Date:
Page:

• Linear Search :-

The simplest of all the searching is linear or sequential search. Sequential searching is nothing but searching element in linear way, we have a need to start the search from beginning and search the element one by one until the end of array or linked list. If search is succesfull then it will return the location otherwise it will return the failure notification.

	0	1	2	3	4
eg →	30	60	90	12	80

Key element → 12

Comparing key element with first element of array.

1. a[0] == Key
   30 == Key ✗

2. a[1] == Key
   60 == 12 ✗

3. a[2] == Key
   90 == 12 ✗

Date:
Page:

4. a[3] = = Key
12 = = 12

search successfull element found at position 4.

Q WAP to search element in an array using linear search technique.

```
# include < stdio.h>
# include < Conio.h>
void main ()
{
int a[10], n, Key, flag=0, i;
clrscr ();
printf ("Enter how many element
            you want in array");
scanf ("%d", &n);
printf ("\n Enter element ");
  for (i=0; i<n; i++)
  {
   scanf ("%d", &a[i]);
  }
printf ("Element of array ");
  {
   printf ("%d", a[i]);
  }
printf ("Enter element to be
         searched");
```

```
scanf("%d", &key)
{
    if (a[i] == key)
    {
        flag = 1;
        break;
    }
}
if (flag == 1)
printf("Element found at
        position %d", i+1);
else
{
printf("\n Element not found
        in array");
}
getch();
}
```

Algorithm for linear search.

Here we represent an unsorted array of element and n represent number of element and key element represent the value to be searched for in the array.

Step-1 [Initialize]
$i = 0$, $flag = 0$

Step-2 → Repeat step III for
i = 0, 1, 2, 3, 4, 5 ... n.

Step-3 if (a[i] == Key), then
flag = 1
printf ("Element is found at %d", i+1);

STOP;
end if

Step-4 if (flag == 0), then
printf ("Element was not found in the array");
end if
Stop

Binary Searching

A unsorted array is searched by linear search that scan the array element one by one until the desired element is found.
The reason for sorting the array is that we can search the array. Now if the array is sorted we can employ binary search which brilliantly divide the list into two part. The size of the search space is halved

Date:
Page:

each time it examine one array element

An array based binary search selects the middle element in the array and compar it value to that of the key element, because the array is sorted if the key is less than the middle element in the array then the key must be in the first half of the array. Likewise if the value of key element is greater than that of the the middle value in the array then the key lies in the second half of the array.

In ideal case we infact check one half of the search space or array with only one comparison.

So, the algorithm narrows the search area by half at each time until it has either found a key or search fails.

As the name suggests binary search divide the array into halves. This search is applicable only to orderd list.

Date:
Page:

Q WAP to search a key element in an array using a binary search technique.

```
#include <stdio.h>
#include <conio.h>
void main()
{
int a[20], n, i, key, low, high, mid,
        flag=1;
clrscr();
printf("\n Enter the element you
        want to store in array");
scanf("%d", &n);
low =0;
high= n-1;
printf(" Enter the element in
        sorted order.");
for(i=0; i<n; i++)
{
  scanf("%d", & a[i]);
}
printf("\n Enter searched element");
scanf("%d", &key);
while (low <= high)
{
  mid= (low + high)/2;
  if (a[mid] == key)
{
```

```
flag = 1;
break;
}
else if (key > a[min])
{
low = mid + 1;
}
else
{
high = mid - 1;
}
}
if (flag == 1)
{
printf("\n Element found at poisition %d", mid + 1);
}
else
{
printf(" Element is not found in array or list");
}
getch ();
}
```

Difference between linear Search and binary search

Linear Search	Binary Search
1. It is Sequential type of Search	1. It is divide and Conquer approch
2. The element can be searched if the list of element are in unsorted order.	2. Necessary and mandatory condition is that the element should be in Sorted order i.e ascending or descending order
3. Linear Search is efficient if the list contain his element	3. Binary search is efficient if list Has large number of elements.

Sorting

i) Bubble Sorting :-

In this sorting algorithm, multiple swapping take place in one pass.

smaller elements move or 'bubble' up to the top of the list, hence the name given to the algorithm.

In this method, adjacent members of list to be sorted are compared. If the item on top is greater than item immediately below it, then they are swapped. This process is carried on till the list is sorted.

Q We have following N numbers

0	1	2	3	4	5	6	7
25	57	48	37	12	92	86	33

ie step I A[0] with A[1] ie A[0] < A[1] ie 25 < 57 so no interchange.

