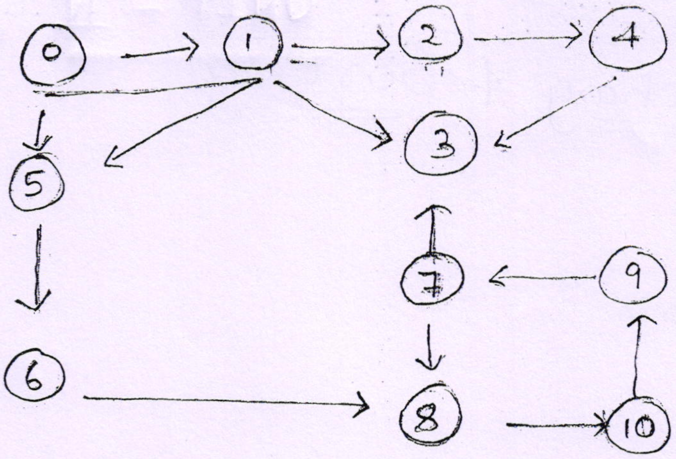
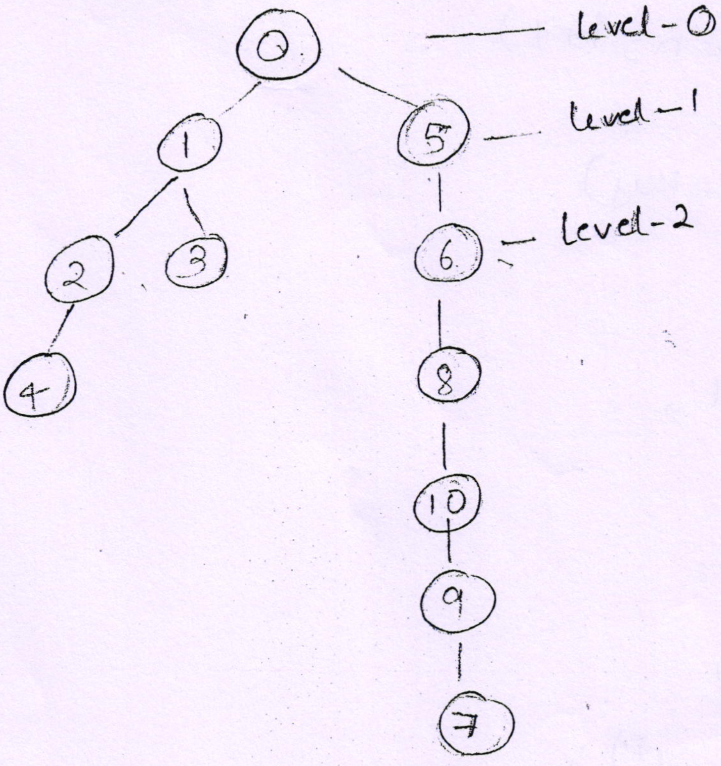


11 Vertices

Edge

v_1 v_2

0 - 1
0 - 5
1 - 2
1 - 3
1 - 5
2 - 4
4 - 3
5 - 6
6 - 8
7 - 3
7 - 8
8 - 10
9 - 7
10 - 9

