Unit-3

Searching and sorting

1) Linear or sequential search -

Concept → Enter array size N = 5

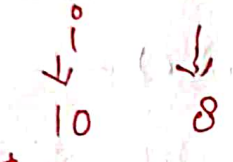
Enter array element

10 8 12 6 14

Enter item to be searched

e = 6 or e = 16

i = 0



if (e == a[i])
found
otherwise
i++;

} repeat these steps while (i <= N)

if (i == N)
Not found

case 1 → e = 6 e == a[i]

1. i = 0 6 == 10 False i = 1.
2. i = 1 6 == 8 False set i = 2
3. i = 2 6 == 12 False set i = 3
4. i = 3 6 == 6 True found

case 2 → e = 13

1. i = 0 13 == 10 False i = 1
2. i = 1 13 == 8 False i = 2
3. i = 2 13 == 12 False i = 3
4. i = 3 13 == 6 False i = 4
5. i = 4 13 == 14 False i = 5

i = 5 ⇒ False Exit from loop i == 5 ⇒ True Not found

Algorithm → Linear Search (A[maxsize], N)

1. read N, f=0
2. read Array elements
3. read element to be searched (item)
4. set i = lb
5. while (i <= UB)
6. if (A[i] = item)
   write <illegible>
   set f=1
   [End of if]
7. set i=i+1
   [End of loop]
8. if (f=1)
   write "found"
   else
   write "Not found"
9. Exit

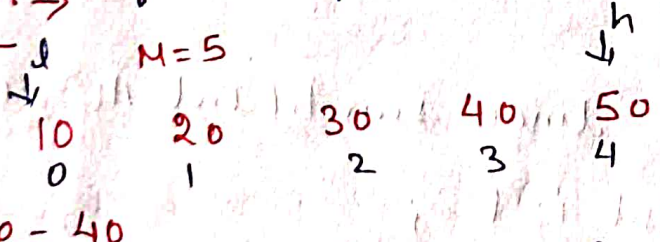
C function →

```
int linear_search(int A[], int N, int e)
{
  int i, f=0;
  for(i=0; i<N; i++)
  {
    if(e == A[i])
      f=1;
  }
return(f);
}
```

```
int main()
{
  int A[30], i, N, e, f;
printf("Enter No. of elements");
scanf("%d", &N);
printf("Enter array elements");
  for(i=0; i<N; i++)
    scanf("%d", &A[i]);
printf("Enter item to be searched");
  scanf("%d", &e);
    f= linear_search(A, N, e);
  if(f==1)
    printf("found");
  else
    printf("Not found");
}
```

Binary Search → for Binary search array must be sorted.

Concept →



e = 40

set l = 0 and h = N-1 = 4

while (l <= h) repeat

mid = (l+h)/2

if(e == A[mid]

found

else if(e > A[mid])

l = mid+1

else

r = mid-1

if(l > h) notfound

$l = 0 \qquad h = 4$

$l <= h \Rightarrow 0 < 4 \Rightarrow T$

$mid = (l + h)/2 = (0+4)/2 = 2$

$e == A[mid] \Rightarrow 40 = A[2] = 30 \Rightarrow F$

$e > A[mid] \Rightarrow 4 > 30 \Rightarrow T$

$l = mid + 1 = 2 + 1 = 3$

again $l <= h \Rightarrow 3 <= 4 \Rightarrow T$

$mid = (l + h)/2 = (3+4)/2 = 7/2 = 3$

$e == A[mid] \Rightarrow 40 == 40 \Rightarrow T$

found

case 2 → $e = 60$

$l = 0 \qquad h = 4$

$l <= h \Rightarrow 0 <= 4 \Rightarrow T$

$mid = (0+4)/2 = 2$

$60 == 30 \Rightarrow F$

$60 > 30 \Rightarrow T$

$l = mid + 1 = 2 + 1 = 3$

Again $l <= h \Rightarrow 3 <= 4$

$mid = (3+4)/2 = 3$

$60 == 40 \Rightarrow F$

$60 > 40 \Rightarrow T$

$l = mid + 1 = 4$

Again.

$l <= h \Rightarrow 4 <= 4 \Rightarrow T$

$mid = (4+4)/2 = 4$

$e == A[mid] = 60 == 50$ F

$60 > 50 \Rightarrow T$

$l = mid + 1 = 4 + 1 = 5$

Again $l <= h \Rightarrow 5 <= 4 \Rightarrow F$

$l > h \Rightarrow 5 > 4 \Rightarrow T$

not found

Algorithm → Binary_search (A[max size], item)

1. Set l = LB and h = UB
2. Repeat step 3 & 4 while (l <= h)
3. Set mid = (l + h)/2
4. if (e == A[mid])
   write "found"
   else if (e > A[mid])
   l = mid + 1
   else
   h = mid - 1.
   [END of if]
   [END of while loop]
5. if (l > h)
   write "Not found"
6. Exit.

C function →

```
int binary_search( int A[], int N, int e)
{
   int l=0, h= N-1, mid, f=0;
   while ( l<h)
   {
      mid = (l+h)/2;
   if ( e == A[mid])
   { f=1; break;}
   else if ( e> A[mid]
      l= mid+1;
   else
      h= mid-1;
```

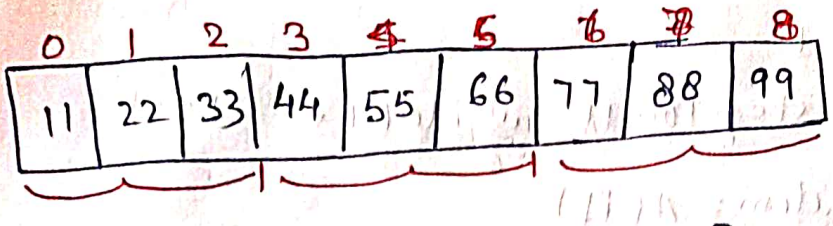
```
    }
  return(f);
}
int main()
{
  int A[30], i, N, e, f;
  printf("Enter Array size");
   scanf("%d", &N);
  printf("Enter array elements");
    for(i=0; i<N; i++)
     scanf("%d", &A[i]);
  printf("Enter item to be searched");
  scanf("%d", &e);
    f= binary_search(A, N, e);
   if(f==1)
     printf("found");
   else
     printf("Not found");
}
```

Binary search (Recursive method) →

```
int binary_search(int A[], int l, int h, int e)
{
   int mid;
    if(l<=h)
    {
       mid = (l+h)/2;
      if(e == A[mid])
        return 1;
```

```
else if (e > A[mid])
  binary_search(A, mid+1, h, e);
else
  binary_search(A, l, mid-1, e);
} return -1;
}
int main()
{
  int A[30], i, N, l, h, f, e;
  printf("Enter array size");
  scanf("%d", &N);
  printf("Enter array elements");
  for (i=0; i<N; i++)
    scanf("%d", &A[i]);
  printf("Enter element to be searched");
  scanf("%d", &e);
  l=0;
  h=N-1;
  f = binary_search(A, l, h, e);
  if(f==1)
    printf("found");
  else
    printf("Not found");
```

Indexed sequential search



item → 88 Set Group size = 3

i = 0 if (item > A[i]) = 88 > 11 ⇒ F

i = i + gs = i + 3 = 0 + 3 = 3 ⇒ i = 3 88 > 44 ⇒ F

i = 3 + 3 = 6 88 > 77 ⇒ F

i = 6 + 3 = 9 i < N ⇒ 9 < 9 ⇒ F

set p = i - gs = 9 - 3 = 6

Apply Linear search for i = p to i = p + gs

found

Algorithm → Indexed_sequential_search()

1. read searched item
2. Divide the array into groups according to group size. (GS)
3. Set i = LB
4. Repeat step while (i <= UB)
5. if (item > A[i])
   break
6. i = i + GS;
7. set p = i - GS

~~for i=1 to~~

8. set i = p
9. repeat step 10 to 11 while (i <= p+gs)
10. if (item = A[i])
    write printf("found");
    break;
11. i = i+1; [END of if]
    [END of loop]
12. if (i == p+gs) then
    write "Not found"
13. exit.

Sorting → To arrange the elements in a particular sequence either in ascending order or descending order.

