MUTHAYAMMAL ENGINEERING COLLEGE

DEPARTMENT OF INFORMATION TECHNOLOGY

Subject Code:19ITC01

Subject Name: Data Structures

UNIT- 5
HASHING,SEARCHING,SORTING

Hashing Techniques:

Hashing:

- Hashing is a technique used to Performing Insertion, deletion & search operations in the constant average time by implementing Hash table Data Structure .
- It is used to Index and Retrieve Items in a Database.

Two Types of Hashing

1. **Static Hashing .**
2. **Dynamic Hashing .**

Static Hashing :

- It is the hash function maps search key value to a fixed set of locations.

Dynamic Hashing :

- The Hash Table can grow to handle more Items.The associated Hash Function must Change as the table grows.

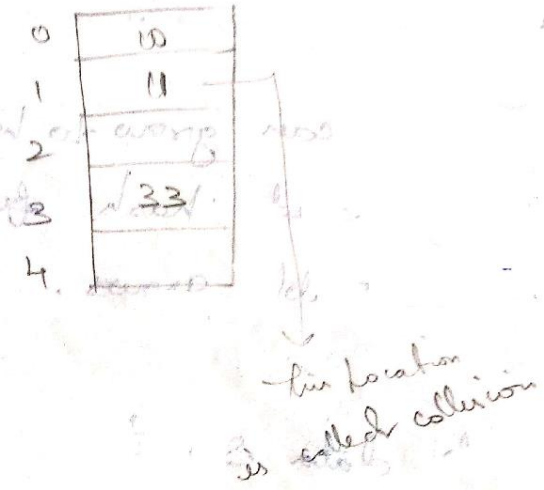
Hash Table :

- The Hash Table data structure is a array of some fixed size table containing the Keys.
- A Key is a values associated with each record.
- A Hash table is partition into array of size.
- Each Bucket has many slots and each slots holds one records.

Hash Function :

- A Hashing Function is a key to address Transformation which acts upon a given Key to complete the Relative Position of the Key in a array

- A Key can be a member of a String etc..
- A Hash Function Formula is
- **Hash (Key Value) =(Key Values % Table Size)**



- Hash (Key Value)=Key Values % Table Size
- Hash(10) =10 % 5=0 Hash(21)= 21 % 5=1
- Hash(33) =33 % 5 =3
- Hash(11)=11 % 5=1

A Good Hashing consist of

- Minimum Collision.
- Be easy and quick to complete.
- Distribute Key Value Every in the Hash Table.
- Use all the Information Provided in the Key.

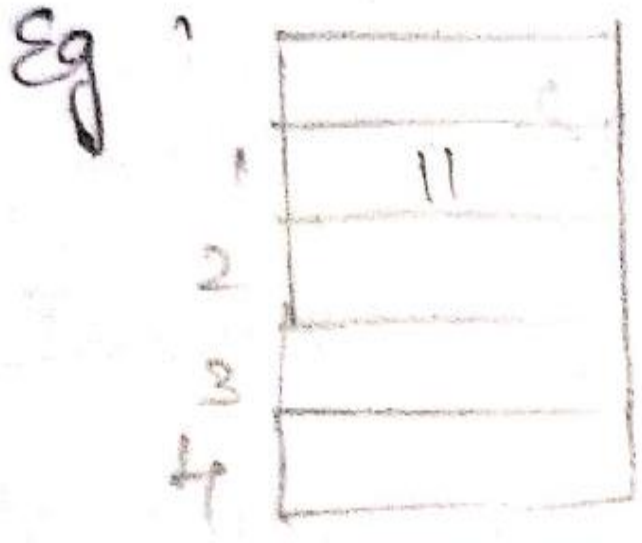
Application of Hash Table:

- Database Systems
- Symbol Tables .
- Data Dictionaries
- Network Processing Algorithm
- Browse Casher.

Collision :

- Collision occurs when a hash values of a records being Inserted hashes to an Address That already contains a difference Record.

" When Two Key values hash to the position".



Insert 11,21 in hash table

11 =Hash(11) =11%5 =1

21= Hash (21) =21%5 =1

(collision occur)

Collision Resolution :

- The Process of finding another Position for The Collide Record Is said to be collision Resolution Strategy.

Two categories of Hashing .

1. **Open Hashing**

   eg: Separate Chaining.

2. **Closed Hashing .**

   eg : Open Addressing ,Rehashing and Extendable hashing.

Open Hashing :

- Each Bucket in the Hash table is the head of a Linked List.
- All Elements that hash to a Particular Bucket are Placed on the Buckets Linked List .

Closed Hashing:

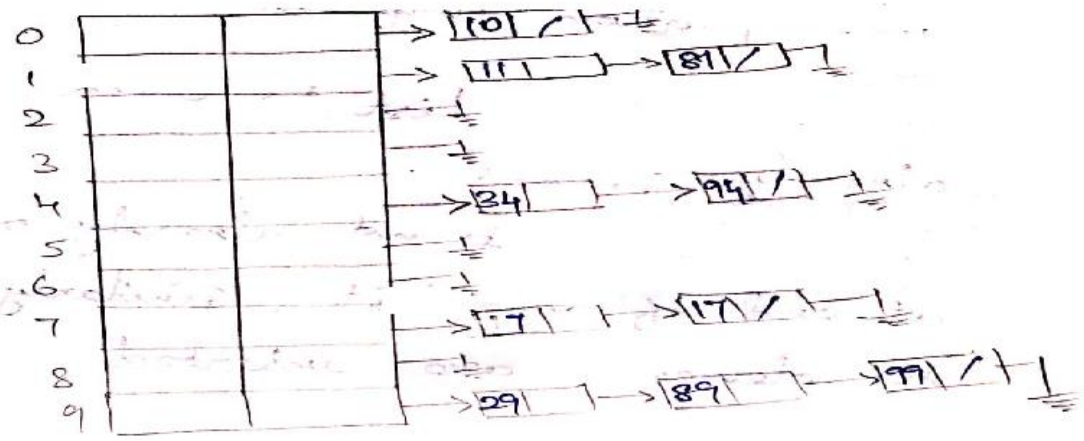
- Ensures that all elements are stored directly in to the Hash Table.

Separate Chaining :

- Separate Chaining is an open hashing Technique
- A Pointer fields is added to each record Location.
- In this method the table can never overflow since the linked are only extended upon or New Keys.

Example:

10 , 11 , 81 , 10 , 7 , 34 , 94 , 17 , 29 , 89 , 99



Hash ('Keyvalue) = keyvalue %. table size

Insert = 10 : Hash(10) = 10 %. 10 = 0

Insert 11 : Hash (11) = 11 %. 10 = 1

Insert 81 : Hash (81) = 81 %. 10 = 1

Insert 10 ⇒ ×

Insert 7 : Hash (7) = 7 %. 10 = 7

Insert 34 : Hash (34) = 34 %. 10 = 4

Insert 94 : Hash (94) = 94 %. 10 = 4

Insert 17 : Hash (17) = 17 %. 10 = 7

Insert 29 : Hash (29) = 29 %. 10 = 9

Insert 89 : Hash (89) = 89 %. 10 = 9

Insert 99 : Hash (99) = 99 %. 10 = 9

The element 81 collides to the same of the hash value to place the value 81 at this position perform the following.

1. Traverse the list to check whether it is already present.
2. Since, it is not already present, insert at end of the list similarly, the rest of the elements are inserted.

Advantages:

- More number of elements can be inserted as it uses of linked list.
- Collision resolution is simple and efficient.

Disadvantages:

It requires pointers, which occupies more memory space.

Closed hashing:

- Collide elements are stored at another slot in the table. Ensures that all elements stored directly in the hash table.

Eg: Open addressing

Rehashing and extendable hashing.