A[1] with A[2] ie A[1] > A[2] so interchange

Step II:-

0	1	2	3	4	5	6	7
25	48	57	37	12	92	86	33

A[2] > A[3] so

Step III

0	1	2	3	4	5	6	7
25	48	37	57	12	92	86	33

step (IV)

0	1	2	3	4	5	6	7
25	48	37	57	12	92	86	33

A[3] > A[4]

so interchange

step (V)

0	1	2	3	4	5	6	7
25	48	37	12	57	92	86	33

57 < 92 so not interchage

step (VI) 92 > 86 ie A [5] > A [6] so interchge

0	1	2	3	4	5	6	7
25	48	37	12	57	86	92	33

step (VII) A[6] > A[7] so interchage

25	48	37	12	57	86	33	92

25	37	48	12	57	86	33	92

25	37	12	48	57	86	33	92

25	37	12	48	57	33	86	92

25	12	37	48	57	33	86	92

25	12	37	48	33	57	86	92

12	25	37	48	33	57	86	92

12	25	37	33	48	57	86	92

12	25	33	37	48	57	86	92

Algorithm :-

1) Begin
2) Read the n elements
3) for i = 1 to n
   for j = n down i + 1
   if a[j] <= a[j-1]
   Swap (a[j], a[j-1])
4) End

Total No of comparision in Bubble sort

$= (N-1) + (N-2) + \ldots + 2 + 1$

$= (N-1) * N/2 = O(N^2)$

Efficiency :-

In the best case if all elements are sorted in the array no interchange is made. Under the worst case condition i.e when there are n-1 passes & n-1 comparison have to be made on each pass total no of comparision $= (n-1) * (n-2)$ which is $O(n^2)$.

Advantages :-

1) Simple to Understand
2) Easy to implement

Disadvantages :-

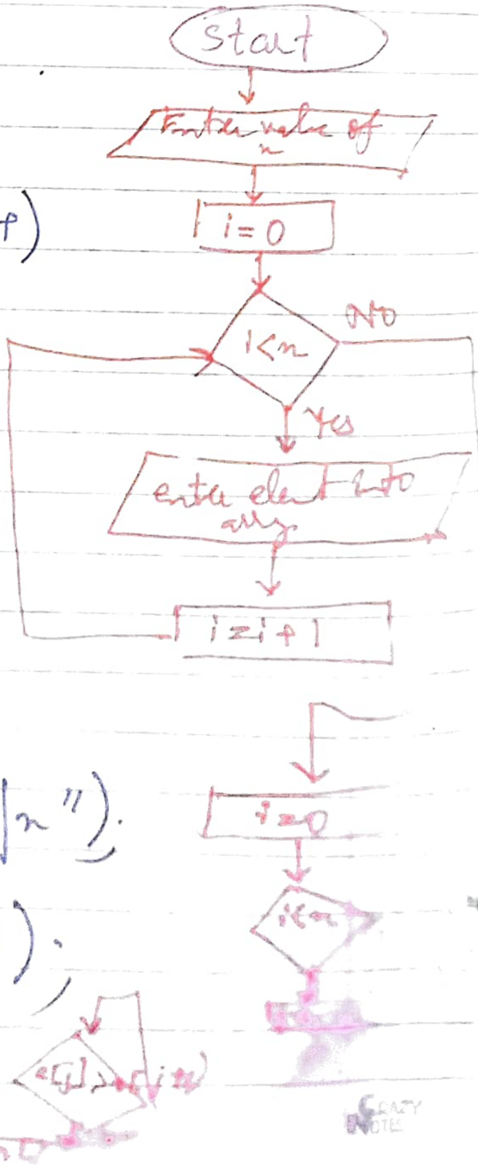
1) slowest sorting technique
2) Most inefficient sorting technique.

CRAZY NOTES

Q Write a program to sort the elements using Bubble sorting technique.