1) Bubble sort → (process)
Arrange in ascending order

M = 4

40 (swap) 30 20 10

if a[0] > a[1] then swap(a[0], a[1])

30 40 (swap) 20 10

a[1] > a[2] then swap(a[1], a[2])

30 20 40 10

a[2] > a[3] then swap(a[2], a[3])

30 20 10 40

pass-1

Now no more comparision left. And array is also not in ascending order. What happen? largest element is in

Indexed sequential search–

```
int index_sequential_search(int a[], int n, int item)
{
    int gs, i, p;
    if( item<a[0] || item>a[n-1])
      return -1;
    else
     {
        printf("Enter group size");
        scanf("%d", &gs);
        for(i=0; i<n; i=i+gs)
         {
            if( item<a[i])
              break;
         }
        p= i- gs;
       for(i=p; i<p+gs, i++)
        {
          if( item== a[i])
            return 1
        }
       return -1;
     }
void main()
 {
int a[30], i, N, item, f;
printf("Enter size");
scanf("%d", &N);
printf("Enter array elements");
 for(i=0; i<N; i++)
  scanf("%d", &a[i]);
  printf("Enter searchitem");
scanf("%d", &item);
   f= index_sequential_search
          (a, n, item);
 if(f==1)
  printf("found");
else
  printf("Not found");
}
```

last position. Now go to pass-2.

30 swap 20 10 40

again repeat the same process

a[0] > a[1] then swap (a[0], a[1])

20 30 10 40

a[1] > a[2] swap (a[1], a[2])

20 10 30 40

pass-2

next comparision is not required as 40 is already sorted. so in every next pass we have one less comparision.

now array is still not sorted. so go to pass-3

20 10 30 40

pass-3

a[0] > a[1] so swap(a[0], a[1])

10 20 30 40

now array is sorted so stop here.

concept→ ~~for (i=~~ N=4.

- Two thins are repeated. so two loops are required one for pass say (i) and another for comparision (j).
- In every pass, we perform all comparisions. so i is outer loop and j is inner loop (nesting)
- for N=4 total pass is 3. so outer loop is

```
for (i=1; i<N; i++)
```

- N=4

pass 1	i=1	comp=3
pass 2	i=2	comp=2
pass 3	i=3	comp=1

so for ith pass there are (N-i) comparision.

inner loop is

for (j = 1; j <= N - i; j++)

- In inner loop, we compare previous element say a[j] to next element a[j+1].

if (a[j] > a[j+1])

swap(a[j], a[j+1])

- But if start j from 1. then for every pass first comparion is a[1] > a[2]. But first comparion is a[0] > a[1]. so j starts from 0. But total comparision is n-i.

so inner loop is for(j=0; j<= N-i-1; j++)

Algorithm →

Void Bubble_sort(A[maxsize], N)

~~1. read array size (N)~~

~~2. read array elements~~

1. for (i = 1 to N-1)

for (j = 0 to N - i - 1)

if (a[j] > a[j+1]) then

swap(a[j], a[j+1])

[End of inner loop]

2. write array [End of outer loop]

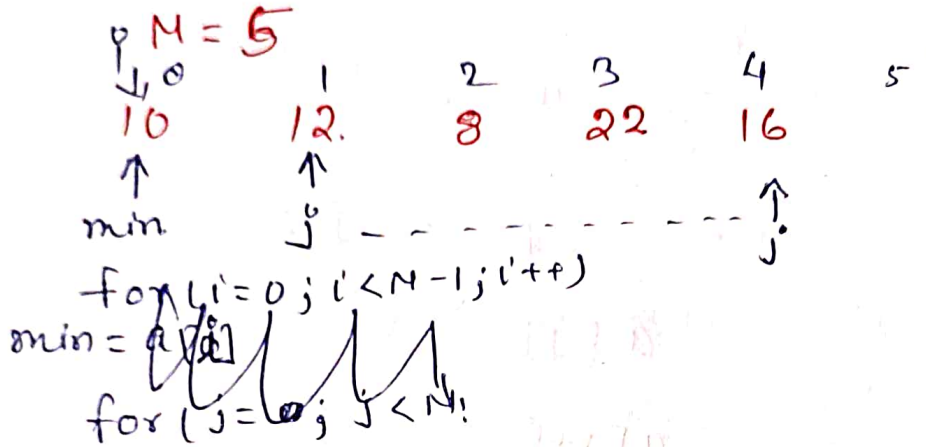
3. Exit.

C program→

```
void bubble_sort(int A[], int N)
{
    int i, j, t;
    for(i=1; i<N; i++)
    {
        for(j=0; j<N-i; j++)
        {
            if(a[j] > a[j+1])
            {
                t = a[j];
                a[j] = a[j+1];
                a[j+1] = t;
            }
        }
    }
}
int main()
{
    int a[30], i, N;
    printf("Enter array size");
    scanf("%d", &N);
    printf("Enter array elements");
    ~~scanf("%d", &a);~~
    for(i=0; i<N; i++)
        scanf("%d", &a[i]);
    printf("Bejor sorting array is");
    for(i=0; i<N; i++)
        printf("%d", a[i]);
```

```
    bubble_sort(a, N);
    printf("After sorting array is");
    for(i=0; i<N; i++)
        printf("%d", a[i]);
}
```

2) Selection sort → process-1



$i = 0$ $\quad min = a[i] = a[0] = 10$

$j = i+1$ $\quad j = 1$ $\quad if(a[j] < min) \Rightarrow a[1] < min$

$\Rightarrow 12 < 10$ False update $J = 1$

$j = 2$ $\quad if(a[2] < min) \Rightarrow 8 < 10 \Rightarrow T$

$min = a[1] = 8$

$j = 3$ $\quad if(a[3] < min) \Rightarrow 22 < 8 \Rightarrow F$

update $J = 4$

$j = 4$ $\quad if(a[4] < min) \Rightarrow 16 < 8 \Rightarrow F$

update $J = 5$

$j = 5$ outside the array

swap(a[i] & min) ⇒ swap(a[0], min)

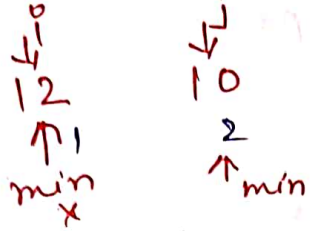
⇒ swap(10, 8)

$i = 0$ pass-1

now the array is

8 12 10 22 16

pass 1 completes when j is outside the array. But array is not sorted. Then go to pass-2



i=1 min = a[1] = 12

j=2 a[j] < min ⇒ a[2] < min ⇒ 10 < 12 = True.
min = 10

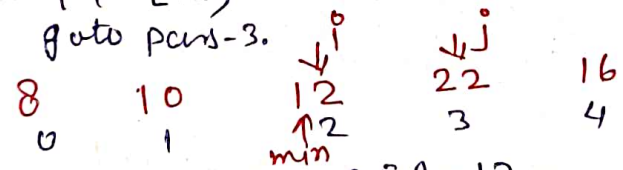
j=3 a[3] < min ⇒ 22 < 10 ⇒ F

j=4 a[4] < min ⇒ 16 < 10 ⇒ F

j=5 inner loop terminates

pass 2

swap(a[1], min) = swap(12, 10). Array is still not sorted.
goto pass-3.



i=2 min = a[2] = 12

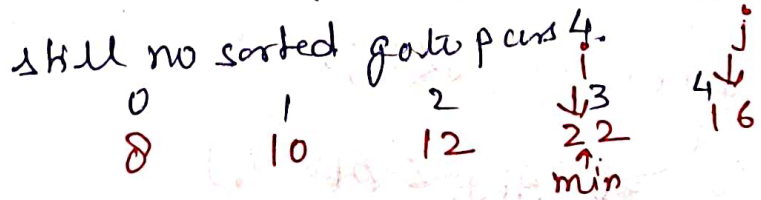
j=3 a[3] < min ⇒ 22 < 12 ⇒ F

j=4 a[4] < min ⇒ 16 < 12 = F.

j=5 outside the inner loop.

swap(a[2], min) ⇒ both are same so no swapping

still no sorted goto pass 4.



i=3 min = a[3] = 22

j=4 a[4] < min ⇒ 16 < 22 ⇒ T
min = a[4] = 16

j=5 Inner loop terminates. swap(a[4], min) ⇒ swap(16, 22)

8 10 12 16 22

now array is sorted. stop.

