SEARCHING AND SORTING TECHNIQUES

DATA STRUCTURES
MODULE 4

Searching-

- Searching is a process of finding a particular element among several given elements.
- The search is successful if the required element is found.
- Otherwise, the search is unsuccessful.

Searching Algorithms

- Linear Search (**Sequential Search**.)
- Binary Search

Linear Search

- Linear Search is the simplest searching algorithm.
- It traverses the array sequentially to locate the required element.
- It searches for an element by comparing it with each element of the array one by one.
- So, it is also called as **Sequential Search**.
- **Time Complexity of Linear Search Algorithm is O(n).**

- Linear Search Algorithm is applied when-

- No information is given about the array.
- The given array is unsorted or the elements are unordered.
- The list of data items is smaller.

Linear Search Algorithm

1. Repeat For J = 1 to N
2. If (ITEM == A[J]) Then
3. Print: ITEM found at location J
4. Return
   [End of If]
   [End of For Loop]
5. If (J > N) Then
6. Print: ITEM doesn't exist
   [End of If]
7. Exit

Binary Search

- Binary Search is one of the fastest searching algorithms.
- It is used for finding the location of an element in a linear array.
- It works on the principle of divide and conquer technique.

- Binary Search Algorithm can be applied only on **Sorted arrays**.
-
- So, the elements must be arranged in-
- Either ascending order if the elements are numbers.
- Or dictionary order if the elements are strings.

- To apply binary search on an unsorted array,
- First, sort the array using some sorting technique.
- Then, use binary search algorithm.

- **Time Complexity of Binary Search Algorithm is $O(\log_2 n)$.**

Binary Search Algorithm-

- Consider-
- There is a linear array 'a' of size 'n'.
- Binary search algorithm is being used to search an element 'item' in this linear array.
- If search ends in success, it sets loc to the index of the element otherwise it sets loc to -1.
- Variables **beg** and **end** keeps track of the index of the first and last element of the array or sub array in which the element is being searched at that instant.
- Variable **mid** keeps track of the index of the middle element of that array or sub array in which the element is being searched at that instant.

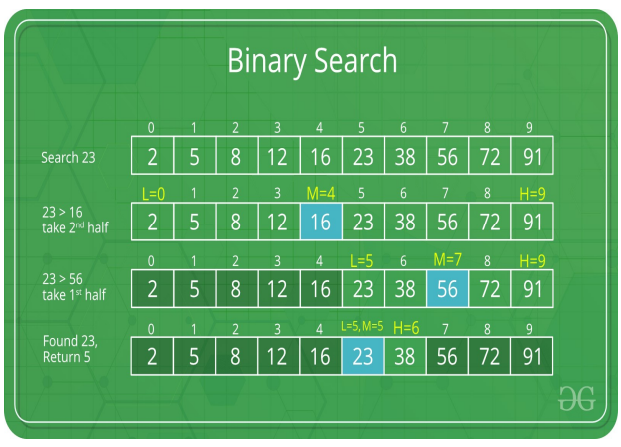
- **We basically ignore half of the elements just after one comparison**.
- Steps are

1. Compare item with the middle element.
2. If item matches with middle element, we return the mid index.
3. Else If item is greater than the mid element, then item can only lie in right half sub array after the mid element. So we recur for right half.
4. Else (item is smaller) recur for the left half.

- **Then, Binary Search Algorithm is as follows-**
-

```
1.  Begin
2.  Set beg = 0
3.  Set end = n-1
4.  Set mid = (beg + end) / 2
5.      while ( (beg <= end) and (a[mid] ≠ item) ) do
6.          if (item < a[mid]) then
7.              Set end = mid - 1
8.          else
9.              Set beg = mid + 1
10.         end if
11.         Set mid = (beg + end) / 2
12.     end while
13. if (beg > end) then
14.     Set loc = -1
15. else
16.     Set loc = mid
17. end if
18. End
```

Example



SORTING

- Sorting is ordering a list of objects.
- Sorting method can be classified into two.
- 1) Internal Sorting
- 2)External Sorting
- In internal sorting the data to be sorted is placed in main memory.
- In external sorting the data is placed in external memory such as hard disk, floppy disk etc.