Sol) ⇒

```c
# include <stdio.h>
# include <conio.h>
void main()
{
 int a[20], n, i, j, temp;
 clrscr();
 printf("\n how many no in array\n");
 scanf("%d", &n);
 printf("\n enter the element into array\n");
 for (i=0; i<n; i++)
  {
   scanf("%d", & a[i]);
  }
 printf("\n Unsorted array\n");
 for (i=0; i<n; i++)
   printf(" %d\t", a[i]);
 for (i=0; i<n; i++)
  {
   for (j=0; j<n-1-i; j++)
    {
     if (a[j] > a[j+1])
      {
       temp = a[j];
       a[j] = a[j+1];
       a[j+1] = temp;
      }
    }
  }
 printf("\n Sorted Array\n");
 for (i=0; i<n; i++)
  printf("%d\t", a[i]);
 getch();
}
```



(2) Insertion sort :-

1) An insertion sort is one that sort a set of values by inserting values into an existing sorted file.

2) The main idea of the algorithm is to build a complete solution by inserting a new element from the unsorted portion of the list.

3) In an array a with n element a[1] a[2] ——— a[n] is in the memory. The insertion sort algorithm scans a from a[1] to a[n] inserting each element a[K] into its proper position in previously sorted subarray.

a[1], ... a[K-1]

Advantages :-

(i) Simple to Understand
(ii) Easy to implement

Disadvantages :-

(i) works inefficiency on large amount of data.

0	1	2	3	4	5	6
25	15	30	9	99	20	26

pass I :- a[1] < a[0] interchanging the position of element.

15	25	30	9	99	20	26
0	1	2	3	4	5	6

pass II :- a[3] is less than a[0], a[1], a[2]

9	15	25	30	99	20	26
0	1	2	3	4	5	6

pass III :- a[4] > a[3] So No change

pass (IV) :- a[5] is less than a[2], a[3], a[4]

9	15	20	25	30	99	26
0	1	2	3	4	5	6

pass (V) :- a[6] is less than a[4], a[5], a[6]

9	15	20	25	26	30	99
0	1	2	3	4	5	6

Algorithm :-

step 1 :- For I = 1 to N
step 2 :- J = I
step 3 :- while (j >= 1)
step 4 :- If (A[j] < A[j-1]) then
step 5 :- temp = A[j]
step 6 :- A[J] = A[J-1]
step 7 :- A[J-1] = temp
step 8 End If
J = J-1
[End of while Loop]
[End of step 1 For LOOP]
p 9 : Exit

Q Write a program to sort element of an array using insertion sort technique.

sol) ⇒
```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[100], n;
int i, j, temp;
clrscr();
printf("\n Enter Number of elements");
scanf("%d", &n);
printf("\n Enter element into array \n");
for (i=0; i<n; i++)
  scanf("%d", &a[i]);
printf("\n The Unsorted list \n");

for(i=0; i<n; i++)
  printf("%d \t", a[i]);
for (i=1; i<n; i++)
  {
    j=i;
    while (j>=1)
    {
      if (a[j]< a[j-1])
       {
        temp = a[j];
        a[j] = a[j-1];
        a[j-1] = temp;
       }
      j=j-1;
    }
  }
 printf("\n The sorted List is \n");
  for (i=0; i<n; i++)
   printf("%d\t", a[i]);
```

```
printf("\n");
getch();
}
```

Example:-

0	1	2	3	4	5	6	7
82	42	49	8	92	25	59	52

Step I:- a[1] < a[0] so interchanging a[0] & a[1]

0	1	2	3	4	5	6	7
42	82	49	8	92	25	59	52

Step II:- a[2] < a[1] but a[2] ≮ a[0] so only interchanging 1 & 2.

0	1	2	3	4	5	6	7
42	49	82	8	92	25	59	52

Step III:- Here a[3] is less than a[0], a[1], a[2]

0	1	2	3	4	5	6	7
8	42	49	82	92	25	59	52

Step IV:-

0	1	2	3	4	5	6	7
8	42	49	82	92	25	59	52

Here a[5] is less than a[1], a[2], a[3], a[4].

Step V:-

0	1	2	3	4	5	6	7
8	25	42	49	82	92	59	52

Step VI:-

0	1	2	3	4	5	6	7
8	25	42	49	59	82	92	52

Step VII

0	1	2	3	4	5	6	7
8	25	42	49	52	59	82	92

selection sort

If we have a list of elements in unsorted order & we want to make a list of elements in sorted order then first we will take the smallest element & keep in the new list, after that second smallest element & so on until the largest element of list.

Algo

pass I:

① search the smallest element from arr[0] ... arr[N-1]

② Interchange arr[0] with smallest element.