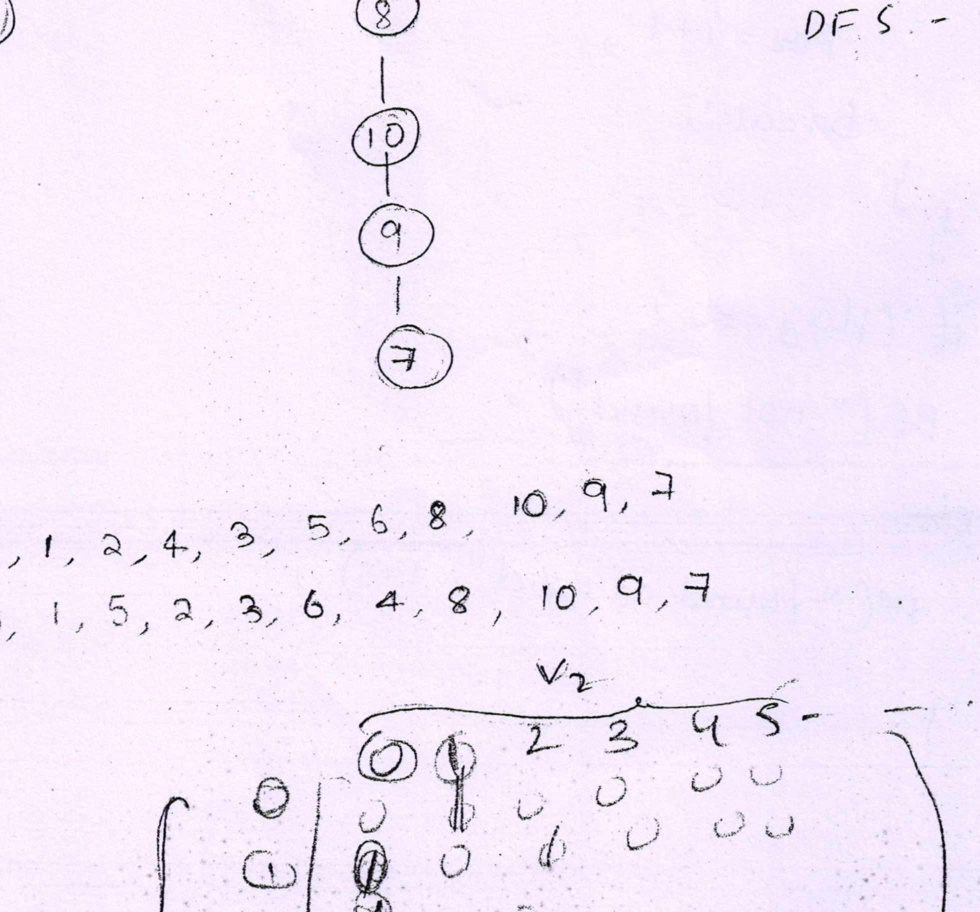


BFS - Q

DFS - S

DFS :- 0, 1, 2, 4, 3, 5, 6, 8, 10, 9, 7

BFS :- 0, 1, 5, 2, 3, 6, 4, 8, 10, 9, 7



20/8/16

UNIT 4

Searching & Sorting.

Linear search.

```
# include <stdio.h>
void main() {
 flag=0;  pos;
 for (i=0; i<n; i++)
 {
  if (A[i]==key)
   {
    flag=1
    pos=i+1;
   break;
  }
 }
if (flag==0)
 pf("not found");
else
 pf("found at=%d", pos);
}.
```

Binary

[In th

are

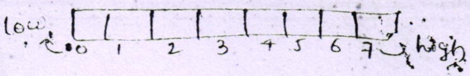
whi

or

Value

```
#incl
void
flag=c
low=c
~~found~~

if

else i

else

if (if

else

}
```

Binary Search:



[In this we consider lower bound and high bound and then calculate the mid value, and check whether the key is towards left array or right array and then change the mid value accordingly].

```
#include <stdio.h>
void main(){
flag=0; pos;
low=0; high=n-1;
      while (low <= high)
      {
      mid = (low+high)/2;
    if (a[mid] == key)
      { flag=1;
           pos=mid+1;
           break;
      }
else if (key > a[mid])
      low = mid+1;
else
      high=mid-1;
if (flag==0)
      pf("not found");
else pf(" found at=%d", pos);
  }
```

23/8/16

Hashing UNIT – IV [4]

Hash Table : This is a data structure which stores, has values associated with a hash key.

for example:

if we wanted to store & access employee records with attributes employee no:, name, designation,

Then we consider one of the field like as key field like employee no: for above record.

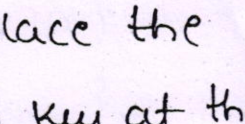
Hash function takes this key as an i/p & produces an hash key which will be used as an index in to the hash table. to place the corresponding record of the given key at that location.

Same procedure is used to retrive or Search a record with a given key.

This process can locate a given record with one (or) few searches.

Hash tables are used to implment, Dictionaries, consisting of key, value pair.

Emno no key → HF (Key) → Hash key →



Hashing Methods / functions

1) Division Method
2) Mid Square Method
3) Multiplication Method
4) Folding Method.

Division Method :

- In this method, we divide the key with the Hash table size, which gives the address of the key.