Internal sorting techniques

Eg: 1)Bubble sort

2) Selection sort

3) Insertion sort

4)Quick sort

5) Heap sort

External Sorting Techniques

Eg: Merge sort

BUBBLE SORT

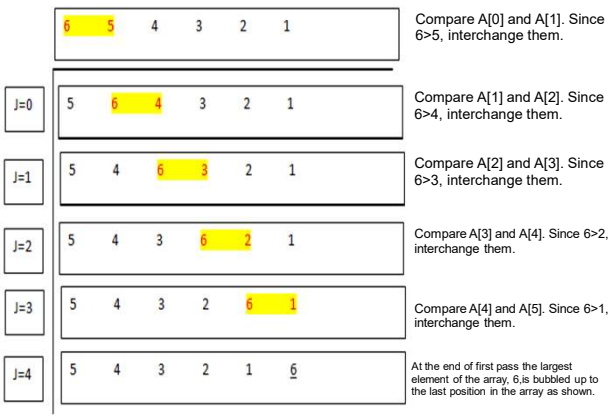
- Bubble sort, sometimes referred as sinking sort.
- It is a simple sorting algorithm that works by repeatedly stepping through the list to be sorted, comparing each pair of adjacent items and swapping them if they are in the wrong order.
- The pass through the list is repeated until no swaps are needed, which indicates that the list is sorted.
- The algorithm gets its name from the way smaller elements "bubble" to the top of the list.
- As it only uses comparisons to operate on elements, it is a comparison sort.
- Although the algorithm is simple, it is too slow for practical use, even compared to insertion sort.
- Time complexity= $O(n^2)$

ALGORITHM

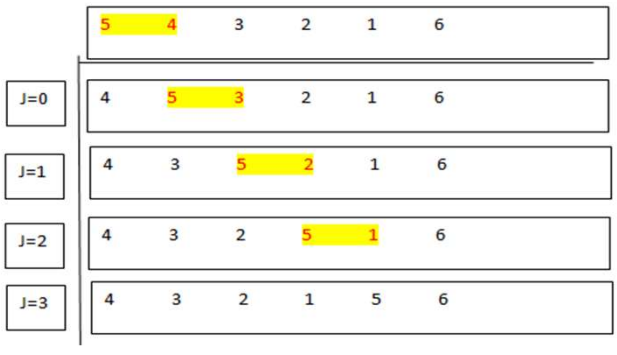
- Bubblesort(a[],n)

1. Start
2. for i ← 0 to n-1 do
   for j ← 0 to n- 1-i do
   If a[j] > a[j+1] then
   /* For decreasing order use < */
   temp ← a[j]
   a[j] ← a[j+1]
   a[j+1] ← temp