Result arr[0] is sorted.

pass II :-

① search the smallest element from arr[1] — arr[N-1]

② Interchange arr[1] with smallest element

Result arr[0], arr[1] is sorted

- - ——————— -

- ——————— —

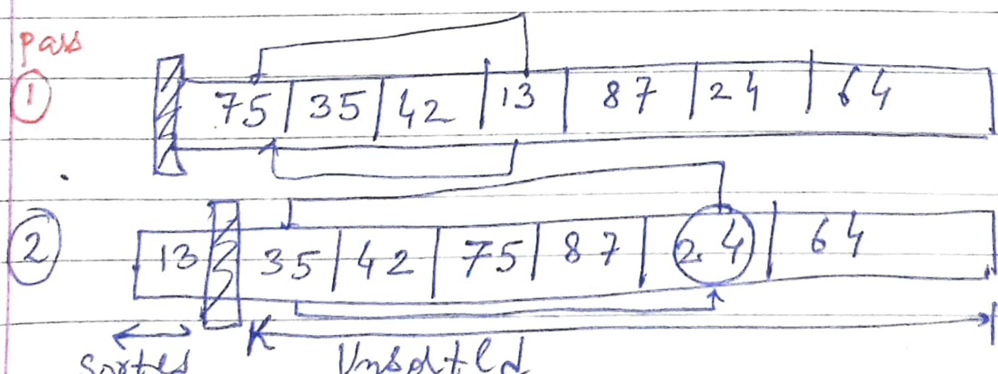
pass N-1 :-

① search the smallest element from arr[N-2] ... arr[N-1].

② Interchange arr[N-2] with smallest element.

Result :- arr[0] — arr[N-1] is sorted.

pass



from arr [1] —— arr [n-1]
select

(3)

13	24	42	75	87	35	64

Here smallest is 35.

(4)

13	24	35	75	87	42	64

sorted ⟵⟶ unsorted

Here smallest element in Unsorted part is 42

(5)

13	24	35	42	87	75	64

Here smallest in Unsorted is 64.

(6)

13	24	35	42	64	75	87

sorted unsorted

output :-

13 24 35 42 64 75 87

Analysis :- $F(n) = (n-1) + (n-2) + (n-3) + \ldots + 3 + 2 + 1$

$$\text{Sum} = \frac{n}{2}\left[2a + (n-1)d\right]$$

a → First element in Series

n = No of element in the series

d = diff between second elet & first elet [ie consecutive elet] ie 2-1

$a = (n-1)$ $d \rightarrow \cancel{n} - 2 - \cancel{n} + 1 \rightarrow -1$

Hence

$$F(n) = \frac{(n-1)}{2}\left[2(n-1) + \{(n-1) - 1\}\{(n-2) - (n-1)\}\right]$$

$$= (n-1)/2 \left[2n - 2 + (n-2)(-1)\right]$$

$$= (n-1)/2 \left[2n - 2 - n + 2\right]$$

$$= \frac{(n-1)}{2} \times n$$

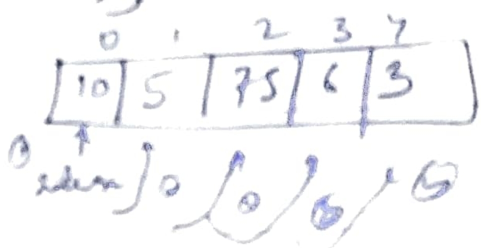
$$= \boxed{O(n^2)}$$

Q. Write a program to sort the elements of array using selection sort.

Sol:

```
#include <stdio.h>
#include <conio.h>
void main()
{
int arr[20], i, j, K, n, temp, smallest;
printf("\n Enter the number of elements\n");
scanf("%d", &n);
for(i=0; i<n; i++)
 {
 printf("\n Enter element %d:", i+1);
 scanf("%d", &arr[i]);
 }
printf("\n Unsorted list: \n");
 for(i=0; i<n; i++)
  printf("%d", arr[i]);
  printf("\n");

for(i=0; i<n-1; i++)
 {
  smallest = i;
  for(K=i+1; K<n; K++)
   {
    if(arr[smallest] > arr[K])
     smallest = K;
   }
   if(i != smallest)
   {
    temp = arr[i];
    arr[i] = arr[smallest];
    arr[smallest] = temp;
   }
```