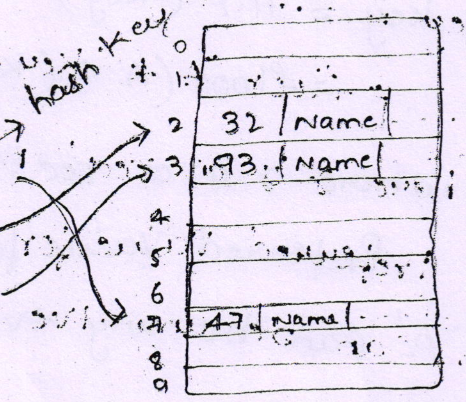
Ex 1:-

Let the H.T size = 10

key = 47 $47 \% 10 = ⑦$

key = 32 $32 \% 10 = 2$

key = 93 $93 \% 10 = 3$



Ex 2:- Let H.T Size = 100

Key = 12475 $12475 \% 100 = 75$

Mid Square Method

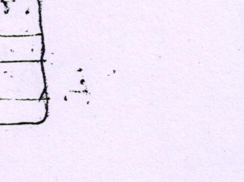
We ~~used~~ Square the key value and then find the mid number of the squared value, which will be the address of the key.

Ex:- key = 3036

HF (key) $= (3036)^2 = 921⑦296$

If H.T size is 10

then address is 7



1234

Ex 2 :- 92<u>17</u>296

If HT size is 100 then address/index is either 17 or 72

[for 7 we can consider left no. or right number but it should be same for all the values]

Ex 3 :-

92<u>172</u>96

If HT Size is 1000. (0-999)

Then 172 is the Hash key

Multiplication Method :-

Hash key = H.F (Key)

= floor (A * (Key * r))

where r is a real number.

Preferred value for r = 0.61803

'A' can be any +ve int Ex = 5 (but should be same for all the values)

Ex :-

key = 25

Hash key = floor(5 * (25 * 0.61803))

= floor(5 * 15.451)

= floor (77.255)

= 77.

$$\left.\begin{array}{l}\text{floor }(3.7)\\ \text{floor }(3.1)\end{array}\right\} = 3 \qquad \begin{array}{l}\text{round }(3.7) = 4\\ \text{round }(3.1) = 3\end{array}$$

$$\left.\begin{array}{l}\text{Seal }(3.7)\\ \text{Seal }(3.1)\end{array}\right\} = 4$$

Fold

[-

Key

Let

Hash

Collis

If th

for

Coll

Chara

1) It s

2) It s

no c

3) The

Key

4) It

Collision

1) Chain

2) line

3) quad

4) Do

Folding Method :

[fold & Sum]

12.

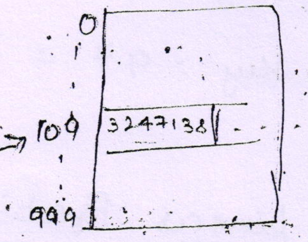
Key = 3247138

let the HT Size is 1000 (0-999 indexes)

Hash key = 324|713|8

$$\begin{array}{r} 324 \\ 713 \\ +\ \ \ 8 \\ \hline 104|5 \end{array}$$

104 + 5 = 109



Collision :

If the hash function producing same hash key for two different key values then it is called collision.

Characteristic of a good hash function :-

ld be
ir all, the
values)

1) It should be easy/simple to compute.
2) It should generate very few collisions, ideally no collisions at all
3) The function should evenly distribute, hash keys, across the hash table
4) It should operate on every bit of the i/p key

Collision handling Methods :

1) Chaining
2) Linear Probing
3) quadratic Probing
4) Double hashing

Chaining :

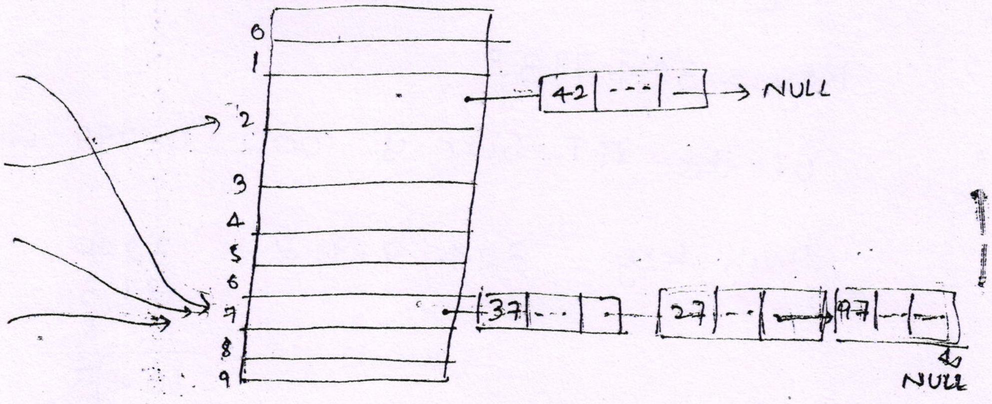
Method = Division Method .