Thank you

MUTHAYAMMAL ENGINEERING COLLEGE

DEPARTMENT OF INFORMATION TECHNOLOGY

Subject Code:19ITC01

Subject Name: Data Structures

Open Addressing:

- Open addressing also called closed hashing which is an attentive to resolve the collision with linked list.
- If a collision occurs, alternative cells are tried until an empty cell is found (i.e) cells ho(x), h1(x), h2(x) are tried in succession.

  There are three common collision strategies, there are

  1. Linear Probing
  2. Quadratic Probing
  3. Double Hashing

1.Linear Probing:

Key value :18,70,65,51,13

Table size : 7

Formula :

$H(k,v) = k.v \,\%\, T.S$

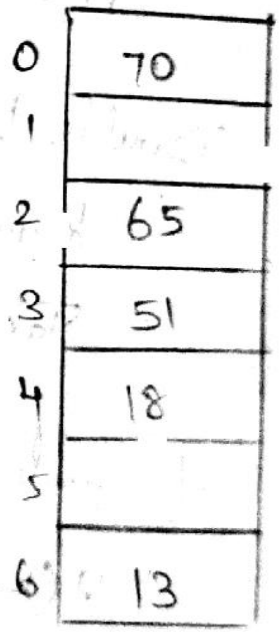
$H(18) = 18 \,\%\, 7 = 4$

$H(70) = 70 \,\%\, 7 = 0$

$H(65) = 65 \,\%\, 7 = 2$

$H(51) = 51 \,\%\, 7 = 2$

$H(13) = 13 \,\%\, 7 = 6$



Quadratic probing

Key Value : 18, 70, 65, 51, 13.

Table Size : 7

Formula :

(i) $H(k.v) = k.v \ \%.T.S$

(ii) $a + i^2$

where

$i = 1, 2, 3, \ldots n$

Solution :

$H(k.v) = k.v \ \%.T.S$

$H(18) = 18 \ \%.7 = 4$

$H(70) = 70 \ \%.7 = 0$

$H(65) = 65 \ \%.7 = 2$

0	70
1	
2	65
3	51
4	18
5	
6	13

$H(51) = 51 \ \%.7 = 2 \rightarrow$ collision.

Formula : location of collision

$a + i^2$

$= 2 + 1^2 = 2 + 1 = 3$

$H(13) = 13 \ \%.7 = 6$

Double hashing

Keyvalue : 18, 70, 65, 51, 13

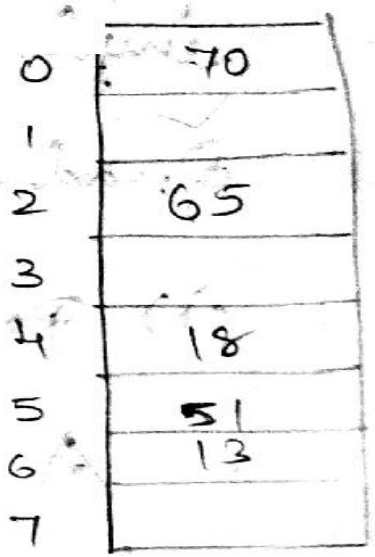
Table size : 7

Formula :

$H(k.v) = k.v \ \%.Ts \rightarrow a$

$H'(k) = 9 - (k.v \ \%.9) \rightarrow a'$

$(a+a') \% T.S$



Solution :

$H(K.V) = K.V \% T_S$

$H(18) = 18 \% 7 = 4$

$H(70) = 70 \% 7 = 0$

$H(65) = 65 \% 7 = 2$

$H(51) = 51 \% 7 = 2 \rightarrow$ collision

$H'(K.V) = 9 - (K.V \% 9)$

$H'(51) = 9 - (51 \% 9)$

$= 9 - 6 = 3 \longrightarrow a'$

$a + a' = 2 + 3 = 5$

$a + a' \% T.S = 5 \% 7 = 5$

$H(13) = 13 \% 7 = 6.$

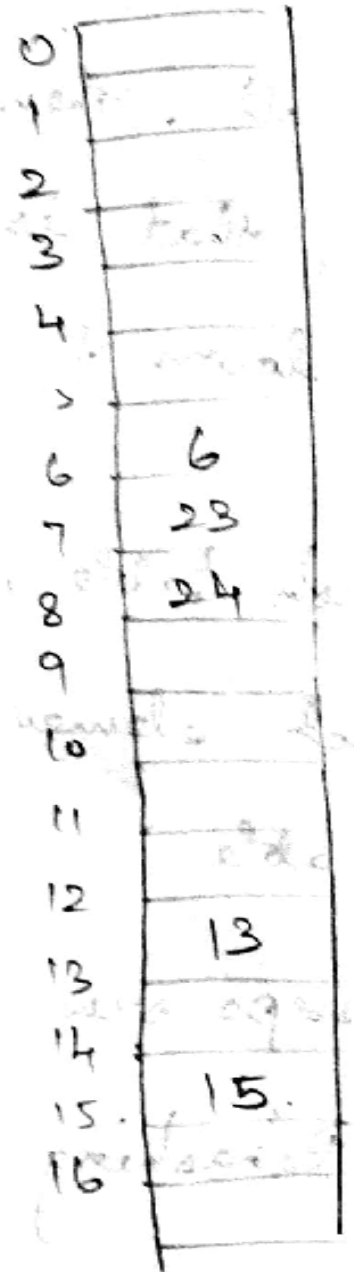
Rehashing:

- If the table get's to full. Then the rehashing method new tables that is about twice as with and scan down the entire original hash table.
- The entire original hash table, computing the new hash value for each element and in sorting it in the new table.
- Rehashing is very expensive operations, the running time is O(N), Rehashing can be implements is several ways with Quadratic probing such as

  i). Rehash as soon as the table is half full.

- A new table is created as table so full. The size of the table so full. The size of the table is 17, as this to the first prime (i.e) a) twice as large as the old table size.

- The new hash function is h(x) = Xmod 17 the old table is scanned and the elements, 6,15,24,23 and 13 are inserted into the new table.



i) insert (6) = 6 %. 17 = 6

ii) insert (15) : 15 %. 17 = 15

iii) insert (23) : 23 %. 17 = 6

$H_1(x) = (6+1) \%. 17 = 7 \%. 17 = 7$

iv) 24 %. 17 = 7

$= (7+1) \%. 17 = 8 \%. 17$

$= 8$

v) insert 13

$= 13 \%. 17$

$= 13.$

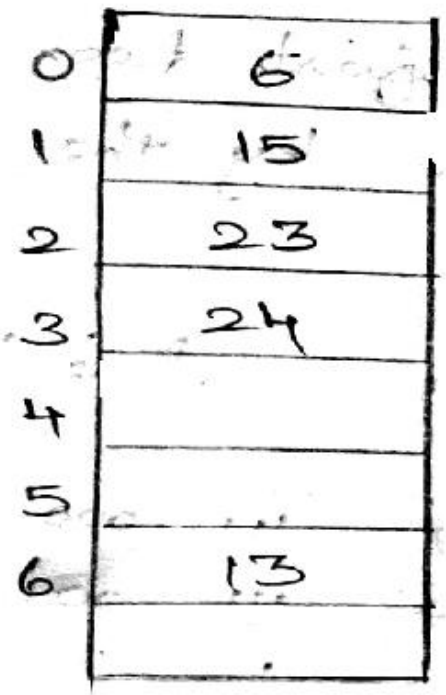
Advantages:

- Programmer does not about the table size.
- Simple to implement.
- It can be used in order data structure as well.

ii) Rehash only when an insertion fails. Suppose the elements 13,15 ,23 ,24 & 6 are insert into an open addressing hash table of size.