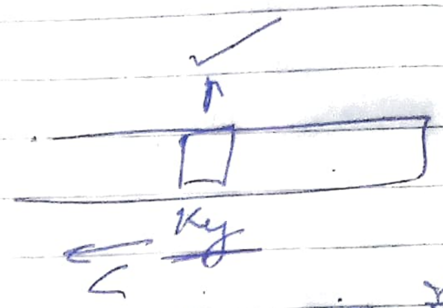


```
printf ("\n After pass %d elements are:"
                                ,i+1);
for(j=0; j<n; j++)
  printf("%d", arr[j]);
  printf("\n");
} /* End of for */
printf ("\n Sorted list is: \n");
 for(i=0; i<n; i++)
  printf("%d", arr[i]);
 printf("\n");
}
```

Q Write a C++ program to sort the elements of array using quick sort technique

```
# include <iostream.h>
# include <conio.h>
# define max 100
class quickSort
{
 int i, l, h;
public:
  void input();
  void output (int *, int);
  void quick_sort (int *, int , int);
};
void main()
{
  int i, l, h, n, a[max];
  clrscr();
  quicksort qs;
  cout <<endl << " How many elements in the array";
  cin>>n;
  cout << endl << "Enter the elements :" <<endl;
  for (i=0; i<=n-1; i++)
   {
    cin >> a[i];
   }
   l=0;
   h = n-1;
   qs.quick_sort (a, l, h);
   cout << "Sorted array"
   qs.output (a, n);
   getch();
}
```

```
void quicksort :: quick_sort (int a[], int l,
                              int h)
{
 int temp, key, low, high;
  low = l;
  high = h;
  key = a[(low+ high)/2];
 do
 {
  while (key > a[low])
   {
    low++;
   }
  while (key < a[high])
   {
    high--;
   }
 if (low <= high)
 {
  temp = a[low];
  a[low++] = a[high];
  a[high--] = temp;
 }
 }
 while (low <= high);
 if (l < high)
   quick_sort (a, l, high);
 if (low < h)
  quick_sort (a, low, h);
}
void quicksort :: output (int a[], int n)
{
   for (i=0; i<=n-1; i++)
   { cout << endl << a[i];
   }
}
```



i.e Key बड़ा है low se

Q write a c program to implement Merge sort.

```
# include< stdio.h>
# include < conio.h>
void merge (int a[], int, int, int);
void merge_sort (int a[], int, int);
void main ()
{
 int a[10], i, n;
 clrscr();
 printf("\n Enter the number of elements in the array:");
 scanf("%d", &n);
 printf("\n Enter the elements of the array");
 for (i=0; i<n; i++)
 {
  scanf("%d", &a[i]);
 }
 merge_sort (a, 0, n-1);
 printf("\n The sorted array is:\n");
 for (i=0; i<n; i++)
   printf("%d\t", a[i]);
 getch();
}
void merge(int a[], int beg, int mid, int end)
 {
 int i=beg, j=mid+1, index=beg; temp[10], k;
 while((i<=mid) && (j<=end))
  { if (a[i] < a[j])
    {
  temp[index] = a[i];
    i++;
  }
```

```
else
{
  temp[index] = a[j];
  j++;
}
index++;
}
if (i > mid)
{
  while (j <= end)
  {
    temp[index] = a[j];
    j++;
    index++;
  }
}
else
{
  while (i<=mid)
  {
    temp[index] = a[i];
    i++;
    index++;
  }
}
for (K= beg; K<index; K++)
      a[K] = temp[K];
}
void  merge-sort(int a[], int beg, int end)
{
  int mid;
  if (beg < end)
  {
    mid = (beg + end)/2;
    merge-sort (a, beg, mid);
    merge-sort (a, mid+1, end);
  } merge(a, beg, mid, end);
}
```

Heap sort

Algo HeapSort (ARR, N)

Step 1: [Build Heap H]

Repeat for I = 0 to N-1

CALL Insert_Heap (ARR, N, ARR [I])

[END OF LOOP]

Step 2: (Repeatedly delete the root elel)

Repeat while N > 0

CALL Delete_Heap (ARR, N, VAL)

SET N = N + 1

[END OF LOOP]

Step 3: END.