Key : 37 = 7

Key = 42 = 2

Key = 27 = 7

Key : 97 = 7



Linear Probing

Method = Division method .

Key = 37 = 7

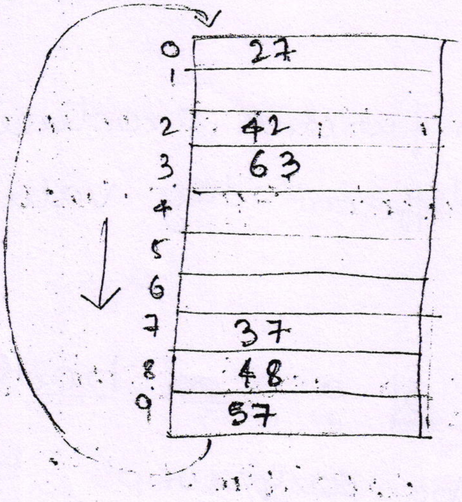
Key = 42 = 2

Key = 48 = 8

Key = 57 = 7

Key = 63 = 3

Key = 27 = 7



Quadratic Probing

If collision occurs for given key

Then

$$\text{Hash Key} = (\text{Key} + x^2) \,\%\, \text{H.T Size}$$

where $x = 1, 2, 3, \ldots$

Ex : 1

key = 37 if collision occurs ,

Step 1 :- Take $x = 1$ Hash Key = $(37+1^2) \% 10 = 38 \% 10 = \boxed{8}$

[If this index is free use this else go to next 'x' value]

Take $x = 2$ Hash key = $(37+2^2) \% 10 = 41 \% 10 = 1$

Take $x = 3$

⋮

Double Hashing :

i) HF1 (key) = hash_key

HF 2 (key) = M - hash key = P

7

Now jump forward by P locations from collision index and place the record there

Key = 34

HF(34) = 34 % 10 = 4

Choose M as the biggest prime no less than size of the table

if size = 10

M = 7

HF 2(34) = |M - 4|

= |7 - 4|

= 3

collision index = 4

P = 3

jump to 7th index & place the record.

Rehashing :-

- The size of the hash table will be doubled to its nearest intiger prime number, whenever, either the hash table becomes full (or overflow occurs) or the hash function producing more collisions.

17/9/16

Sorting Techniques

BUBBLE SORT

```
for (i=1 ; i<n ; i++)
{
 for (j=0 ; j<n-1 ; j++)
  {
    if (A[j] > A[j+1])
    {
      temp = A[j]
       A[j] = A[j+1]
        A[j+1] = temp.
    }
  }
}.
```

SELECTION SORT :-

Eg :-

10	13	7	53	62	69	4	46	35	12
0	1	2	3	4	5	6	7	8	9

Step 1

10 13 7 53 62 69 4 46 35 12

7 13 10 53 62 69 4 46 35 12

7 13 10 53 62 69 4 46 35 12

7 13 10 53 62 69 4 46 35 12

7 13 10 53 62 69 4 46 35 12

4 13 10 53 62 69 7 46 35 12

4 13 10 53 62 69 7 46 35 12

4 13 10 53 62 69 7 46 35 12

4 13 10 53 62 69 7 46 35 12

Step 2 :- ×$\boxed{4}$ 13 10 53 62 69 7 46 35 12

⋮

4 7 13 53 62 69 10 46 35 12

Step 3 :- ×$\boxed{4}$ ×$\boxed{7}$ 13 53 62 69 10 46 35 12

⋮

Selection Sort

```
for (j=0 ; j< n-1 ; j++)
for (K =j+1 ; k<n ; k ++)
  {
      if [ A[j] > A(k)]
         {
            temp = A[j] ;
            A[j] = A[k] ;
            A[k] = temp ;
         }
  }
```

12

12

12

12

12

Insertion Sort :