Hash function h(x) = x mod 7

Hash Function $h(x) = x \bmod 17$



1) Insert = 13

$H(13) = 13 \% 7$

$= 6$

2) Insert = 15

$H(15) = 15 \% 7$

$= 1$

3) Insert = 24

$H(24) = 24 \% 7$

$= 3$

4) Insert = 6

Hash(6) = 6 % 7

= 6 (collid

Hash(6) = (6+1)

= 7 % 7 =

Linear Probing is used to resolve the collisi

If 23 is inserted into the Table ,the Resulting Table Will be over 70% Full.

1) Insert 23:

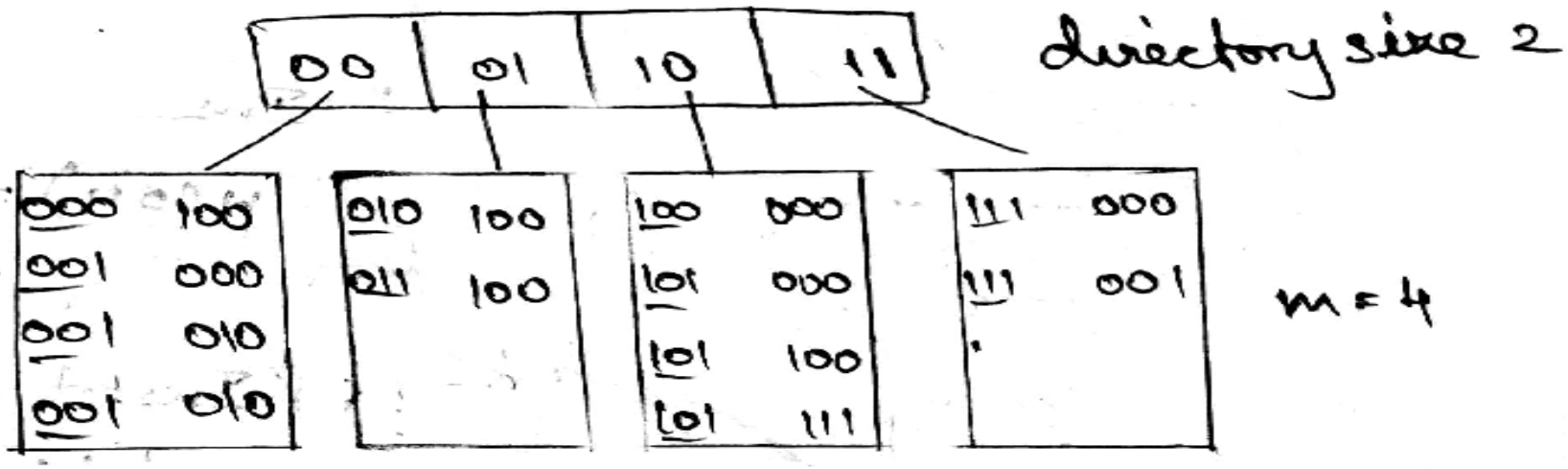
Hash(23)=23%7

=2

Extendable Hashing:

- Extendable Hashing Allows a find to be Performed in two disk accesses. Insertion Also Requires few Disk accesses.
- Let us support consider our data consist of determined by the leading two bits are the data .

- In each leaf has upto m=4 elements ,identified This is indicated by the number in Parenthesis.



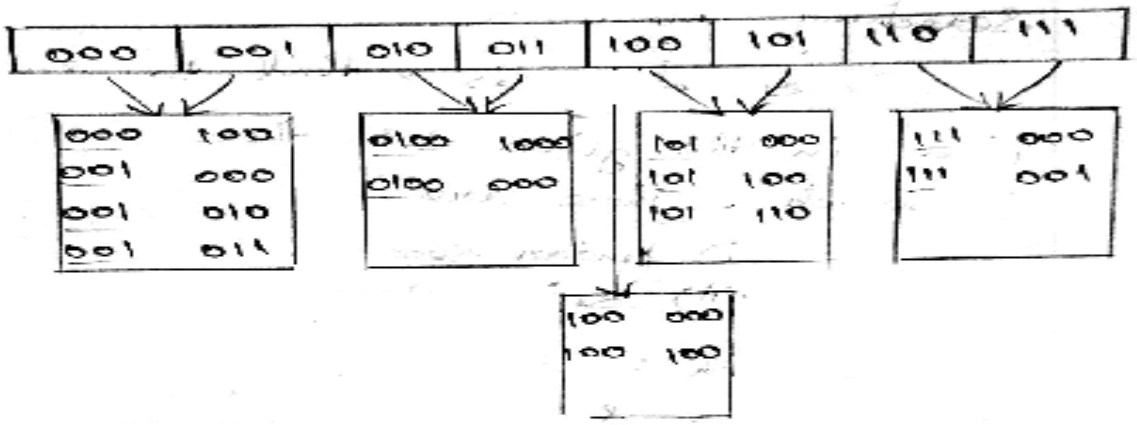
Extendible hashing original data

Suppose that we want to insert the key

100

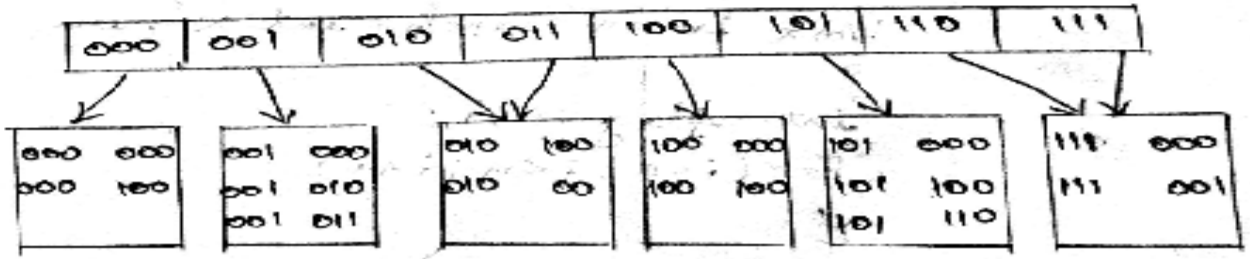
This would go into the third leaf but as the third leaf is already full, that there is no room.

We split this leaf into two leaves, which are now determined by the first three bits now the directory size is increased to three.



extendible hashing after insertion of 100 100 and directory split

Similarly 000 000 is inserted, then first leaf is split



Advantages:

- Provides quick access times for insert an find operation on large database.

Disadvantages:

- This algorithm does not work if there are more then m duplicates.

Thank you

MUTHAYAMMAL ENGINEERING COLLEGE

DEPARTMENT OF INFORMATION TECHNOLOGY

Subject Code:19ITC01

Subject Name: Data Structures

Searching Algorithm:

Searching is method to search a data item in the given set. There are two types of search.

There are

1.Linear Search

2. Binary Search

Linear Search:

- Linear search is used to search and data item in the given set in the sequential manner Starting from the first element it is also called as Sequential Search.