Algorithm →

selection_sort(A[maxsize], N)

1. for (i = 0 to N-2)

   Set min = i

   for (j = i+1 to N-1)

      if (A[j] < A[min])

         min = j

   [End of inner loop]

   swap (a[i], a[min]),

[End of outer loop]

2. Exit

C program →

```
void selection_sort( int A[], int N)
{
  int i, j, min, t;
  for ( i= 0; i<N-1; i++)
    {
      min = i;
      for(j = i+1, j<N; j++)
       {
         if (A[j] < A[min])
            min = j;
       }
     t= A[i];
    A[i] = A[min];
   A[min] = t;
```

```
}
}
int main()
{
    int A[30], i, N, j;
    printf("enter array size");
    scanf("%d", &N);
    printf("Enter array elements");
    for(i=0; i<N; i++)
    scanf("%d", &A[i]);
    printf("Before sorting Array is");
    for(i=0; i<N; i++)
    printf("%d", A[i]);
    selection_sort(A, N);
    printf("After sorting Array is");
    for(i=0; i<N; i++)
    printf("%d", A[i]);
}
```

Insertion sort →

concept → N = 5

	0	1	2	3	4
A[]	12	11	13	5	6
	↑ j	↑ key			

pass 1 →

for i = 1 set key = A[i] = 11

j = i − 1 = 0

compare key elements to all the previous element

until key < A[j] and j >= 0

11 < 12 and j = 0 if true then

move A[j] to one position right and set j = j − 1. (A[j+1] = A[j])

otherwise set A[j+1] = key.

A[2] = A[1] and A[0] = key.

0	1	2	3	4
11	12	13	5	6
	↑ j	↑ key		

Pass-2

i = 2

~~key = A[j] = A[2] = 13~~

j = i - 1 = 2 - 1 = 1

now key > A[j] i.e 13 > 12 and j >= 0 so

move A[1] to right i.e. A[2] = A[1] and j = 1 - 1 = 0

11	12	12	5	6
↑ j				

~~now again~~

Pass-2 i = 2

key = A[2] = 13 j = 1

11	12	13	5	6
	↑ j	↑ key		

key < A[j] ⇒ 13 < 12 ⇒ False so

A[j+1] = key ⇒ A[3] = key which is already in place.

Pass-3 i = 3

0	1	2	3	4
11	12	13	5	6
		↑ j	↑ key	

key = A[3] = 5 j = 2

(key < A[j] AND j >= 0) ⇒ 5 < 13 and j = 2 >= 0 ⇒ T

so move A[2] to A[3] A[3] = A[2] = 13 and j = 1

11	12	13	13	6
	↑ j			

Again Key < A[1] and j >= 0 ⇒ 5 < 12 and 1 >= 0 ⇒ T

move A[1] to A[2] which gives.

and j = 0

11	12	12	13	6
	↑			
	j			

Again Key < A[0] and j >= 0 ⇒ 5 < 11 and 0 >= 0 ⇒ T

move A[0] to A[1] and j = -1

11	11	12	13	6
↑				
j				

now j >= 0 is false so go outside the loop. and

Set A[j+1] = Key ⇒ A[0] = 5 which gives

0	1	2	3	4
5	11	12	13	6
			↑	↑ Key
			j	

Pass 4- i = 4

Key = A[4] = 6 j = i - 1 = 4 - 1 = 3

Check Key < A[j] and j >= 0 ⇒ 6 < 13 and 3 >= 0 ⇒ T

move A[3] to A[4] and j = 3 - 1 = 2

0	1	2	3	4
5	11	12	13	13
		↑		
		j		

Again check (Key < A[j] and j >= 0) ⇒ 6 < 12 and 2 >= 0 ⇒ T

move A[2] to A[3] and j = 2 - 1 = 1

5	11	12	12	13
	↑			
	j			

Again check (key < A[j], and j >= 0)

6 < 11 and j >= 0 ⇒ T

move A[1] to A[2] and j = 1-1 = 0

0	1	2	3	4
5	11	11	12	13

j

Again check (key < A[j] and j >= 0) ⇒ 6 < 5 and 0 >= 0 ⇒ F

Set A[j+1] = key ⇒ A[1] = key = 6 which gives

5	6	11	12	13

Now array is sorted.

Algorithm ⇒

insertion sort (A[maxsize], N)

1. for (i = 1 to N-1)
2. Set key = A[i]
3. Set j = i-1
4. while (j >= 0 AND A[j] < key)
5. Set A[j+1] = A[j]
6. j = j-1

[End of while loop]

7. Set A[j+1] = key

[End of for loop]

8. Exit.

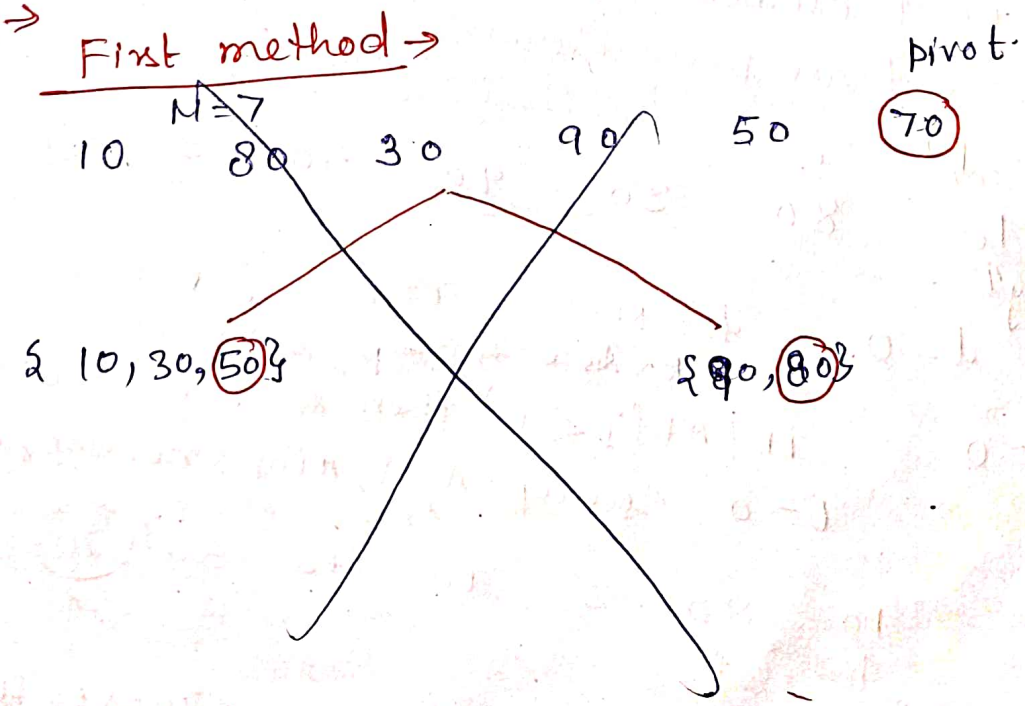
C function →

```
void insertion_sort(int A[], int N)
{
    int key, i, j;
    for (i=1; i<N; i++)
    {
        key = A[i];
        j = i-1;
        while (j>=0 && key<A[j])
        {
            A[j+1] = A[j];
            j=j-1;
        }
        A[j+1] = key;
}
```

Quick sort →

First method →

concept →



Quicksort

M = 7

{ 10, 80, 30, 90, 40, 50, 70 }

{ 10, 30, 40, 50 } { 80, 90 }

{ 10, 30, 40 } { } { 80 } { }

{ 10, 30 } { }

{ 10 } { }

In Quick sort we take last elements as a pivot element. and partition Array into two subarray such that left subarray contains elements less than the pivot element and right subarray contains elements greater than pivot element.

process for partitioning.