10	13	7	53	62	69	4
0	1	2	3	4	5	6

n = 7

```
while (k != n)
{
  for (i = k ; i <= 0 ; i--)
  {
    j = i + 1
    if (A[i] > A[j])
    {
      swap;
    }
    k++ ;

  for (i = 1 ; i < n ; i++)
  {
    j = 1 ;
    while (j > 0 && [j] < A[j-1])

    { swap;
      j--;

    }
  }
```

Double

HF1 (k

HF2 (

Now

i

Key = 3

HF(34)

Choo

bl

Rena

- The

nearus

the h

occur

collis

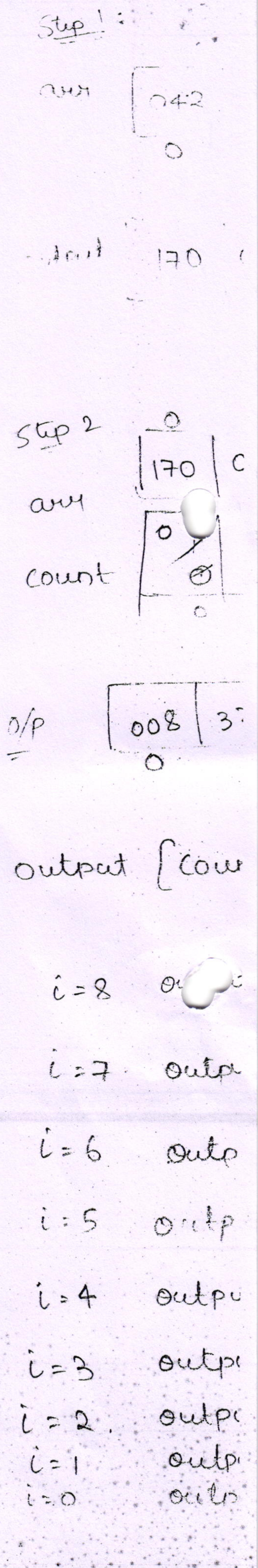
Radix Sort :-

```
# include <stdio.h>
int get Max (int arr [], intn) {
 int mx = arr [0];
 int i;
 for(i=1; i<n; i++)
   if (arr[i]>mx)
     mx = arr [i];
  return mx;
}
Void Countsort (int arr [], int n, int exp)
{
 int Output [n]; //output array.
 int i, count [10] = {0};
 // Store count of occurrences in count[].
  for (i=0; i<n; i++)
   Count[(arr[i] /exp) % 10] ++;
  for (i=1; i<10; i++)
    Count[i] += Count [i-1];
 // Build the output array.
 for (i= n-1; i >=0; i--)
 {
   output [count[(arr [i] /exp) % 10]-1] = arr [i];
   count [(arr[i] /exp) % 10]--;
 }
```

a+=b
a=a+b

count [i] = Count [i] + Count[i-1]

```
for (i=0 ; i<n ; i++)
  arr [i] = output [i];
}

// The main function to that sorts arr[] of size n
using Radix Sort .

void radixsort (int arr[], int n){
  int m = getMax (arr,n);
 int exp;
 for (exp=1; m/exp>0; exp*=10)
   countSort(arr,n, exp);
 }
void print (int arr[], int n)
{
 int i;
for(i=0; i<n;i++)
 Printf("%d", arr[i]);
}
t main()
{
int arr[] = {170, 45, 75, 90, 802, 24, 2, 66};
int n = sizeof(arr)/sizeof(arr[0]);
 radixSort (arr,n);
 print(arr,n);
 return0;
}
```



Step 1:

arr	042	170	323	894	666	047	449	179	008
	0	1	2	3	4	5	6	7	8

ar[] of size n

output	170	042	323	894	666	047	008	179	449

Step 2

	0	1	2	3	4	5	6	7	8
arr	170	042	323	894	666	047	008	179	449

count	0 ~~1~~ ~~0~~	1 ~~0~~	1 ~~2~~ ~~0~~	2 ~~0~~	2 ~~3~~ ~~1~~ ~~4~~ ~~0~~	5 ~~0~~	5 ~~6~~ ~~0~~	6 ~~7~~ ~~8~~ ~~0~~	8 ~~0~~	8 ~~9~~ ~~0~~
	0	1	2	3	4	5	6	7	8	9

O/P	008	323	042	047	449	666	170	179	894
	0	1	2	3	4	5	6	7	8

output [count [(arr[i]/exp) % 10] - 1] = arr [i]

