UNIT – 6

Sorting

* Types of sorting –
1. Insertion sort
2. Selection sort
3. Merge sort
4. Heap sort

Insertion sort :-

1- Insertion sort is based on the idea that one element from the input elements is consume in each iteration to find a correct position to which it belongs in a sorted array.

2- It iterates the input element by growing the sorted array at each iteration. It compares the current element with the largest value in the sorted array.

3- If the current element is greater, then it leaves the element in its place and moves on the next element else to find its correct position in the sorted array, and moves it to that position.

4- This is done by shifting all the elements, which are larger than the current element, in the sorted array to one position ahead.

Date / /

Insertion sort is used when number of elements is small.
It can also be useful when input array is sorted only for elements are misplace.
In complete big array, use binary search to reduce the number of comparisions in normal insertion sort.

Example:

7	8	3	1	2

Sorted array | Unsorted array

Step 1- | 7 (0) | | 8 (1) | 3 (2) | 1 (3) | 2 (4)

Step 2- | 7 (0) | 8 (1) | | 3 (2) | 1 (3) | 2 (4)

Step 3- | 3 (0) | 7 (1) | 8 (2) | | 1 (3) | 2 (4)

Step 4- | 1 (0) | 3 (1) | 7 (2) | 8 (3) | | 2 (4)

Step 5- | 1 (0) | 2 (1) | 3 (2) | 7 (3) | 8 (4) |

Methods of insertion sort :-

1- A sub-list (or sorted array) is maintained which is always sorted.
2- Not suitable for large data set.
3- Average and worst case complexity is n^2 (n is no. of lines).
4- n-1 steps are required to sort elements.
5- In each pass, we insert current element

Date / /

at appropriate place so that the element in current range are in order.

6- Each pass has K comparision where k is the pass number.

```
#include <stdio.h>
#include <conio.h>
void insert (int [], int)
void main()
{
int a[20], i, n;
clrscr();
printf ("Enter the no. of items");
for (i=0; i<n; i++)
{
scanf ("%d", &a[i]);
}
insert (a, n);
getch();
}
void insert (int a[], int n)
{
int i, j, temp;
for (i=1; i<n; i++)
{
temp = a[i];
for (j=i-1; j>=0; j--)
{
if (a[j] > temp)
{
```

```
        a[j+1] = a[j];
    }
    else break;
    }
    a[j+1];
    }
    printf("Data after insertion sort");
    for(i=0; i<n; i++)
    printf("\n%d", a[i]);
    }
```

Selection sort :-

1- The selection sort technique is based upon the extension of the minimum and maximum technique.

2- By means of nest of for loops, a pass through the array is made to locate the minimum value.

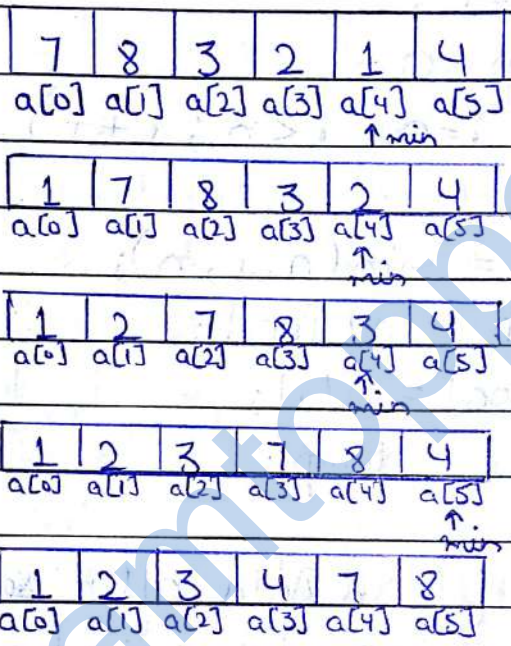
3- Once this element is found, it is placed in the first position of the array (position 0).

4- Another the remaining elements, is made to find the next smallest element, which is placed to the second position of array (position 1) and so on.

Date / /

5- Once the next two last elements is compared with the last element, all the elements have been sorted into the ascending order.

Example:




```
#include <stdio.h>
#include <conio.h>
void select (int[], int);
void main()
{
int a[20], i,n;
clrscr();
printf ("Enter the no. of items in array");
scanf ("%d", &n);
printf ("Enter the data in the array");
for(i=0; i<n; i++)
{
scanf ("%d", &a[i]);
select (a,n);
getch();
```

```
}
void select(int a[], int n)
{
int i, loc, temp;
loc = 0;
temp = 0;
for(i=0; i<n; i++)
 {
loc = min(a, i, n);
temp = a[loc];
a[loc] = a[i];
 a[i] = temp;
 }
printf("\n Data after selection sort");
 for(i=0; i<n; i++)
printf("\n%d", a[i]);
}
```

★ Merge sort :-

1- Merge sort is a sorting technique which divides the array into some arrays of size 2 and merge adjecent pairs.