	0	1	2	3	4	5	6
	10	80	30	90	40	50	70

i = -1, j = 0, l = 0, h = 6

$h = M-1 = 7-1 = 6$ $i = l-1 = 0-1 = -1$

~~R=A[0]=A[h]=6~~ $p = A[h] = 70$

$j = 0$ if ($A[j] < p$) then set $i = i+1$ and swap($A[i], A[j]$) and $j = j+1$

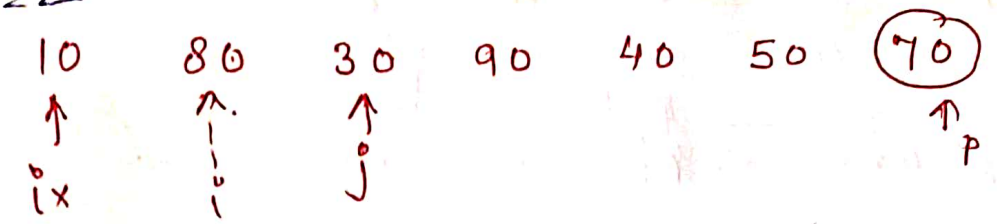
$i = 0$ swap($A[0], A[0]$) ⇒ no change. $j = 1$

$j = 1$

	0	1	2	3	4	5	6
	10	80	30	90	40	50	70
	i	j					p

if ($A[j] < p$) ⇒ $A[1] < p$ ⇒ $80 < 70$ ⇒ F then set $j = j+1$

$j=2$

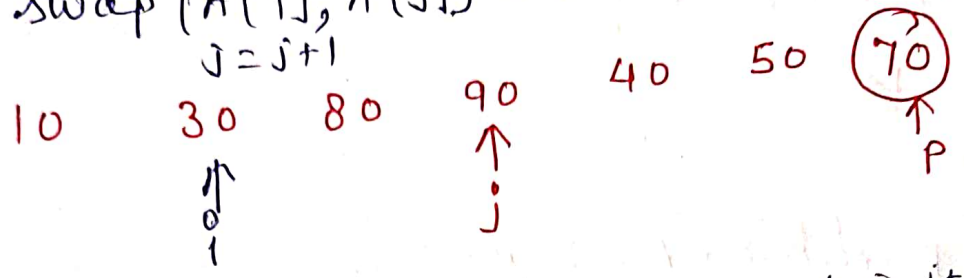


$A[j] < p \Rightarrow A[2] < p \Rightarrow 30 < 70 \Rightarrow T$

$i = i+1 = 1$

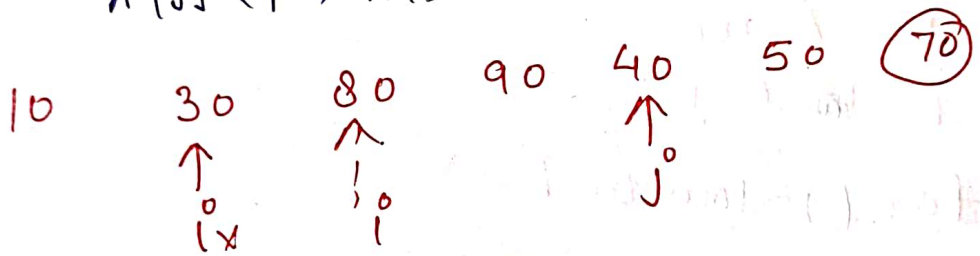
swap $(A[i], A[j]) \Rightarrow$ swap $(A[1], A[2])$

$j = j+1$



$j=3$ $A[j] < p \Rightarrow A[3] < p \Rightarrow$ False set $j=j+1$

$j=4$



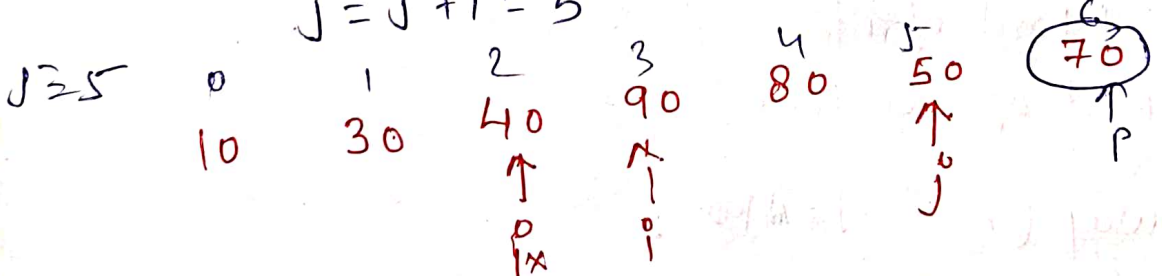
$A[j] < p \Rightarrow 40 < 70 \Rightarrow T$

$i = i+1 = 2$

swap $(A[i], A[j]) \Rightarrow$ swap $(A[2], A[4])$

$j = j+1 = 5$

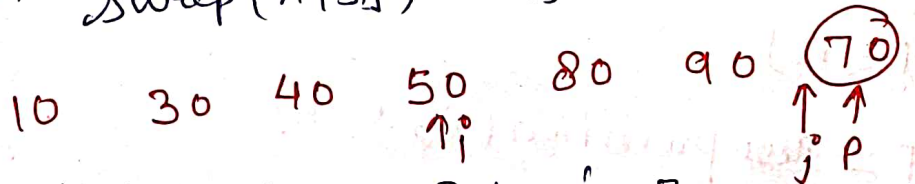
$j=5$



$A[j] \leq p \Rightarrow A[5] < p \Rightarrow 50 < 70 \Rightarrow T$

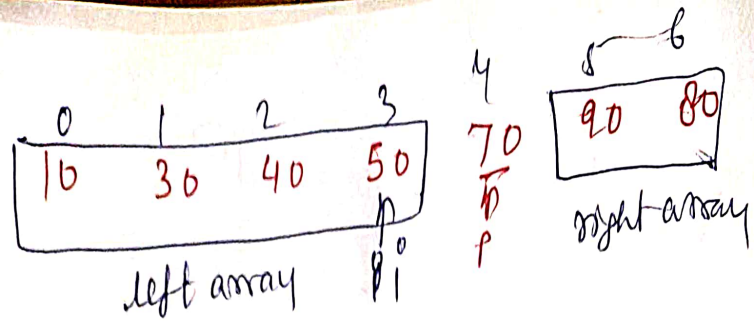
$i = i+1 = 3.$

swap $(A[3], A[5])$ and $j = 5+1 = 6$



$j > high - 1 \Rightarrow$ ~~False~~ $j > 5 \Rightarrow 6 > 5 \Rightarrow T$

now swap $(A[i+1], p)$



return (i+1) ⇒ return (4)

now Apply partitioning process for left and right subarray recursively.

Algorithm

partition (A[], l, h)

1. p = A[h]
2. i = l-1
3. for (j = low to h-1)
   if (A[j] < p) then
   set i = i+1
   swap (A[i], A[j])
   [End of if]
   [End of for loop]
4. swap (A[i+1], A[h])
5. return (i+1)

Quicksort (A[], l, h)

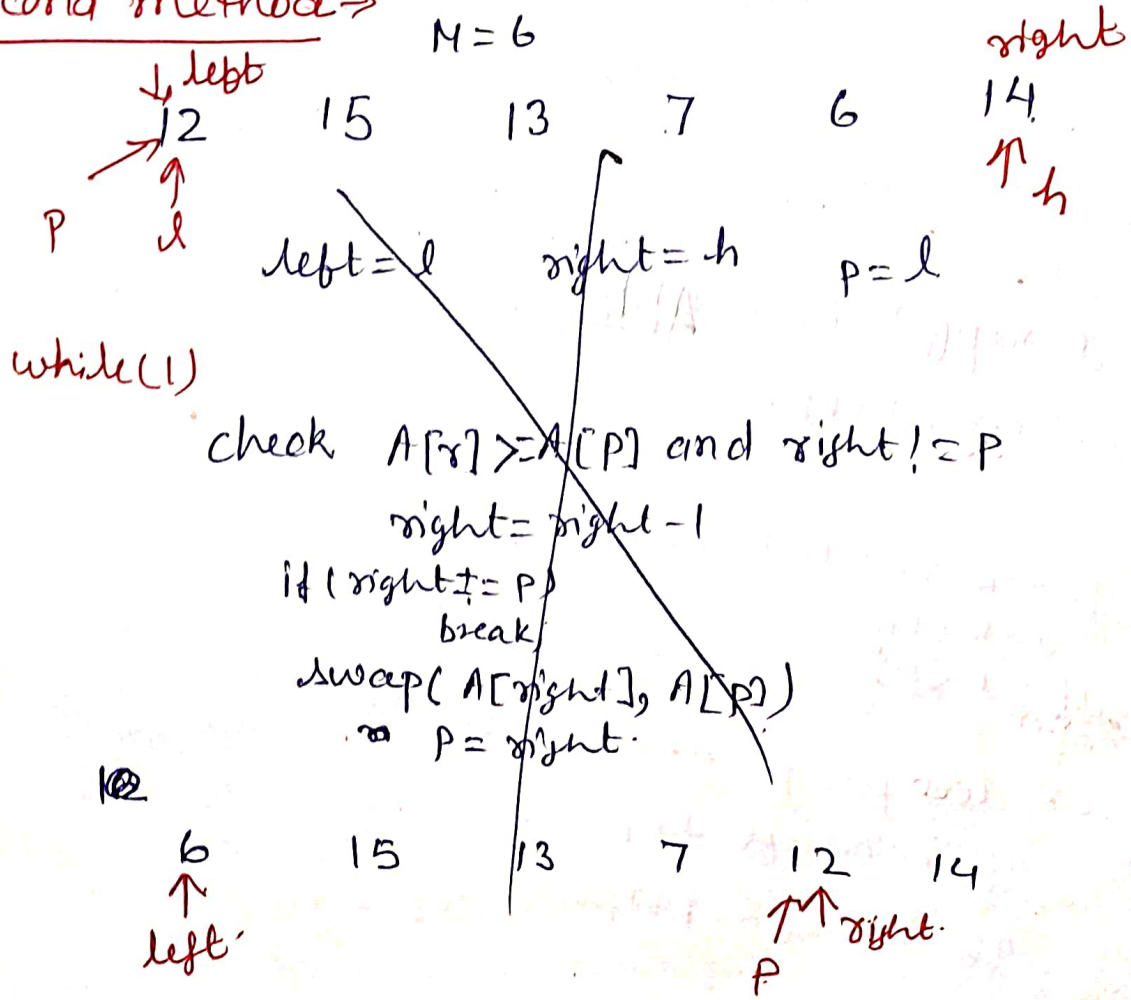
if (l < h)
   p = partition (A, l, h)
   Quicksort (A, low, p-1)
   Quicksort (A, p+1, h)
[End of if]