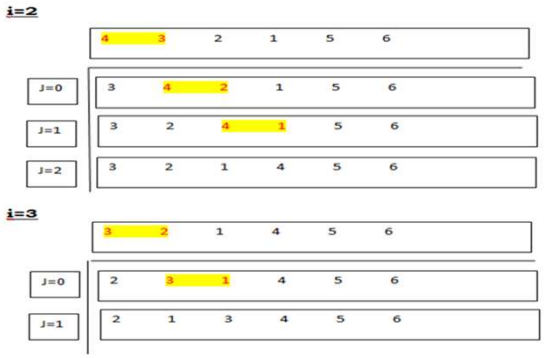
PASS 1 :i=0



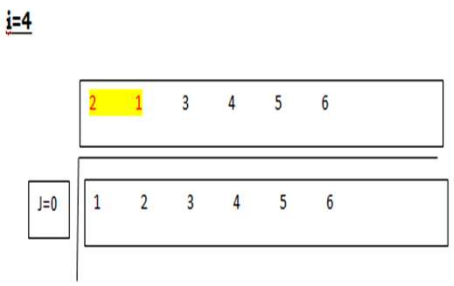
PASS 2: i=1



Pass 3 and Pass 4



Pass 5



SELECTION SORT

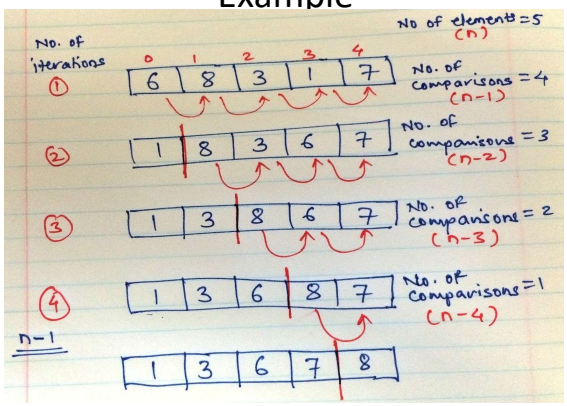
- It is an in-place comparison-based algorithm.
- Array is imaginarily divided into two parts - sorted one (left part) and unsorted one (right part).
- At the beginning, sorted part is empty, while unsorted one contains whole array.
- At every step, algorithm finds minimal element in the unsorted part and adds it to the end of the sorted one.
- When unsorted part becomes empty, algorithm stops.
- Time complexity = $O(n^2)$

ALGORITHM

- Selectionsort(a[],n)

```
1. Start
2. for i ← 0 to n-1 do
{
      min ← i;
       for j ← i + 1 to n do
        {
           If (a[min] > a[j]) then
                min ← j
        }
      If (min!=i )then
                {
              temp ← a[i]
              a[i] ← a [min]
              a[min] ← temp
                }
}
3. stop
```

Example

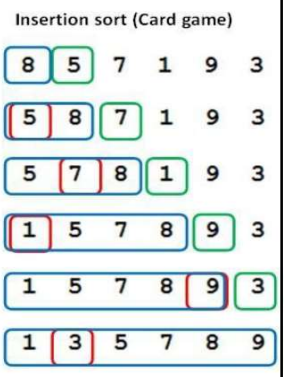


Insertion Sort

- **Insertion sort is a sorting algorithm that places an unsorted element at its suitable place in each iteration.**
- **Insertion sort works similarly as we sort cards in a card game.**
- **We assume that the first element is already sorted then we select an unsorted element.**
- **If the unsorted element is greater than the first element , it is placed on the right ,otherwise to the left.**
- **In the same way, other unsorted elements are taken and put in their right place.**
- Time complexity = $O(n^2)$

- Insertionsort(a[],n)

```
1. Start
2. for(i=1;i<n;i++)
{
  k=a[i];
  for( j=i – 1;j>=0 &&k<a[j];j--)
       {
              a[j+1]=a[j];
       }
   a[j+1]=k;
}
3. stop
```



HASHING

Module 4

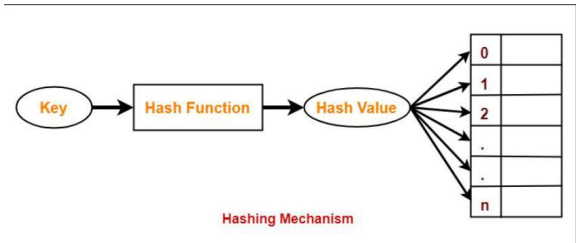
DS

Hashing

- Hashing in data structure is an efficient technique to perform the search and insertion operations very fast irrespective of the size of the data.
- **Drawback of linear and binary search:-**
- **Search time is dependent on the number of elements.**
- **Many key comparisons are involved.**
- **As the number of elements increases, time taken to perform the search also increases**

- **HASHING**
- **Hashing is a well-known technique to search any particular element among several elements.**
- **It minimizes the number of comparisons while performing the search.**
- **Search time is independent on the number of elements.**
- **Searching can be performed in constant time(time**
- **complexity O(1)).**
- **Hashing is extremely efficient.**

- **Hashing Mechanism- In hashing,**
- **An array data structure called as Hash table is used to store the data items.**
- **Based on the hash key value, data items are inserted into the hash table.**
- **Hash Key Value-**
- **Hash key value is a special value that serves as an index for a data item.**
- **It indicates where the data item should be stored in the hash table.**
- **Hash key value is generated using a hash function.**