Routine for Linear Search:

```
void linear search(int x, int a[], int n)
{
int flag=0,i;
for(i=0;i<n;i++)
{
```

```
    if(x==a[i])
{
    flag=1
    break;
}
}
    if(flag==1)
    print f(“the element is found”);
else
    print f(“the element is not found”);
}
```

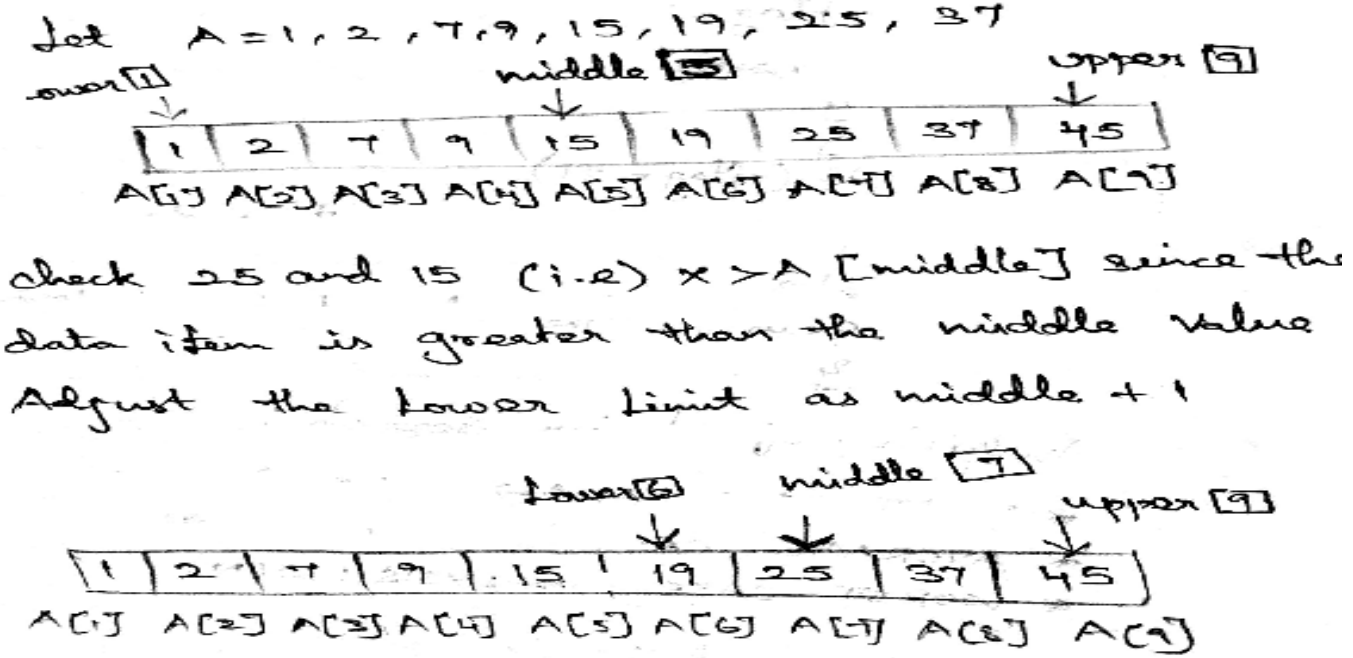
Analysis of Linear Search :

- BEST CASE ANALYSIS:0(1)
- AVERAGE CASE ANALYSIS:0(N)
- WORST CASE ANALYSIS: 0(N)

Binary Search:

- Binary Search is used to Search an Item in a Sorted list . In this Method ,Initialize the lower Limit as 1 And Upper Limit as N(N-1)
- The middle position is computed as (Lower +Upper)/2 and check the element in the Middle Position with the Data item to be Searched.

- If the data item is greater then are Middle Value then the Lower limit Is adjusted to one Greater then the middle value.
- Otherwise , The Upper Limit is adjusted to One Less then the Middle Value Ex: X=25.



Routine For Binary search:

```
   void binary search(int x,int a[],int n);
{
   int lower,upper,mid ;
   lower=1;
   upper=n;
   while(lower<upper)
{
   mid=(lower+upper)/2;
   if(x>a[mid])
   lower=mid+1;
```

```
    else if(x<a[mid])
    upper=mid-1;
else
{
    printf("element is found");
    break;
}
}
}
```

Analysis;

- BEST CASE ANALYSIS: 0(1)
- AVERAGE CASE ANALYSIS:0(logN)
- WORST CASE ANALYSIS: 0(LogN)

Thank you

MUTHAYAMMAL ENGINEERING COLLEGE

DEPARTMENT OF INFORMATION TECHNOLOGY

Subject Code:19ITC01

Subject Name: Data Structures

SORTING

Def: Sorting is a Process (or) Technique of Arranging a group or a Sequence of Data Elements in an order either in ascending or descending.

Two Types of Sorting:

1. Internal Sorting
2. External Sorting.

Internal Sorting:

- A sorting in which all records of the file to be sorted should be within the main memory at the time of sorting.

External Sorting:

- Sorting is which at the time of Sorting Some Record of the file to be Sorted can be in the secondary memory.

Internal Sorting:

1. Insertion Sort
2. Shell sort

3.Heap Sort

4.Quick Sort

5.Radix Sort

6.Bubble Sort

7.Selection Sort

External Sorting

1.Merge Sort

2.Two Way Sort

3.Multiple Way Merge Sort

4.Polyphase merge sort.

Insertion Sort:

- Insertion Sort Works by tasking element from the list one by one and inserting them in their current position into a new sorted list.
- Insertion sort consists of N-1 Passes Where N is the number of element to be sorted .
- The ith Pass of insertion sort will insert the ith
- Element A[1] , A[2] ,….A[i-1] .After Doing this Insertion the Record occupying A[1],….A[i] are in sorted order.

Insertion and Routine:

```
   void insertion sort (element type a[], int N )
{
   int  j, p ;
   element type tmp ;
   for(p=1; p<N ; p++)
{
   tmp =a[p];
   for (j=p ; j>0&& a[j-1]>tmp ;j--)
   a[j] =a[j-1];
   a[j]=tmp ;
} }
```

Example:

Consider an Unsorted array as Follows :

20, 10, 60, 40, 30, 15

Passes of Insertion Sort

								Position Moved
original		20	10	60	40	30	15	
After	i=1	10	20	60	40	30	15	1
	i=2	10	20	60	40	30	15	0
	i=3	10	20	40	60	30	15	1
	i=4	10	20	30	40	60	15	2
	i=5	10	15	20	30	40	60	4
sorted array		10	15	20	30	40	60	

Analysis of insertion Sort :

- WORST CASE ANALYSIS - O(N^2)
- BEST CASE ANALYSIS - O(N)
- AVERAGE CASE ANALYSIS -O(N^2)

Shell Sort :

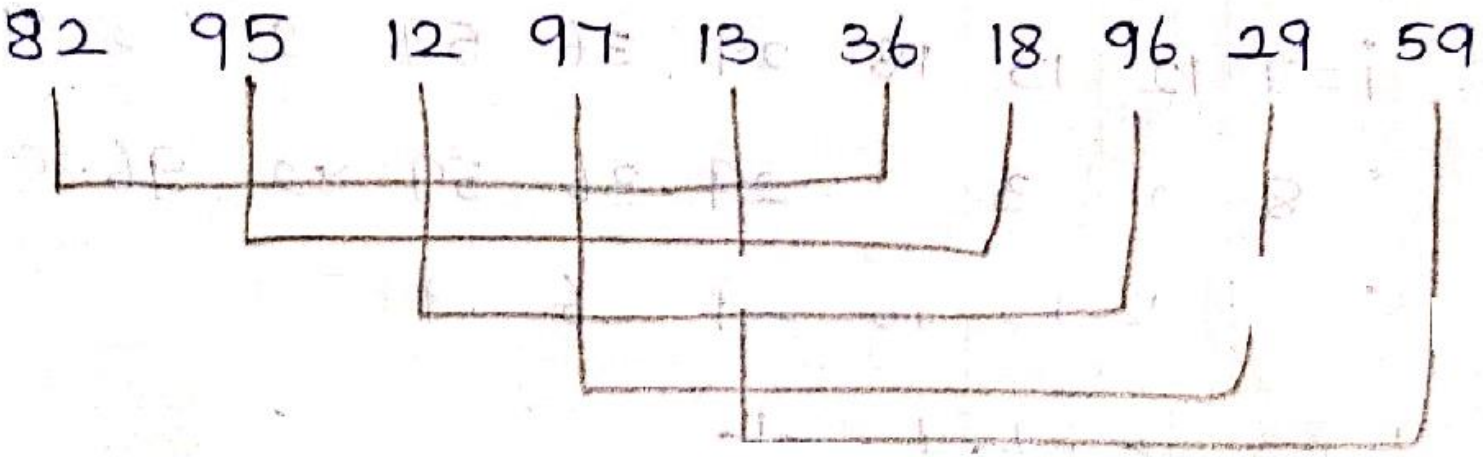
- This method is an Improvement over the Simple Insertion Sort .In this Method The Element at Fixed Distance K (K is Preferably prime Number) or compared .
- The distance will then be decremented by Some fixed amount and again the Comparison will be made . Finally Individuals elements will be compared.