C program →

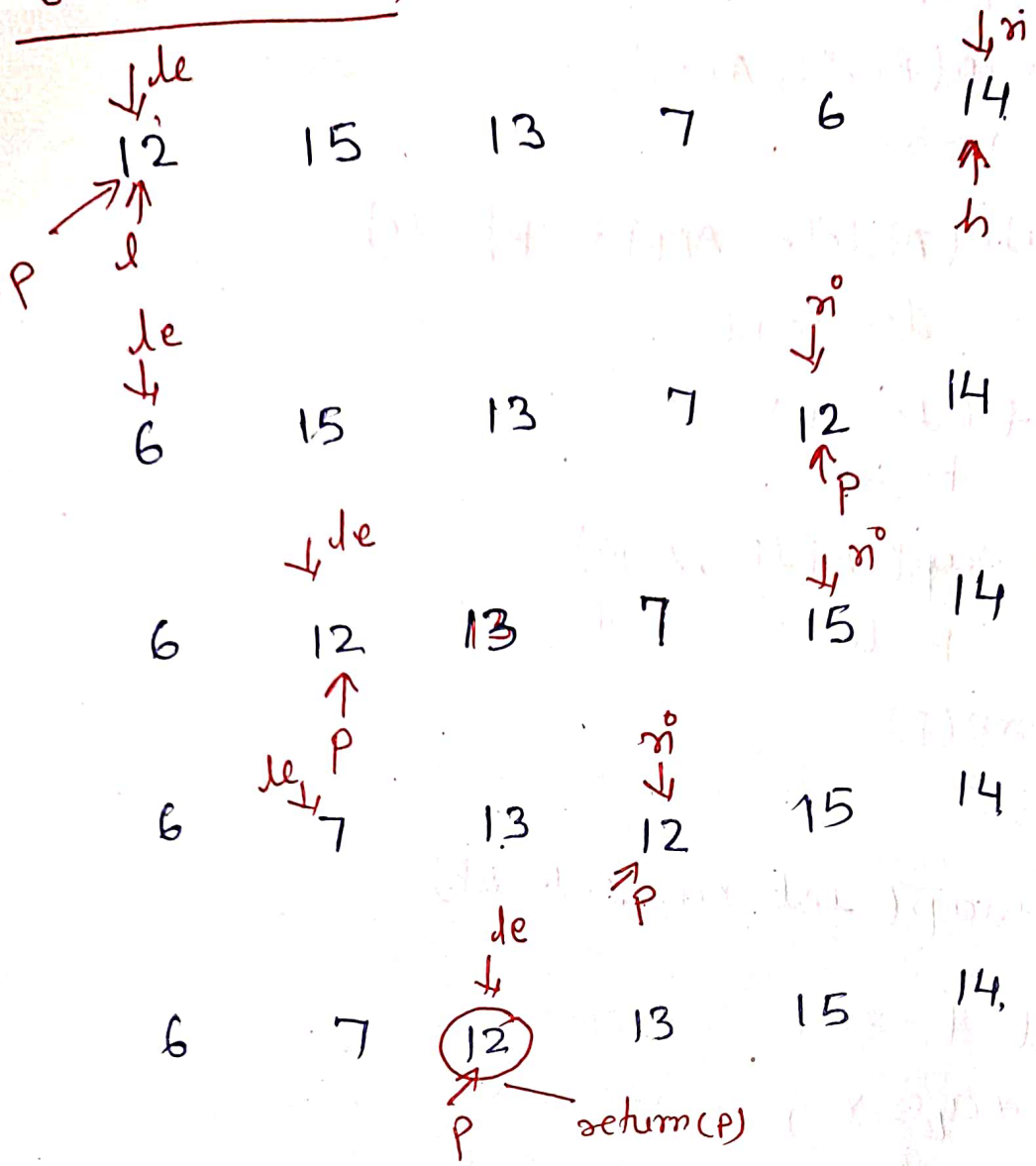
```
void swap(int *a, int *b)
{
    int t = *a;
    *a = *b;
    *b = t;
}
int partition(int A[], int l, int h)
{
    int p, i=-1, j;
    p = A[h];
    for(j=0; j<h; j++)
    {
        if(A[j] < p)
        {
            i++;
            swap(&A[i], &A[j]);
        }
    }
    swap(&A[i+1], &A[h]);
    return(i+1);
}
void quicksort(int A[], int l, int h)
{
    int p;
    if(l<h)
    {
        p = partition(A, l, h);
        quicksort(A, l, p-1);
        quicksort(A, p+1, h);
    }
}
```

```
int main()
{
   int A[30], i, n;
   printf("Enter size");
   scanf("%d", &n);
   printf("Enter array elements");
    for (i=0; i<n; i++)
     scanf("%d", &A[i]);
       quicksort(A, 0, n-1);
     printf("After sorting");
       for (i=0; i<n; i++)
         printf("%d", A[i]);
}
```

Second method →



Second method →



Algorithm

```
quick_sort( A, l, h)
 if ( l < h) then
 p = partition ( A, l, h)
  quick_sort (A, l, p-1)
  quick_sort ( A, p+1, h)

partition ( )
  le = l , ri = h , p = l
  while (1)
     while( A[ri] >= A[p] && p != ri)
        ri = ri - 1
```

```
if (ri == p)
    break
swap(A[ri], A[p])
    p = ri
while (A[le] <= A[p] && p != le)
        le = le + 1
if (li == p)
    break
swap(A[li], A[p])
    p = le
return(p)
```