↳ last but 1 digit

i = 8 output [count [4] - 1] ⇒ op [5-1] = op[4] = arr [8]

i = 7 output [count [7] - 1] ⇒ op [8-1] = op [7] = arr [7]

i = 6 output [count [0] - 1] ⇒ op [1-1] = op [0] = arr [6]

56};

i = 5 output [count [4] - 1] ⇒ op [4-1] = op[3] = arr [5]

i = 4 output [count [6] - 1] ⇒ op [6-1] = op [5] = arr [4]

i = 3 output [count [9] - 1] ⇒ op [9-1] = op [8] = arr [3]

i = 2. output [count [2] - 1] ⇒ op [2-1] = op[1] = arr [2]

i = 1 output [count [4] - 1] ⇒ opt [3-1] = op [2] = arr [1]

i = 0 output [count [7] - 1] ⇒ opt [7-1] = op [6] = arr [0]

Step 3:

	0	1	2	3	4	5	6	7	8
arr	008	323	042	047	449	666	170	179	894

	0	1	2	3	4	5	6	7	8	9
Count	3 ~~0~~	~~2~~ ~~0~~	0 ~~0~~	1 ~~0~~	1 ~~0~~	0 ~~0~~	1 ~~0~~	0 ~~0~~	1 ~~0~~	0 ~~0~~

	0	1	2	3	4	5	6	7	8
O/P									

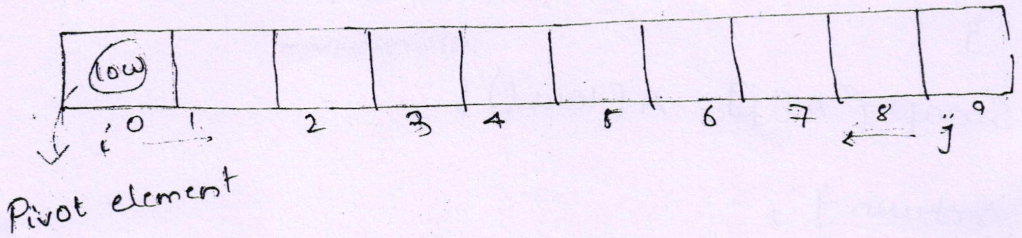
Output = [count[(arr[i]/exp) % 10] - 1] = arr[i]

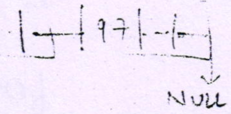
i = 8   output [Count[8] - 1] ⇒ op [

```
Radix Sor
# include
int get
 int mx
 int i;
 for(i=1;
   if (a
     mx
   ret  ur
 }
void cou
{
 int out
 int i,
 // Sto
 for(
   cou
 for(
   co
 // Buil
 for(i
 {
   out
   cou
 }
```

27/9/16

Quick Sort :-





Pivot = A[low]

Increment i such that it should be greater than the Pivot element A[i] ≤ Pivot

Move j towards left (decrement) side that the value should be less than the Pivot element A[j] > Pivot.

After the increment of i & j value

If i < j (index values)

Swap the elements

If j < i then Swap A[j] with Pivot value or A[low]

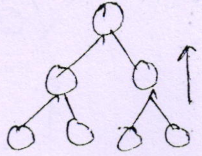
```
# include <stdio.h>
int Partition (int A[10], int low, int high)
{
 int Pivot = A[low], i, j ;
  i = low;
  j=high;
 while (i ≤ j)
 {
  while (A[i] ≤ Pivot)
      i++ ;
  while ( A[j] > Pivot)
      j - - ;
```

(8)
this index
ee 'use this
go to
next 'x'
value

```
    if (i<j)
     swap ( A[i], A[j]);
     }
   swap ( A[j], A[low]);
    return j;
    }
void quicksort (int A[10], int low, int high)
{
    int k;
   if (low < high)
   {
   k = partition (A, low, high);
   quicksort (A, low, k-1);
   quicksort (A, k+1, high);
    }
    }
   void  swap (
   {
     ___
     ___
     ___
    }
    void main ( )
    {
      int A[10];
      for (i=0; i<j; i++)
      pf
      sf
      read A.
      display A
       quicksort (A, 0, 9);
       display A.
    }
```

29/9/16

Heap Sort :