Ex :

82, 95, 12, 97, 13, 36, 18, 96, 29, 59

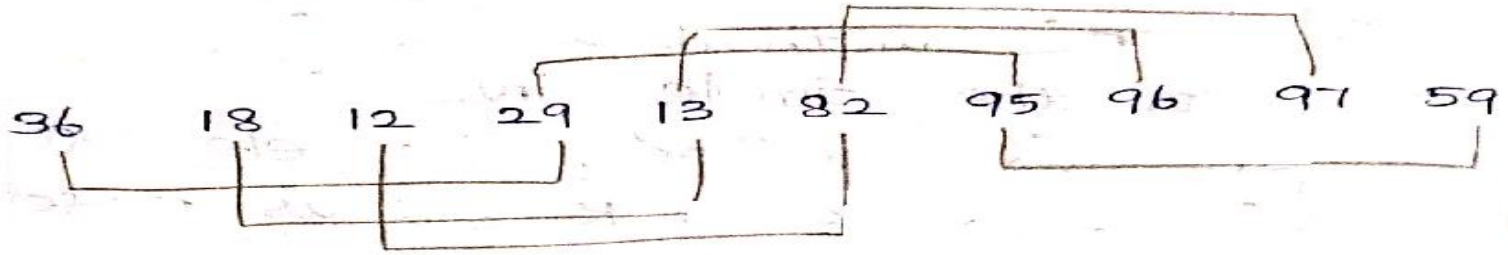
K = 5, 3, 1

Pass 1 : K = 5



36 18 12 29 13 82 95 96 97 59

Pass 2 : K = 3



29: 13 12 36 18 82 54 46 97 95

Pass 3 : K=1

	29	13	12	36	18	82	59	96	97	95	P.I
i=1	13	29	12	36	18	82	59	96	97	95	1
i=2	12	13	29	36	18	82	59	96	97	95	2
i=3	12	13	29	36	18	82	59	96	97	95	0
i=4	12	13	18	29	36	82	59	96	97	95	2
i=5	12	13	18	29	36	82	59	96	97	95	0
i=6	12	13	18	29	36	59	82	96	97	95	1
i=7	12	13	18	29	36	59	82	96	97	95	0
i=8	12	13	18	29	36	59	82	96	97	95	0
i=9	12	13	18	29	36	59	82	95	96	97	2

The sorted elements are

12, 13, 18, 29, 36, 59, 82, 95, 96, 97

Quick Sort [Partition Exchange Sort]:

- The idea Behind This Sorting is Much easier In the Two Sort List Rather Then One Long list.
- All The Element on the Left Side Of Pivot Should Be Smaller Or Equal To The Pivot.
- All The Element On the Right Side Of Pivot Should Should Be Greater Then Or Equal To Pivot.

The Process for Sorting the Element Quick Sort is as:

I. Take the First Element of List as Pivot .

II. Place At The Proper Place In List So one Element of The List (i.e) Pivot will be at its Proper Place.

III. Create two Sublist's Left Child, Right Child of Pivot.

IV. Repeat the same Process Until all element of List are at Proper Position in List.

For Placing the Pivot at Proper Place we have a Need to do The Following Process:

I. Compare the Pivot Element One by One From Right to Left For Getting The Element Which Has Value Less Then Pivot Element .Interchange The Element With Pivot Element.

II. Now the Comparison will start from the Interchanged element position from left to right for getting the element which has higher Value then Pivot.

III. Repeat the same process Until Process is at its Proper position.

Let us take a list of element and process through quick sorting :

(48), 29, 8, 59, 72, 88, 42, 65, 95, 19, 82, 62

← low

Here we are taking 48 as pivot and we share as start comparison from right to left.

Now the first element less than 48 is 19 so interchange it with pivot, (i.e) 48.

19 29 8 59 72 98 42 65 95 (48) 82

high →

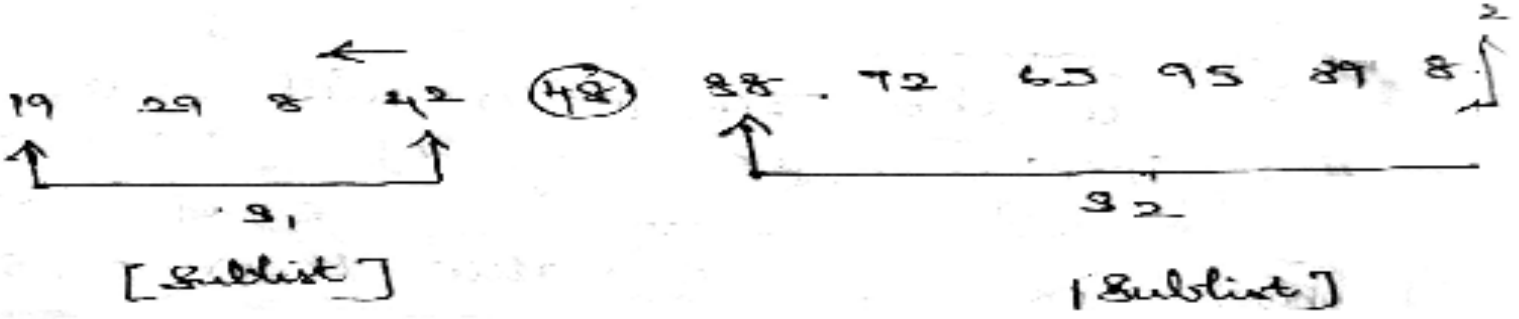
19 29 8 (48) 72 88 42 65 95 59 82 62

19 29 8 42 72 88 (48) 65 95 59 82 62

19 29 8 42 (48) 88 72 65 95 59 82

First element greater then 48 is 72.

So, we can divide list into two sublist left and right side first.



19 29 8 42

now 19 as pivot

19 29 8 42

8 29 19 42

8 19 29 42 → the sublist one is sorted

8 19 29 42 48

88 72 65 95 59 82 68

sublist 2

88 72 65 95 59 82 68

Now 88 as pivot :

68 72 65 72 59 82 88

68 72 65 88 59 82 95

68 72 65 82 59 95

sublist 4

68 72 65 82 88

88 as pivot :

57 72 65 82 68

59 68 65 82 72

59 65 82 72

sublist 5 sublist 6

82 72

59 65 68 72 82 88 95

Now Combine 2 Sub list

8 19 29 42 48 59 65 68 72 82 88 95

All The Elements Are New Sorted.

Analysis of Quick Sort :

- WORST CASE ANALYSIS - O[N^2]
- BEST CASE ANALYSIS - O[N log N]
- AVERAGE CASE ANALYSIS – O[N log N]

Advantage :

- It is Faster than other O(N log N) algorithm.
- It has better cache Performance and High Speed.

Limitations: Requires More Memory Space

Heap Sort :

- In heap sort the array of Interpret as a binary Tree. This Method pass 2 Phases.
- In Phase 1: Binary heap is Constructed.
- In Phase 2: Delete min Routine is Performed.

Phase 1: Two Properties of Binary Heap :

- Structure Property .
- Heap order Property.