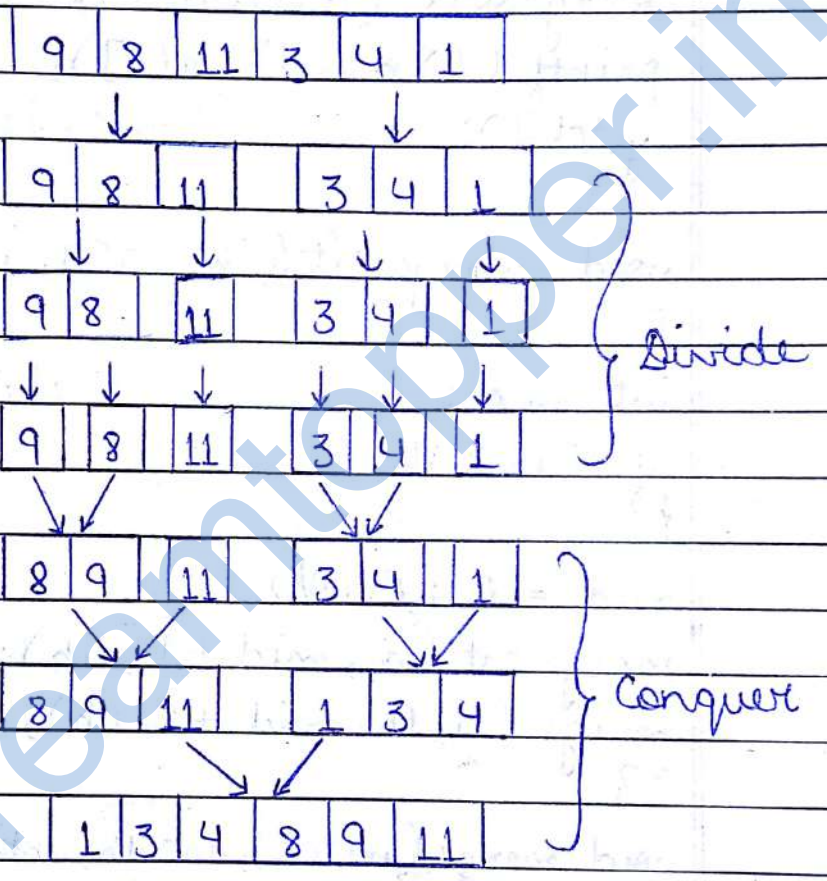
2- We then have approximately $\frac{n}{2}$ array of size 2.

Date / /

3- This process is repeated until there is only one array remaining of size n.

4- Suppose an array A with n elements A_1, A_2 to An is in memory.

Example:




```
#include <stdio.h>
#include <conio.h>
void mergesort(int a[], int, int);
void merge(int a[], int, int, int);
void main()
{
int a[20], i, n;
clrscr();
printf("Enter the number of elements");
scanf("%d", &n);
printf("Enter the elements");
```

Date / /

```
for(i=0; i<n; i++)
{
scanf("%d", &a[i]);
}
mergesort(a, 0, n-1);
printf("\n%d", a[i]);
getch();
}
void mergesort(int a[], int lb, int ub)
{
int mid;
if (lb<ub)
{
mid = (lb+ub)/2
mergesort(a, mid+1, ub);
merge(a, lb, mid+1, ub);
}}
void merge(int a[], int lb, int mid, int ub)
{
int k, p1, p2, p3, b[20];
p1 = lb;
p3 = ub;
p2 = mid;
while((p1<mid)&&(p2<=ub))
{
if (a[p1] <= [p2])
b[p3++] = a[p1++];
else
b[p3++] = a[p2++];
}
```

Date / /

```
while (p1 <= mid)
{
b[p3++] = a[p1++];
}
while (p2 <= ub)
{
b[p3++] = a[p2++];
}
for (k = lb; k < p3; k++)
{
a[k] = b[k];
}
}
```

★ Heap Sort :-

1- A heap is define as an almost complete binary tree of n nodes such that the value of each node is less than or equal to the value of the parent. In sequential representation of an almost complete binary tree. This implies that the root of the binary tree (i.e. the first array element) has the largest key in the heap.