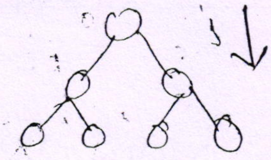


```
Void makeheap [int A[10], int n)
{
 int i, val, j, Parent ;
 for(i=1; i<n; i++)                 Here i : index value
 { val = arr[i];
   j = i ;
  Parent = (j-1)/2 ;
  while (j>0 44  A[Parent] < val)
   {
      A[j] = A [Parent];
          j = parent ;
      Parent = (j-1)/2..
    }
     A[j] = val;
   }
 }
```

n)

8	7	10	26	16	45	89	43	69	39	54	17
0	1	2	3	4	5	6	7	8	9	10	11

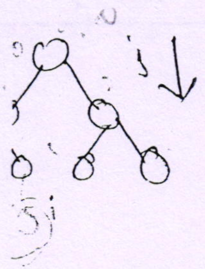
```
Void heapsort [int A[10], int n)
{
 int i, k, temp, j;

 for (i=n-1; i>0; i--)
 {
  temp = A[i];
  A[i] = A[0];
  k=0;     k = Parent index
           j = Child index
  if (i==1)
  j = -1;
 else
   j =1;
 if (i>2 && A[2] > A[1])
 j= 2;
While (j>=0 && temp < A[j])
{
 A[K] = A[j];
 k=j;
 j = 2* K+1; (going to left child)
if (j+1 <= i-1 && A[j]< A[j+1])
 j++;
 if (j>i-1)
    j= -1;
}
```

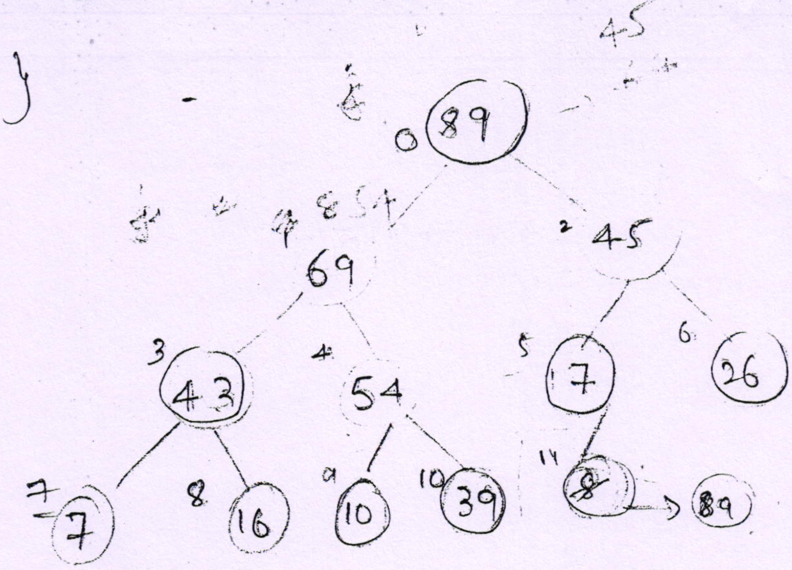


Here i = no. of elements to delete. (and) also for no. of elements it should consider for each iteration.

A[k] = temp;

}





climents to

and)

or no. of

it should

for each

n.

~~Step 1~~

i = 11 ; 11 > 0

temp = A[i] = 8

k = 0

A[i] = 89 ; j = 1

45 > 69 ×

11 > 2

8 < 69 ✓

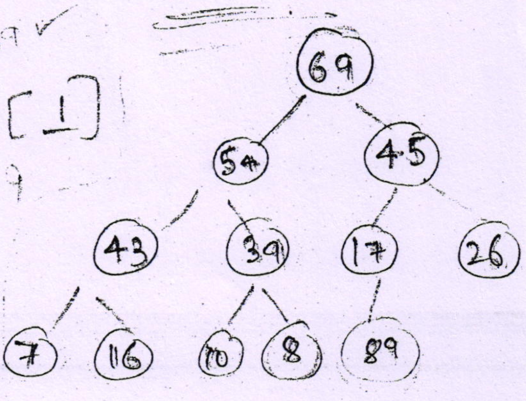
i > = 0

8 < 69 ✓

A[0] = A[1]

A[0] = 69

K = 1



After Step 1

~~j = 0~~

j = (2 * 1) + 1

= ~~2 * 2~~

j = 3

~~j = 10~~

A[3] < A[4]

~~54 < 17~~

43 < 54 ✓

K = 1 j = 4

4 > 11 - 1 ×

A[9] < A[