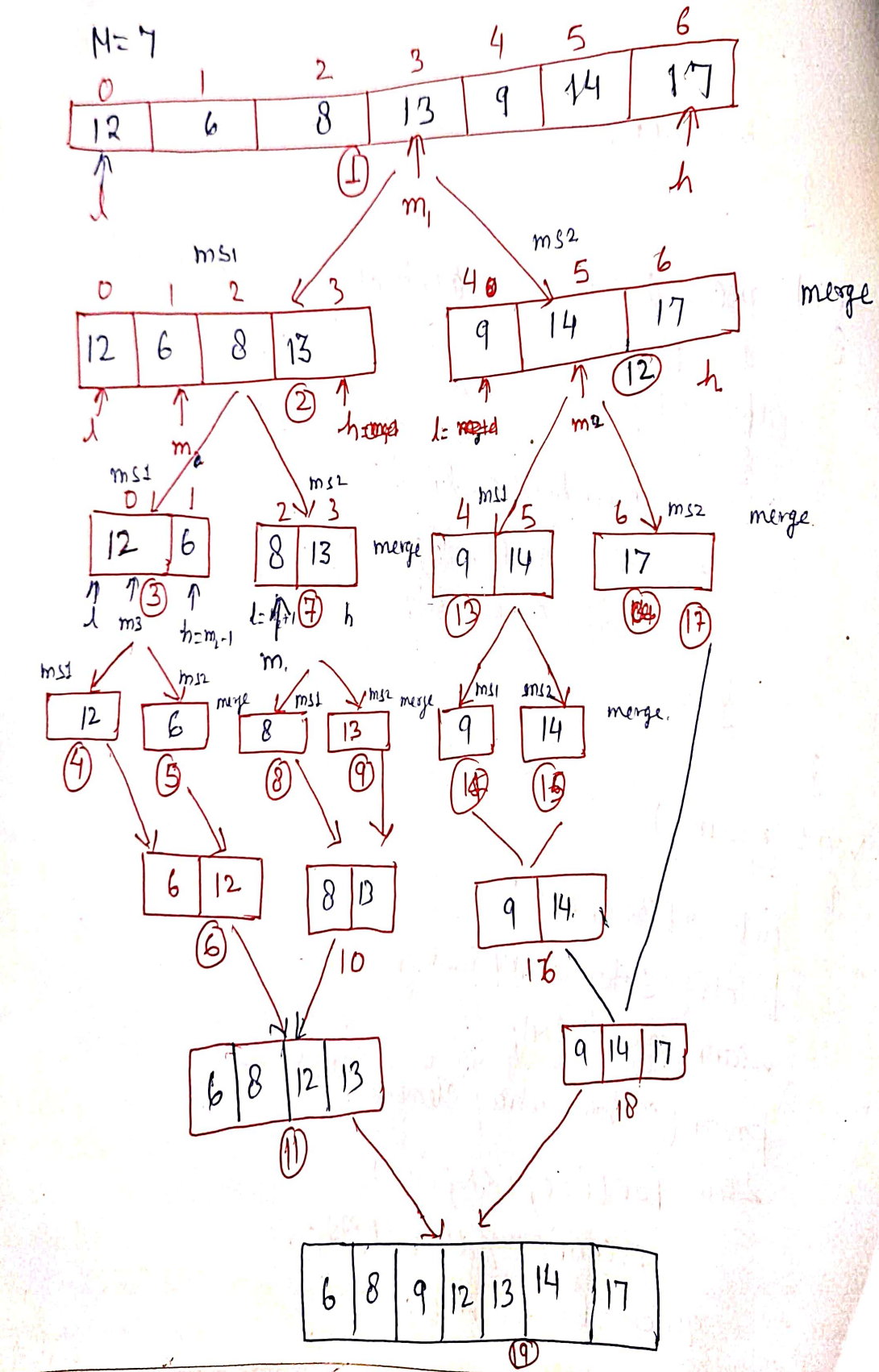
C program →

```
void swap(int *a, int *b)
{
    int t = *a;
      *a = *b;
     *b = t;
}
int partition(int A[], int l, int h)
{
    int le = l, ri = h, p = l;
    while(1)
    {
        while(A[ri] >= A[p] && p != ri)
            ri = ri - 1;
        if(p == ri)
            break;
            swap(&A[p], &A[ri]);
            p = ri;
```

```
    while( A[le] <= A[p] && p != le)
        le = le+1;
    if (p == le)
        break;
    swap(&A[p], &A[le]);
        p = le;
    }
    return(p);
}
void quicksort(int A[], int l, int h)
{
    int p;
    if(l < h)
    {
        p = partition(A, l, h);
        quicksort(A, l, p-1);
        quicksort(A, p+1, h);
    }
}
void main()
{
    int A[30], i, N;
    printf("Enter array size");
    scanf("%d", &N);
    printf("Enter array elements");
    for(i=0; i<N; i++)
        scanf("%d", &A[i]);
    quicksort(A, 0, N-1);
```

```
printf("sorted array is");
for(i=0; i<N; i++)
    printf("%d", A[i]);
}
```

mergesort →



Algorithm→

mergesort(A[], l, h)

1. ~~mergesort(A)~~ if (l < h)
2. m = (l + h)/2
3. call mergesort(A, l, m)
4. call mergesort(A, m+1, h)
5. merge (A, l, m, h)
6. Exit.

merge (A, l, m, h)

1. set i = l
   j = m+1
   k = l.
2. repeat ~~step~~ while (i <= m and j <= h)
   if (A[i] < A[j])
   temp[k] = A[i]
   k = k+1
   i = i+1
   else
   temp[k] = A[j]
   k = k+1
   j = j+1
   [End of if]
   [End of while]

```
while ( i<=m)
  temp[k]=A[i]
    k=k+1
    i=i+1 [end of while]
while [j<=h)
  temp[k]= A[j]
     k=k+1
     j=j+1 [End of while]
for (i=0 to k-1)
    a[i]=temp[i]
```

Exit.

c program →

```
void merge(int a[], int l, int m, int h)
{
    int temp[30], i, j, k;
     i=l
     j=m+1
     k=l
  while (i<=m && j<=h)
   {
       if (a[i]<a[j])
          temp[k++]=a[i++];
       else
          temp[k++]= a[j++];
   }
  while(i<=m)
     temp[k++]=a[i++];
```

```
    while(j<=h)
     temp[k++]=a[j++];
    for(i=0;i<k;i++)
     a[i]=temp[i];
  }

void mergesort(int a[], int l, int h)
  {
     int m;
     if(l<h)
     {
        m=(l+h)/2;
      mergesort(a, l, m);
      mergesort(a, m+1, h);
      merge(a, l, m, h);
     }
  }
int main()
  {
   int a[30], i, n;
   printf("Enter size of array");
   scanf("%d", &n);
   printf("Enter array elements");
     for(i=0; i<n; i++)
        scanf("%d", &a[i]);
     mergesort(a, 0, n-1);
    printf("After sorting array elements are");
     for(i=0; i<n; i++)
      printf("%d", a[i]);
  }
```

Radix sort:→ To sort decimal number where radix or base is 10, we need 10 packets. These packets are numberd 0, 1, 2, 3, ---, 9. numbers are stored from right digit to left digit.

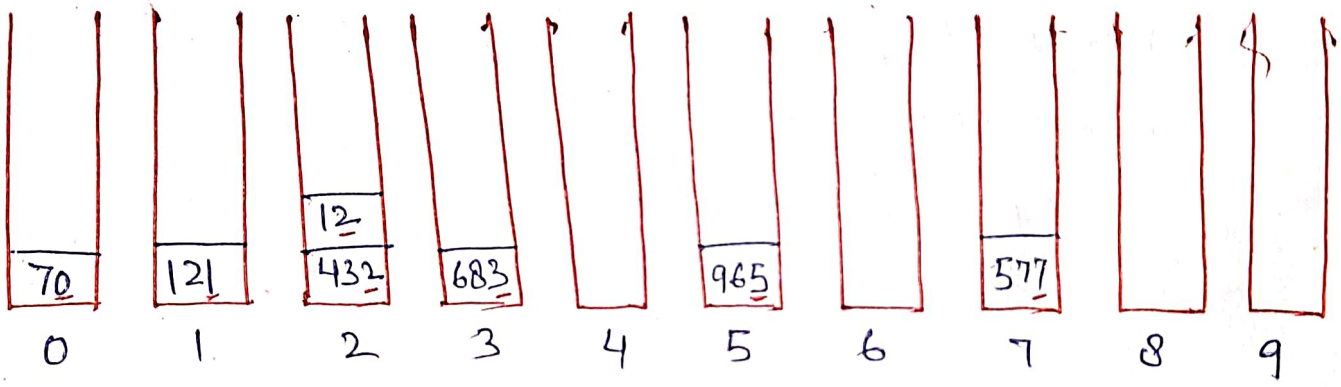
number of required passes = no. of digits in largest number.

* for storing names we need 26 packets.