Various Operations of Heap

Consider Heap-size (A) a variable containing the array index of the last element of the Heap A. Elements in the array that are beyond this index are not part of heap. Thus the last element of Heap is referenced by A[HEAP-SIZE (A)]

Traversal of Heap

parent:-

This operation returns index of parent

Algo :- parent operation

step I :- start
step II :- Return i/2
step 3 :- End

left:- This operation returns index of left child. let us assume that index i is given to us.
Consider HEAP-SIZE(A) a variable containing the array index of the last element of heap:-

Algo :- left operation

step ① start
step ② compare Heap-size (A) with 2i, if 2i > HEAP-SIZE (A) then go to step ⑤ else go to step ④

step (3) Return i ie leaf node
step (4) Return 2i
step (5) End

Algo Right operation

step (I) start
step (II) Compare Heap-size (A) with 2i+1, if $2i+1 >$ Heap-size (A) then go to step (3)
else goto step (4)
step (3) Return i ie leaf node
step (4) Return 2i+1
step (5) End

Algo Heapify operation

step (I) start
step (II) Repeat the following steps for all levels of Heap
step (III) compare the value of i with the children of i (left & right). If the value of i is the smallest of three then goto step (5) else goto step (4)
step (4) Swap the position with the smaller child.
step (5) End

Worst case comparisons

It takes 2 comparisons to move down 1 level ie 1 iteration of loop, comparing i to left child & then the smaller to those to its right child. The max no of times the procedure will make these comparisons is the height of the heap ie $\log_2 n$. Therefore the cost of heapify is approximately $2\log_2 n$ where n is the no of elements of heap rooted at i.

This worst case no of comparisions is $O(\log_2 n)$.

Build_Heap

Algo :- Build-Heap operation

step 1 Start

step 2 starting from $i = n/4*2$ repeat step 3 & 4 until it reaches the first element of the array.

step 3 call the operation Heapify (i, A)

step 4 i = i - 1

step 5 End

Worse Case Comparisons (Heap_Build) is better than Insert operator to build a heap.

The elements in the last level of Heap are already single element heaps & therefore Build_Heap (i, A) starts at the last element of the second last level & works its way back one element at a time until it reaches the first element of the array. Heapify (i, A) uses two comparisons for each call.

Therefore at the second last level which contains elements, the cost will be $n/4 * 2$. For the third last level, which contains $n/8$ levels, the cost will be $n/8 * 4$, since each level is two comparisons. Summing the cost each level gives the cost of Build_Heap (A) as :- $2n = O(n)$

This is much better than using Insert(K,A) ntres, $O(n \log_2 n)$

Analysis of sorting Method

Sorting	Best Case	Average Case	Worst Case
Bubble	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion	$O(n)$	$O(n^2)$	$O(n^2)$
Shell	$O(n)$	$O(n^2)$	$O(n^2)$
Heap	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Merge	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Quick	$O(\log n)$	$O(\log n)$	$O(n^2)$

Internal sorting :- This sorting method is applied on the data stored on the main memory. since the main memory its is not feasible to apply the sorting technique always

