKOO HASHING AND UNIVERSAL HASHING

- The worst case time for cuckoo hashing is O(1/n2) while that for universal hashing is O(1/n).
- Cuckoo hashing uses two hash functions while UHashing uses only one.
- Cuckoo hashing resolves collisions by rehashing while UHash does it using Linear Probing.