Hashing Mechanism

- **Hash Function-**
- **Hash function is a function that maps any big number to a small integer value.**
- **Hash function takes the data item as an input and returns a small integer value as an output.**
- **The small integer value is called as a hash value.**
- **Hash value of the data item is then used as an index (address) for storing it into the hash table.**
- **ie,h(k)=a ,here h is the hash function, k is the key and 'a' is (array index/hash table address) the hash value of the key.**

- Types of hash functions
- Hash function generates a table address from a given key.
- If the size of hash table is m, then we need a hash function that can generate addresses in the range 0 to m-1.
- Two criteria for choosing a good hash function ;
  - 1. It should be easy to compute.
  - 2. It should generate addresses with minimum collision.
- Different types of hash functions are:
- Truncation(extraction)
- Mid square method
- Folding method
- Division method(Modulo Division)

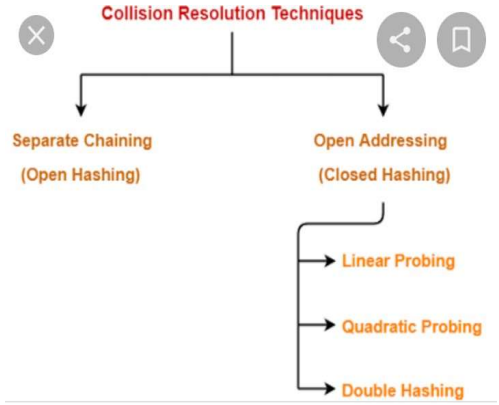
- 1. Truncation (or Extraction)
- This is the easiest method for computing address from a key.
- Here we take only a part of the key as address, for example we may take some rightmost digits or leftmost digits.
- Eg: 82394561,87139465,83567271,85943228 and hash table size is 100, take 2 rightmost digits for hash key value.
- So 61,65,71 and 28 are hash table address or hash key value.
- There may be a chance of collision because last two digits can be same in many keys.

- 2. Division method
- ● In this method, the key is divided by the table size and the remainder is taken as the address for hash table.
- h(k)=k mod m
- Example: elements to be placed in a hash table are 42,78,89,64 and let's take table size as 10.
- Hash (key) = Elements % table size;
- 2 = 42 % 10;
- 8 = 78 % 10;
- 9 = 89 % 10;
- 4 = 64 % 10;

- 3. Mid Square Method
- In this method, the middle part of the squared element is taken as the index.
- Eg: Element to be placed in the hash table are 210, 350, 99, 890
- and the size of the table be 100.
- 210* 210 = 44100, index = 1 as the middle part of the result (44100) is 1.
- 350* 350 = 122500, index = 25 as the middle part of the result (122500) is 25.
- 99* 99 = 9801, index = 80 as the middle part of the result (9801) is 80.
- 890* 890 = 792100, index = 21 as the middle part of the result (792100) is 21.

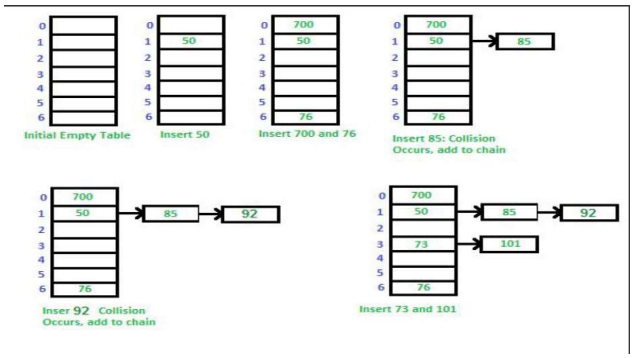
- 4. Digit Folding Method
- In this method the element to be placed in the table uses a hash key value, which is obtained by dividing the elements into various parts and then combine the parts by performing some simple mathematical operations.
- Element to be placed are 23576623, 34687734.
- hash (key) = 235+766+23 = 1024
- hash key) = 34+68+77+34 = 213