Example- sort 121, 70, 965, 432, 12, 577, 683

largest no = 965

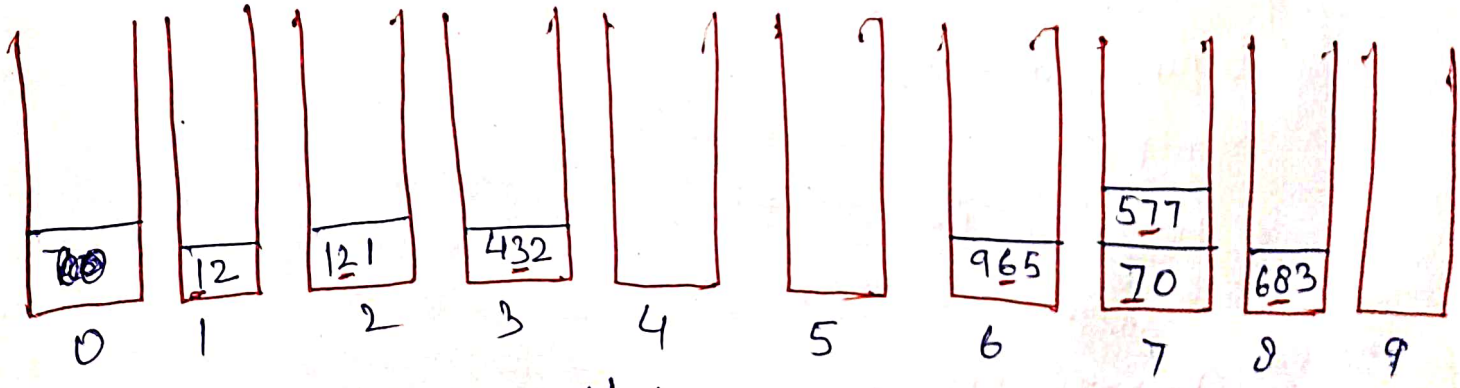
no. of pases required = 3. The number are stored in packets according to unit digit.



After 1st pass result array is

70, 121, 432, 12, 683, 965, 577

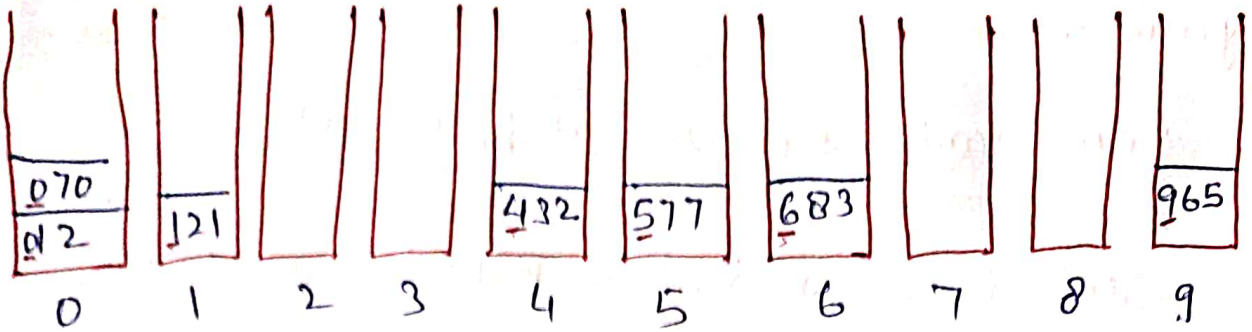
Now the numbers are stored according to Ten's digit.



After second pass resulted array is

012, 121, 432, 965, 070, 577, 683

Now the numbers are stored acording to hundred's digit.



After Third or last pass resulted array is

12, 70, 121, 432, 577, 683, 965

~~C program~~ Algorithm –

1. Find the largest element of the array.
2. Find the total number of digits in largest number. (n)
3. repeat steps 4 to 8 for pass = 1 to n
4. repeat step 4.1 for k = 0 to 9

   4.1 buck[k] = 0;
5. repeat step 5.1 for i = 0 to n-1

   5.1 l = (a[i]/div)%10

   pocket[l][buck[l]] = a[i]

   buck[l] = buck[l] + 1
6. i = 0
7. repeat step 7.1 for k = 0 to 9

   7.1 j = 0 and repeat 7.2 while j < buck[k]

   7.2 a[i] = pocket[k][j]

   i = i + 1.

   j = j + 1.
8. div = div * 10
9. Exit.

C program →

```
void radix_sort(int a[], int n)
{
  int t, i, j, max, temp[10][30], r[10], f[10];
  int k=0, count=0, d=1, s;
  // find max element
    max = a[0];

  for(i=1, i<n; i++)
      if(a[i]>max)
        max=a[i];
  // find no. of digits in max element
  while(max!=0)
    {
       count= count+1;
       max = max/10;
    }
// logic for passes.
   for(i=1; i<=count, i++)
    {
       for(s=0; s<10; s++)
         r[s] = f[s] = -1;
       for(j=0; j<n; j++)
        {
            t= (a[j]/d) %10;
          if(r[t]==-1)
          f[t]=r[t]=0;
        else
            r[t]=r[t]+1;
      temp[t][r[t]] = a[j];
        }
```

```
        d = d * 10;
    }   //End of i loop

for(i=0; i<10; i++)
  {
    for(j=f[i]; j<=r[i]; j++)
        if(f[i] != -1)
            a[k++] = temp[i][j];
  }

void main()
{
  int a[30], i, n;
  printf("Enter size");
  scanf("%d", &n);
  printf("Enter array elements");
    for(i=0; i<n; i++)
        printf("%d", a[i])
  radix_sort(a, n);
  printf("After sorting array is");
      for(i=0; i<n; i++)
          printf("%d", a[i]);
}
```

Pradeep Pant

Unit-3

Set-3

Hashing- ~~Alhash function~~ Hashing is the process of generating a value from a text from a text or list of numbers using a mathematical function known as hash function.

Suppose there are n elements which are to be stored in hash table of size M where $m \geq n$.

An element with key value K will be put in slot j of hash table if $\boxed{j = H(K)}$

where H[K] is a hash function.

It is possible that $H(K_1)$ may be same as $H(K_2)$. This is called collision. and K_1 and K_2 are called synonyms.

Different Hash function-

1) **Division method**- This is the most simplest and easiest method to generate a has value. The has function divides the value K by M and uses the remainder obtained.

$$\boxed{h(K) = K \bmod M}$$

where K = key value
M = size of hash table.

example→ K = 12345 M = 95

$h(K) = 12345 \bmod 95 = 90$

K = 1276 ~~bbb~~ M = 11 $h(K) = 1276 \bmod 11 =$ ~~0000~~ 0

Advantages:- This method is very fast since it requires only a single division operation.

Disadvantage:-

1) This method leads poor performance since consecutive keys map to consecutive has value in has table.

2) Extra care should be taken to choose the value of M.

2) Mid square method - It involves two steps to compute hash value.

1) square the value of key (k^2)

2) Extract the middle r digits as the hash value.

The value of r can be decided based on the size of a Hash Table.

Example → 1) let the hash table has 100 memory locations. and key = 60. Caculate Hash value.

~~Solution~~

Ans:- M = 100 So r = 2.

$k \times k = 60 \times 60 = 3600$

$h(60) = 60$

Advantages:- 1) The performance of this method is good as most or all digits of the key contribute to the result.

2) The result is not dominated by the distribution of the top digit or bottom digit of the key.

Disadvantages:- 1) The size of the key is limitation, as the key is of big size then its square will double the number of digits.

2) There will be collisions.

3) Digit Folding method - This method involves two steps:

1) Divide the key-value K into number of parts, i.e. $K_1, K_2, K_3, \ldots, K_n$ where each part has the same number of digits except for the last part that can have lesser digits than the other parts.
2) Add the individual parts. The hash value is obtained by ignoring the last carry if any.

example→ $K = 12345$

$K_1 = 12 \quad K_2 = 34 \quad K_3 = 5$

$S = K_1 + K_2 + K_3 = 12 + 34 + 5 = 51$

$H(K) = 51$

(4) Truncation method - In this method we take only part of the key as address, it can be some rightmost digit or leftmost digit. part may depend on size of Hash table.

ex→ $K = 12345 \quad m = 100$ then we take either left two digits or right two digits of key.

$H(K) = 12$ or 45

5) Multiplication method :-

1. Choose a constant A such that $0 < A < 1$
2. multiply K with A
3. Extract the fractional part of KA
4. multiply the result of step 3 by the size of Hash Table (M).
5. The resulting hash value is obtained by taking the floor of the result obtained in step 4.

Example- $K = 123$ $m = 100$

$A = 0.35$

1.