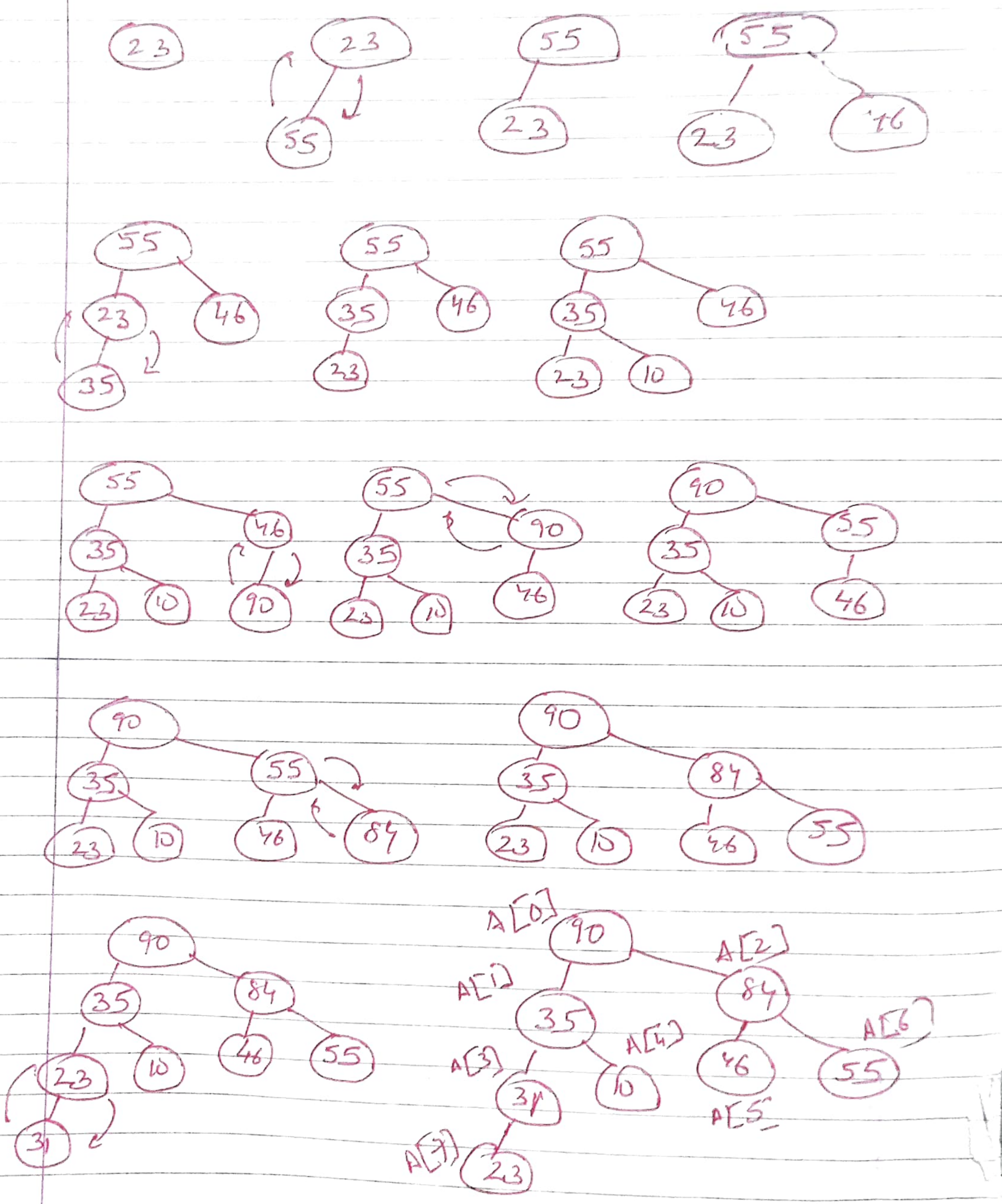
External sorting

External storing is a complex sorting technique when a large data has to be sorted & if we need to place part of date on the main memory & remaining on the secondary memory then external sorting can be done.

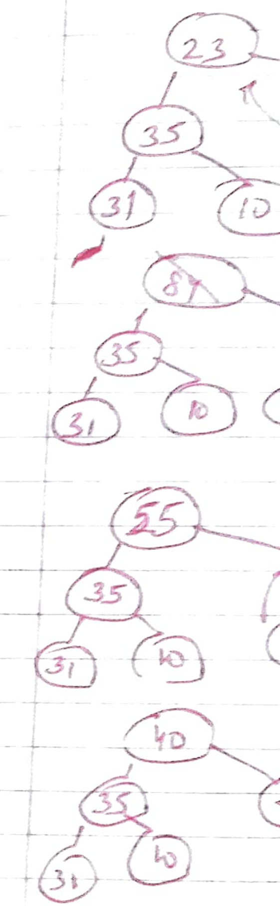
The data stored on secondary memory is part by part loaded into the main memory & sorting can be done over there. The sorted data can be then stored in the intermediate files. Thus many intermediate files can be generated. And finally all the sorted intermediate files are merged in a single file. Thus the large amount of date can be sorted using this technique

CRAZY NOTES

23 55 46 35 10 90 84 31



Deletes



Delete 90 & insert 23 at top

Q Write a C program to implement heap sort.

```
#include <stdio.h>
#include <conio.h>
#define MAX 10
void RestoreHeapUp(int *, int);
void RestoreHeapDown(int *, int, int);
void main()
{
  int Heap[MAX], n, i, j;
  clrscr();
  printf("\n Enter the number of elements:");
  scanf("%d", &n);
  printf("\n Enter the elements");
  for (i = 1; i <= n; i++)
  {
     scanf("%d", & Heap[i]);
     RestoreHeapUp(Heap, i);
  }
  //Delete the root element & heapify the heap
  j = n;
  for (i = 1; i <= j; i++)
  {
    int temp;
    temp = Heap[1];
    Heap[1] = Heap[n];
    Heap[n] = temp;
    n = n - 1; //The element Heap[n] to be deleted
    RestoreHeapDown(Heap, 1, n); //Heapify
  }
  n = j;
  printf("\n The sorted elements are:");
```

suppose it