Structure Property :

- For Any Element in Array Position i , The Left child is in 2i+1 (i.e) The Cell after the Left Child

Heap Order Property:

- The key Values in the parent Node is Smaller then or equal to the key Value of any in its Child node.
- To Build the Heap , apply the heap order Property Starting from the Right Most Non – Leaf Node at the Bottom level.

Phase 2:

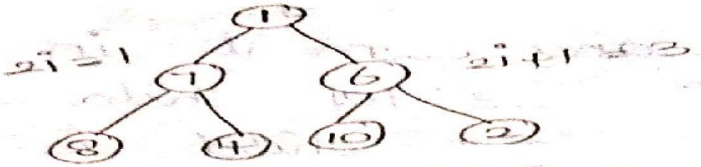
The Array Element are Stored using Deletion Operation.

Example :

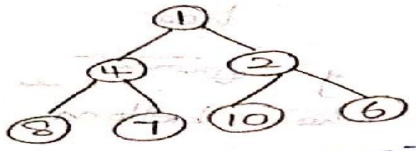
For example :

1	7	6	8	4	10	2
1	2	3	4	5	6	7

Phase 1 :



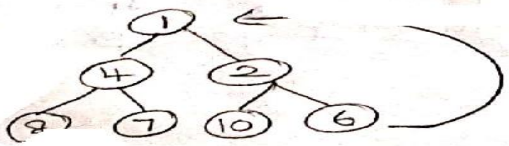
Binary heap satisfying structure property.

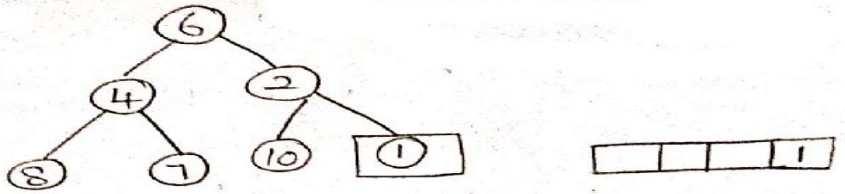


Binary heap satisfying heap order property.

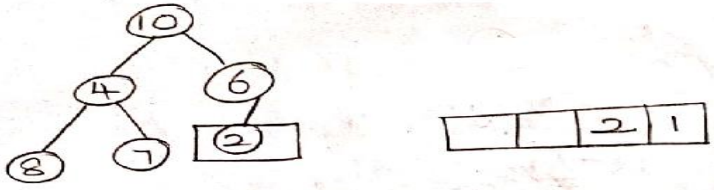
Phase 2 :

Remove the smallest element from the heap and place it in the array.

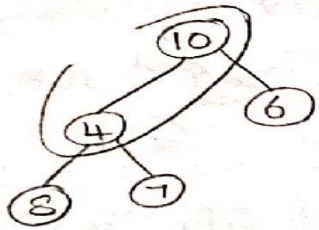


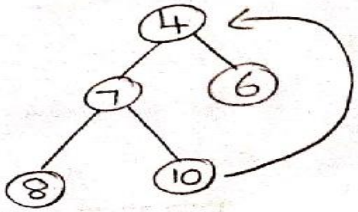


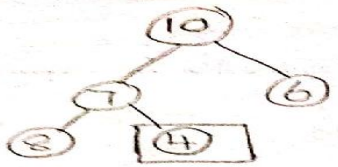
satisfies heap order property

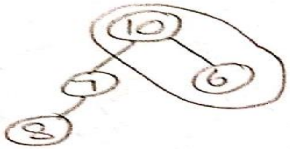


swap (4) :

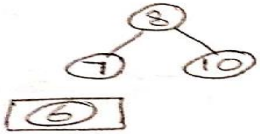


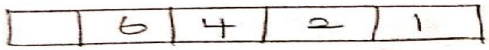


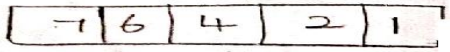


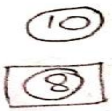


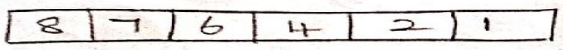
similarly apply the procedure for other element.

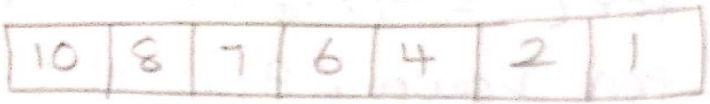












Routine for Heap sort :

Routine for Heap Sort :

```
#define left child [i] (2*(i)+1)
void perdown (element type A[] , int i , int N)
{
int child ;
element type tmp ;
for(tmp =A[i] , left child (i)<N , i=child)
{
child =left child (i);
if (child!=N-1&&A[child +]>A[child])
child ++ ;
```

```
    if (tmp<A[child])
    A[i]=A[child]);
else
Break ;
}
    A[i] =tmp;
}
    void heap sort(element type a[],int N)
{
    int I;
    for(i=N/2;i>=0 ; i--)
```

```
    perdown (A ,i , N);
    for(i=N-1; i>0; i--)
{
    swap (&A[0] ,&A[i]);
    per down (A,o,i);
}
}
}
```

Analysis of Heap Sort :

- Worst Case Analysis =O(N log N)
- Best case Analysis = O(N log N)
- Arg Case Analysis = O(N log N)

Advantages:

- It is efficient for Sorting Large number of Element.
- It has the Add of Worst case.

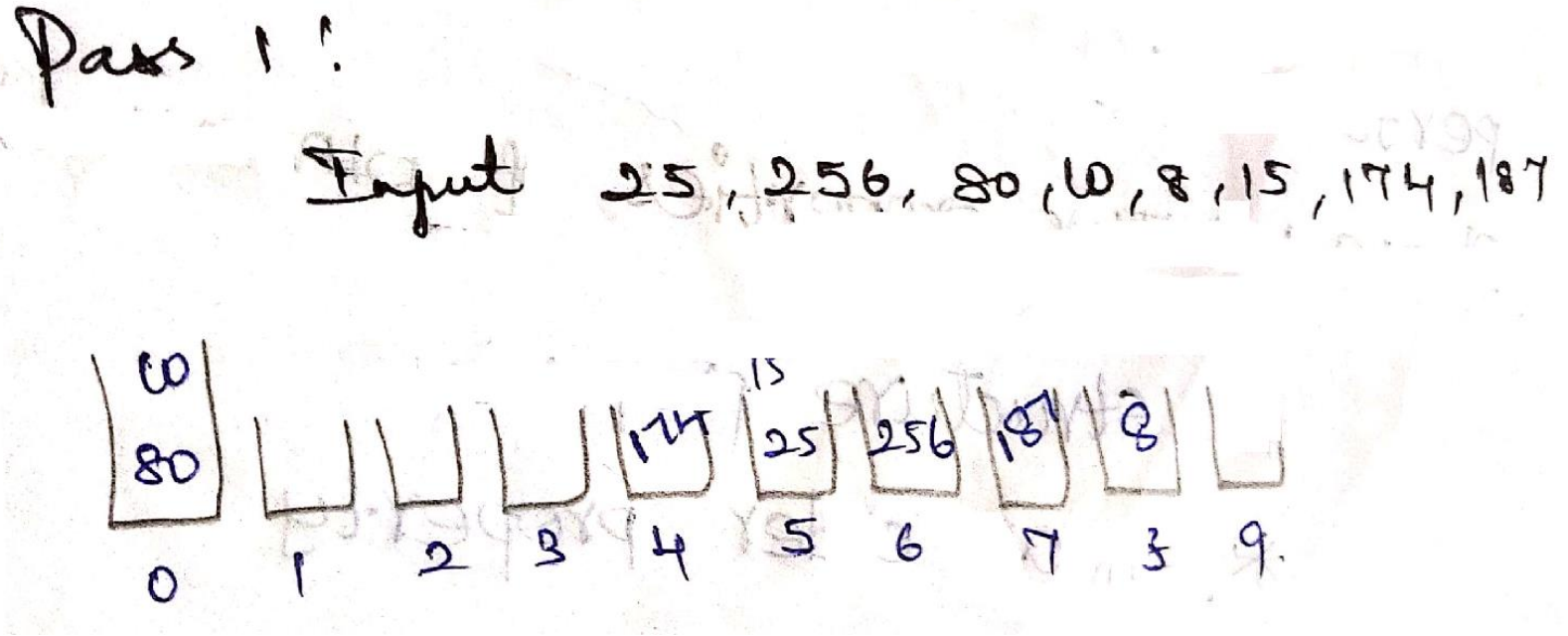
Limitation :

- It is Not a stable Sort .
- It Require Most Processing Time .