1. $A = 0.35$
2. $K \times A = 0.35 \times 123 = 43.05$
3. fractional part $= 0.05$
4. $M \times$ fractional part $= 100 \times 0.05 = 5$
5. floor value $= 5$

$H(123) = 5$

Collision conflict resolution techniques:-

There are two methods for conflict resolution

(i) open chaining Addressing.

(ii) Separate chaining.

1) ~~Open chaining~~ open Addressing :→ In open addsering, ~~addelements~~ an empty position is found out within the hash table itself and the new key is inserted in that empty position.

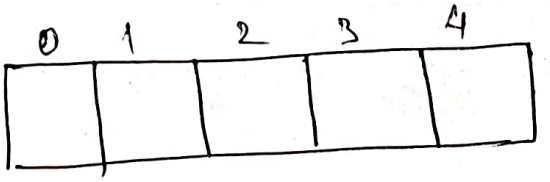
Different types:- ~~of open Addsering~~

(a) Linear probing:- In linear probing, the hash table is searched sequentially that starts from the original location of the hash. If in case the location that we get is already occupied, then we check for next location.

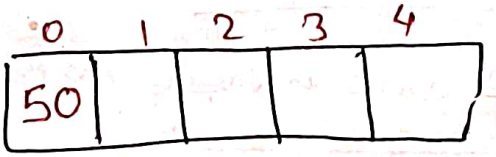
Example →

M = 5 keys = 50, 70, 76, 93

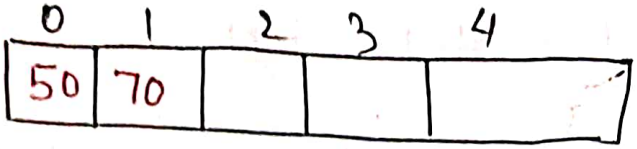
1. Draw empty Hash Table



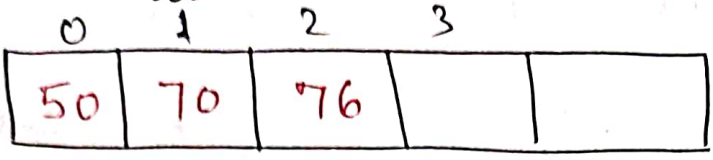
2. first key is 50 compute ~~HK~~ K mod N = 50 % 5 = 0. So insert into slot no. 0.



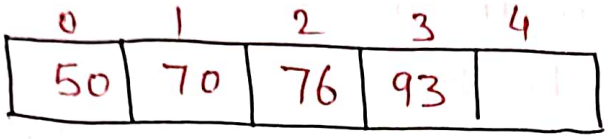
3. next key is 70. compute K mod N = 70 % 5 = 0. But 50 is already to slot no 0, so search for the next empty slot and insert it.



4. next key is 76. $76\%5 = 1$ but slot no 1 is already occupied by 70. so search for next empty slot and insert it.



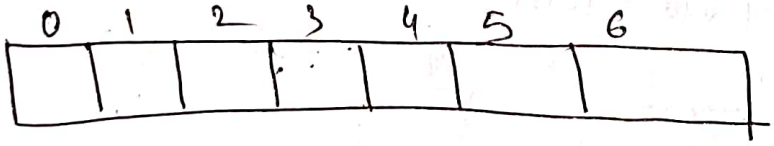
5. next key is 93. $93\%5 = 3$. slot 3 is empty. Insert 93 at slot 3.



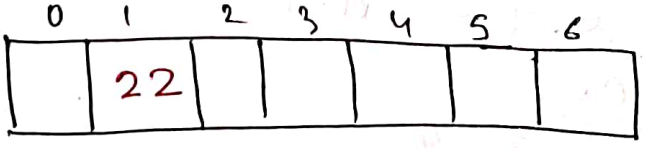
2. Quadratic probing :- In this method, we look for the i^{th} slot in the i^{th} iteration. We always starts from the original hash location. If only the location is occupied then we check the other slots.

example → $m = 7$ $K = 22, 30, 50$

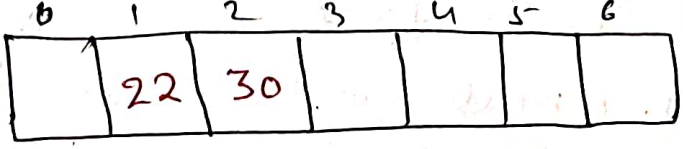
1.



2. $K = 22$ $22\%7 = 1$ slot 1 is empty, insert 22 at 1.

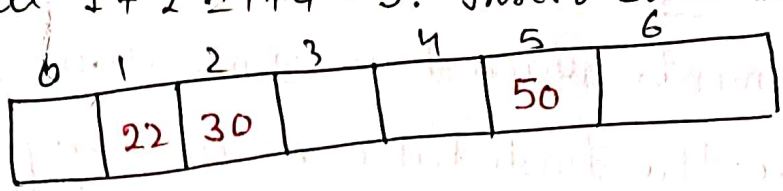


3. $K = 30$ $30\%7 = 2$ slot 2 is empty; insert 30 at 2.



4. $K = 50$ $50\%7 = 1$ slot 1 is not empty. So we will search for slot $1 + 1^2 = 2$

Again slot 2 is found occupied, so we will search for cell $1+2^2 = 1+4 = 5$. Insert 50 at 5.

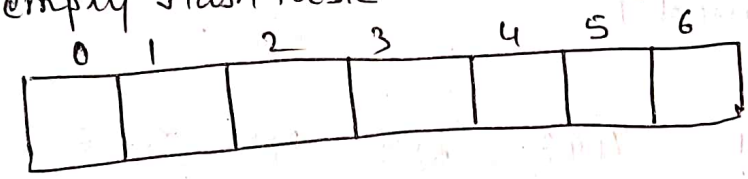


3) Double hashing :- In this technique, the increment for the probing sequence are computed by using another hashing function. We use another hash function hash2(x) and look for the $i \times hash2(x)$ slot in i^{th} iteration.

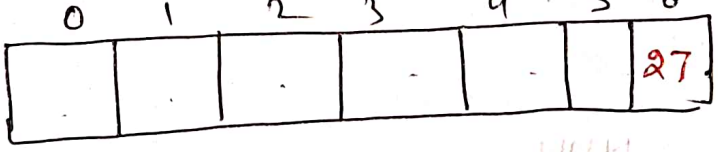
Example :- M=7 Key=27, 43, 92, 72

$H_1(K) = K \bmod 7$ $H_2(K) = K \bmod 5 + 1$

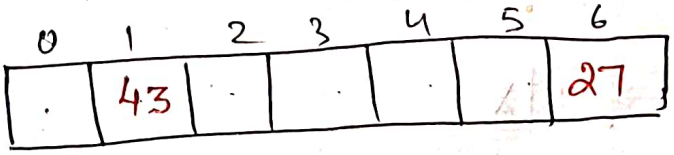
1. Create empty HashTable



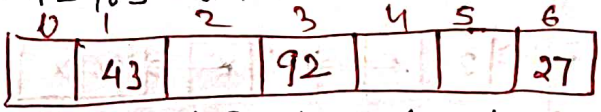
2. Key=27 Compute 27%7 = 6. 6 is empty. insert 27 into 6.



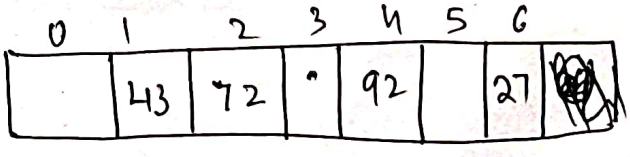
3. Key=43 43%7 = 1 slot 1 is empty. Insert 43 in slot 1.



4. Key = 92 92%7 = 1. slot 1 is not empty. so apply second Hash function $1 + 92\%5 = 1+2 = 3$. slot 3 is empty. Insert 92 in slot 3.



5. Key = 72 72%7 = 2. slot 2 is empty. insert 72 in slot 2.



Separate chaining:- This method maintains the chain of elements which have same hash address.

We can take the hash table as an array of pointers

Here each pointer will point to one linked list and elements which have same hash address will be maintained in the linked list

Nodes are stored outside the space of hash table so this is called separate chaining.

example- 351, 493, 251, 71, 706, 146

$h(k) = k \bmod 7$

Key	351	493	251	71	706	146
h(k)	1	3	6	1	6	6