- What is Collision?
- Since a hash function gets us a small number for a key which is a big integer there is a possibility that two keys result in the same value.
- The situation where a newly inserted key maps to an already occupied slot in the hash table is called collision and must be handled using some collision handling technique.
- There are mainly two methods to handle collision:
- 1. Separate Chaining (Open Hashing)
- 2. Open Addressing (Closed Hashing)



- **Separate Chaining:**
- **The linked list data structure is used to implement this technique.**
- **When multiple elements are hashed into the same slot index, then these elements are inserted into a singly-linked list which is known as a chain.**
- **Hence, the conclusion is that in separate chaining, if two different elements have the same hash value then we store both the elements in the same linked list one after other.**

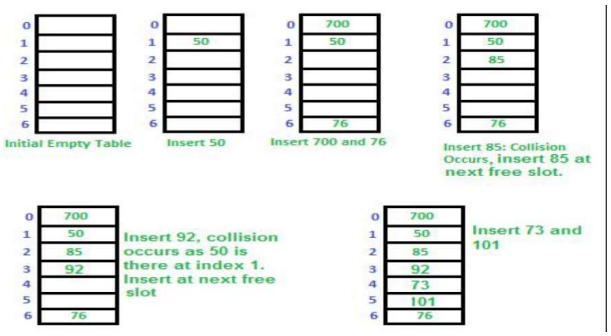
Let us consider a simple hash function as "key mod 7" and a sequence of keys as 50, 700, 76, 85, 92, 73, 101.



- **Open Addressing:**
- **In Open Addressing, all elements are stored in the hash table itself.**
- **The key which caused the collision is placed inside the hash table itself but at a location other than its hash address.**
- **So at any point, the size of the table must be greater than or equal to the total number of keys.**
- **This approach is also known as closed hashing because the array is assumed to be closed or circular.**
- **This entire procedure is based upon probing(Searching).**

- **a) Linear Probing:**
- **If address given by hash function is already occupied, then the key will be inserted in the next empty position in the hash table.**
- **If the address given by hash function is 'a' and it is not empty, then we will try to insert the key at next location ie at address 'a+1'. If address a+1 is also occupied then we will try to insert at next address ie a+2 and we will keep on trying successive location till we find an empty location where the key can be inserted.**
- **Thus the formula for linear probing can be written as**
- **H(k,i)=(h(k)+i) mod m**
- **Drawback of Linear Probing:**
- **Primary Clustering:- groups of records stored next to each**
- **other**

eg:Let us consider a simple hash function as "key mod 7" and a sequence of keys as 50, 700, 76, 85, 92, 73, 101.



- **b) Quadratic Probing**
- **In this method we look for i^2'th slot.**
- **We always start from the original hash location.**
- **If only the location is occupied then we check the other slots.**
- **The formula for quadratic probing can be written as:-**
- **$H(h,i)=(h(k)+i^2) \bmod m$**
- **Eg:Let h(k) be the slot index computed using hash function.**
- **If slot h(k) % m is full, then we try (h(k) + 1*1) % m**
- **If (h(k) + 1*1) % m is also full, then we try (h(k) + 2*2) % m**
- **If (h(k) + 2*2) % m is also full, then we try (h(k) + 3*3) % m**
- **Drawback of Quadratic Probing:**
- **Secondary Clustering:- Clusters are formed by records which follow the same collision path.**
- **It can't access all the positions of the table.**

- **c) Double Hashing**
- **Double hashing is a technique that reduces clustering in an optimized way.**
- **In this technique, the increments for the probing sequence are computed by using another hash function.**
- **We use another hash function h2(k) and look for i*h2(k) slot in i'th rotation.**
- **Let h(k) be the slot index computed using hash function.**
- **If slot h(k) %m is full, then we try (h(k) + 1*h2(k)) % m**
- **If (h(k) + 1*h2(k)) % m is also full, then we try (h(k) + 2*h2(k)) % m**
- **If (h(k) + 2*h2(k)) % m is also full, then we try (h(k) + 3*h2(k)) %m**