Radix Sort :

- Radix Sort is one of the Linear Sorting algorithm for Integers.
- It is generated from radix Sort.
- It can be performed using Bucket 0 to 9.

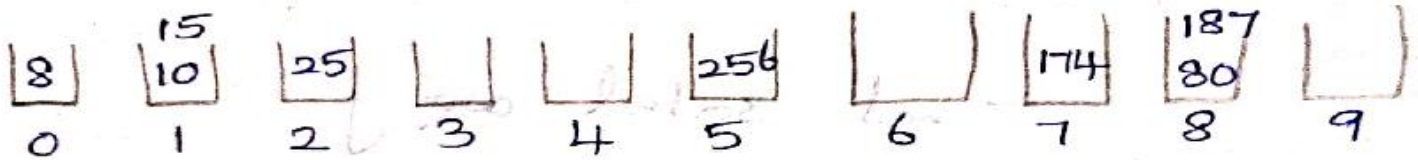
- It is also called as Binsort.
- In First Pass all element arranged according to the least Significant digit. In Second Pass ,the element are arranged according to the next least significant digit and so on.



After pass 1 : 80, 10, 174, 25, 15, 256, 187, 8.

Pass 2 :

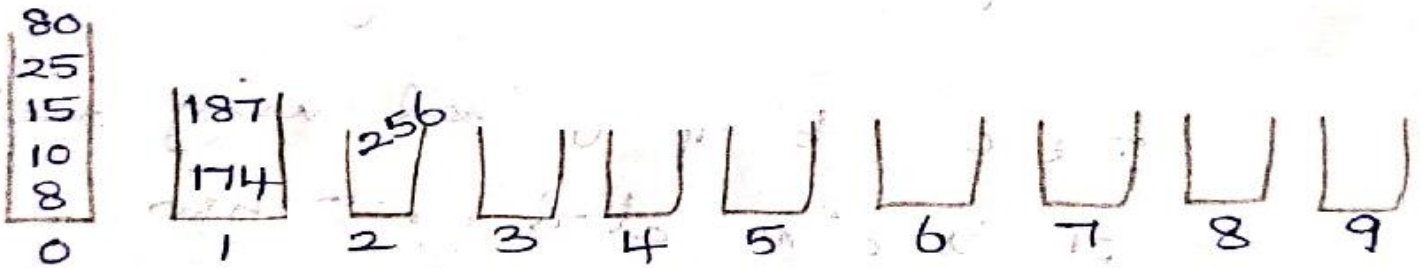
Input : 80, 10, 174, 25, 15, 256, 187, 8.



After pass 2 : 8, 10, 15, 25, 256, 174, 80, 187

Pass 3 :

Input : 8, 10, 15, 25, 256, 174, 80, 187



After pass 3 : 8, 10, 15, 25, 80, 174, 187, 256

External Sorting
Merge Sort :

- The most common algorithm used in external Sorting is the merge sort this algorithm follows **Divide and Conquer strategy**.

Merge Sort Routine :

```
void Msort ( element type A[], element type
tmparray [],int left ,int right)
int center ;
if (left<right)
{
```

```
    center =(left +right)/2;
    msort(A, tmp array, left ,center ) ;
    msort (A,tmp array , center , right ) ;
    merge (A , tmp array ,left ,center+1,right);
}
}
    void merge sort (element type A[],int N))
{
    element type *tmparray ;
    tmparray =malloc(N*sizeof(element type ));
```

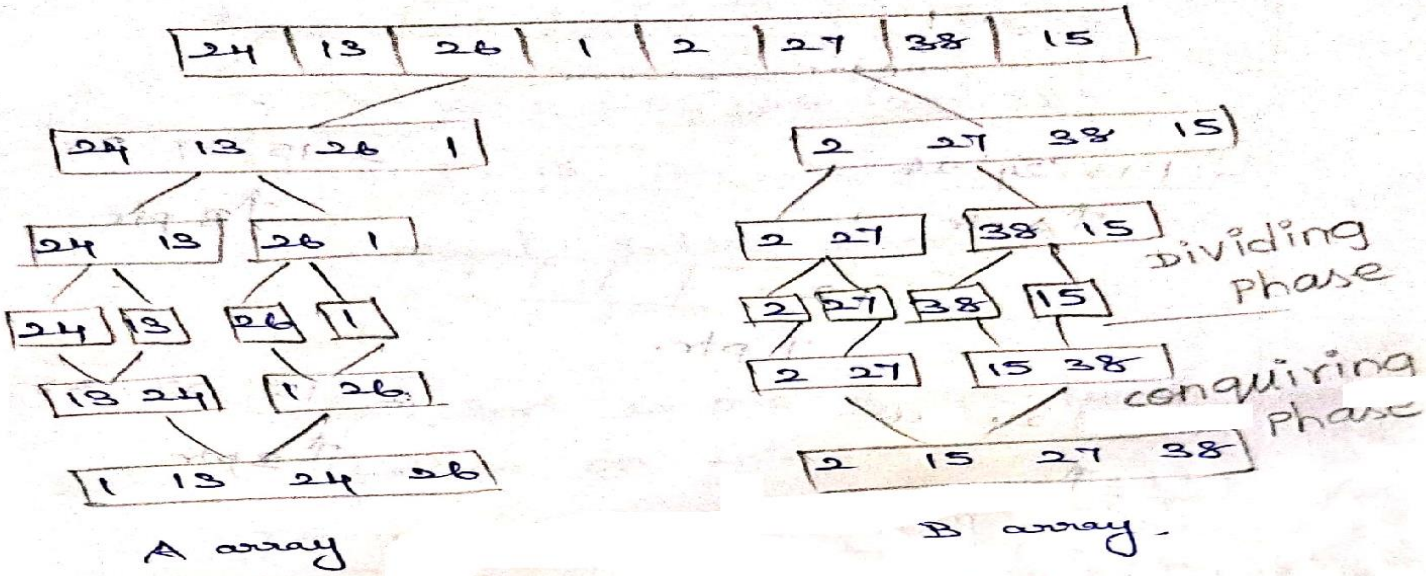
```
   if (tmparray!=NULL)
{
   msort (A,tmparray,0,N++);
   free(tmp array);
}
else
{
   fatal error (“no sapce for tmp array”)
}
}
```

Merge Routine:

```
void merge(element type A[], element type
tmparray [], int LPOS , int RPOS , int rightend)
{
int I, leftend ,numelement , tmppos;
left end =RPOS-1;
TMPPOS =LPOS ;
num element  = rightend –LPOS +1;
while (Lpos<=leftend && RPOS<=rightend)
if (A[LPOS] <=A[RPOS])
tmp array [tmppos++] =A[LPOS++];
```

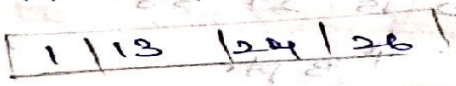
```
else
   tmp array [tmppos++]=A[RPOS++];
   while(LPOS<=leftend)
   tmparray [tmppos++] =A[LPOS++];
   while(RPOS< =rightend)
   tmparray [tmppos++] =A[RPOS++];
   for(i=0; i<numelements; i++; rightend--)
   A[rightend ]=tmparray [rightend];
}
```

Eg: 24, 13, 26, 1, 2, 27, 38, 15.



Let us consider first 4 elements.

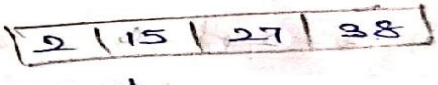
1, 13, 24, 26 as A



Aptr

A Array

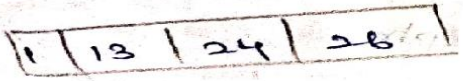
2, 15, 27, 38 as B.



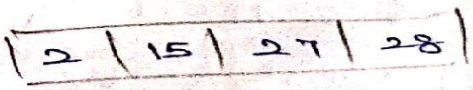
Bptr

B Array

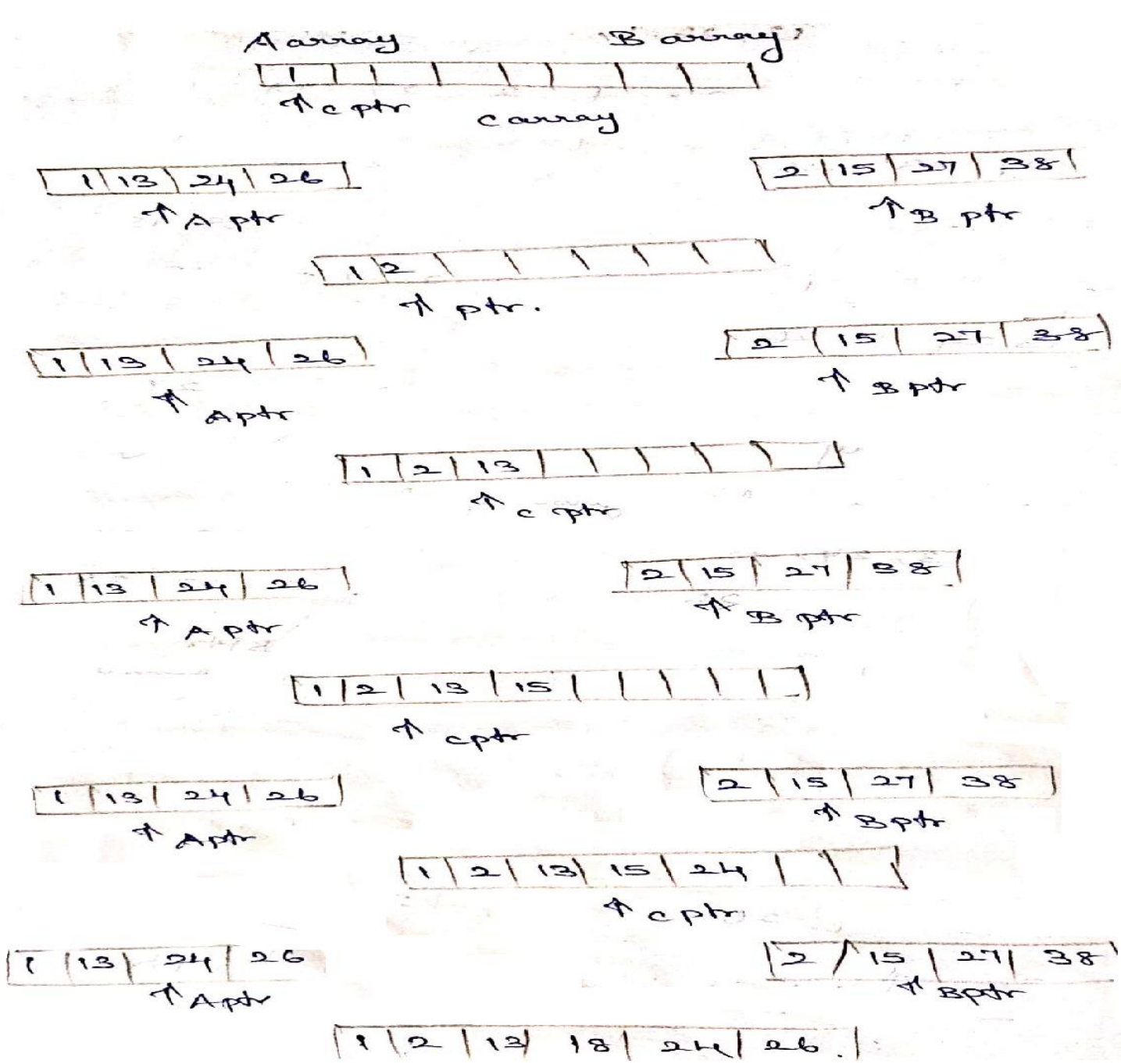
cptr & array

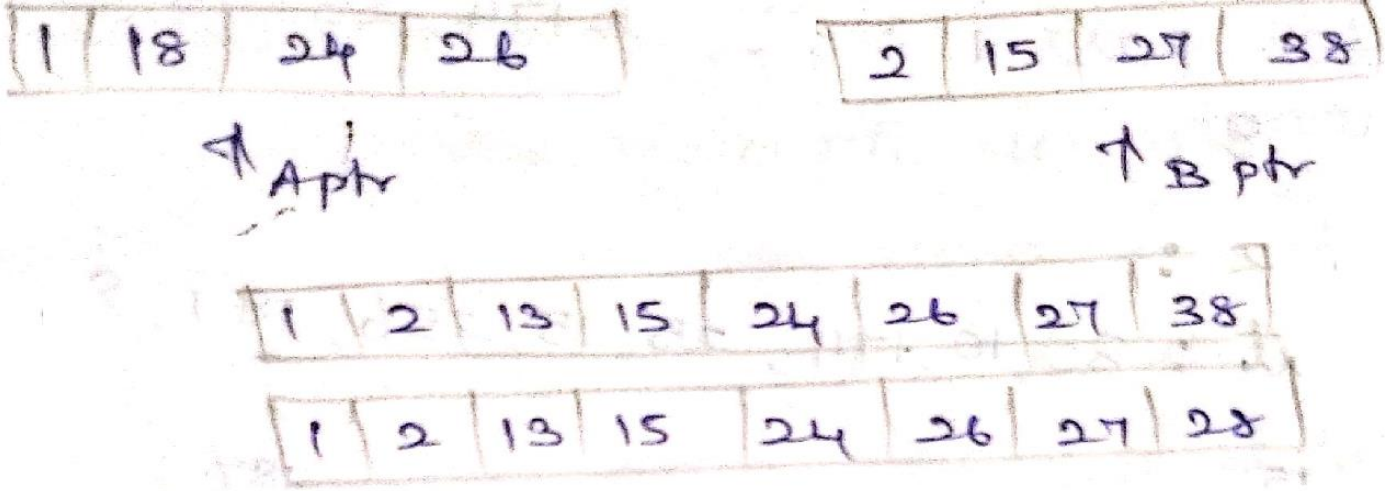


Aptr



Bptr


A array
B array
1
cptr
c array
1 13 24 26
A ptr
2 15 27 38
B ptr
1 2
ptr.
1 13 24 26
A ptr
2 15 27 38
B ptr
1 2 13
c ptr
1 13 24 26
A ptr
2 15 27 38
B ptr
1 2 13 15
cptr
1 13 24 26
A ptr
2 15 27 38
B ptr
1 2 13 15 24
cptr
1 13 24 26
A ptr
2 15 27 38
B ptr
1 2 13 15 24 26


1 18 24 26
2 15 27 38
A ptr
B ptr
1 2 13 15 24 26 27 38
1 2 13 15 24 26 27 28
Final sorted array.

Thank you