2- This type of heap is usually called descending heap or max heap, as the path from the root node to the terminal node from an

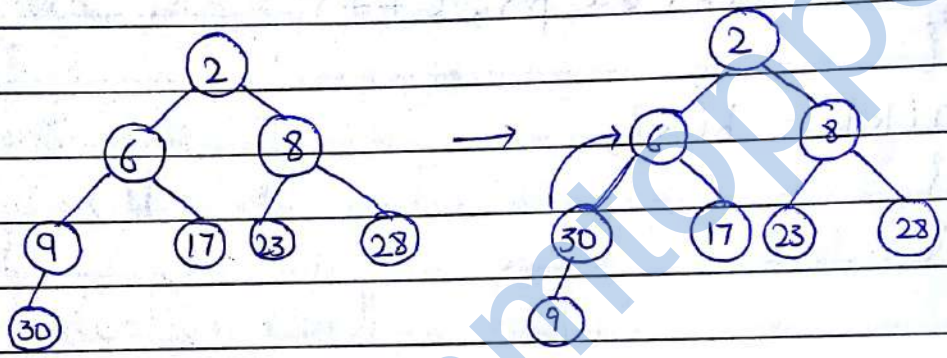
Date / /

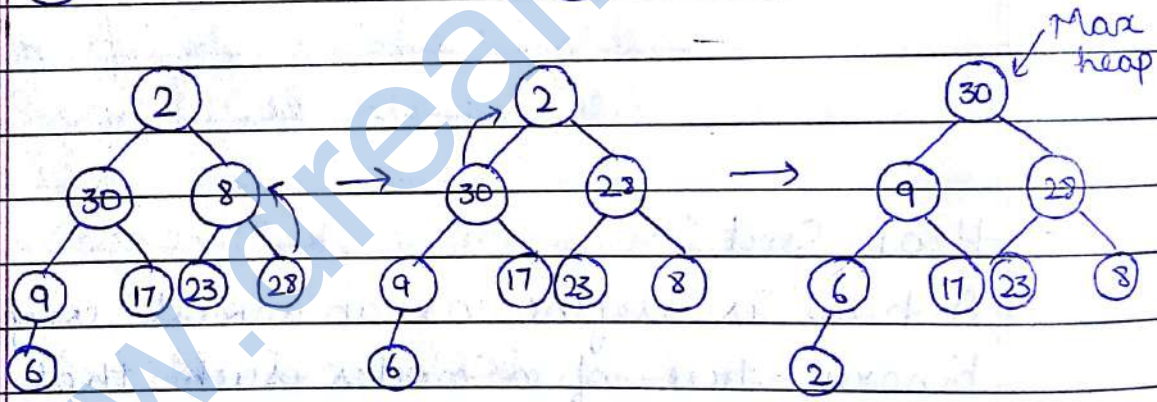
ordered list of elements arranged in descending order.

We can also define an ascending heap as an almost complete binary tree in which value of each node is greater than or equal to the value of its father.

Example-

2	6	8	9	17	23	28	30





Array after sorting - | 30 | 28 | 23 | 17 | 9 | 8 | 6 | 2 |

A heap is very useful in implementing priority queues. The priority queue is a data structure in which the intrinsic ordering of the data items determines the result of its basic operation. Primarily queues can be classified as -

1- Ascending priority queue

Date / /

2- Descending priority queue :-

1- Ascending priority queue – An ascending priority queue can be define as a group of elements in which new elements are inserted orbiterily but only the smallest element is deleted from it.

2- Descending priority queue – A descending priority queue can be define as a group of elements in which new elements are inserted orbiterily but only the largest element is deleted from it.

* Sorting :- Sorting is the basic operation in computer science. Sorting refers to the operation of arranging data in some given sequence i.e. increasing order or decreasing order.

Sorting is categorized as internal sorting and external sorting. By internal sorting, means we are arranging the numbers within the array which is in computer primary memory, whereas the external sorting is the sorting of numbers from the external file by reading in from secondary memory.

There are various sorting methods –

1- Insertion sort
2- Selection sort

Date / /

3- Merge sort

4- Heap sort

Date / /

* Insertion sort algorithm –

Algorithm: INSERTION_SORT (ARR, N)

1- Repeat step 2 to 7 for A = 2 to N:
2- Set TEMP = ARR[A] and B = A-1
3- Repeat step 4 to 6 while TEMP < ARR[B]:
4- Set ARR[B+1] = ARR[B]. [Moves element forward]
5- Set B = B-1
6- If B = 0; then goto step 4.
[End of Step 3 loop]
7- Set ARR[B+1] = TEMP.
[Inserts element in proper place]
[End of Step 1 loop]
8- Exit

* Illustration of Insertion Sort Algorithm –
Let 3, 6, 2, 7, 1, 8, 9, 4, 0 be any list

Pass 1: ARR[2] (i.e. 6) is greater than ARR[1] (i.e. 3) so ARR[1] and ARR[2] (i.e. 3 and 6) are sorted.

Pass 2: ARR[3] will be inserted before ARR[1] (i.e. 3) so that the list from ARR[1] to ARR[3] is sorted.

Date / /

* Selection-sort method —

Pass 1: Step 1: If ARR[1] > ARR[2] then Swap (ARR[1], ARR[2])
Step 2: If ARR[1] > ARR[3] then Swap (ARR[1], ARR[3])
⋮
Step N-1: If ARR[1] > ARR[N] then Swap (ARR[1], ARR[N])

Pass 2: Step 1: If ARR[2] > ARR[3] then Swap (ARR[2], ARR[3])
Step 2: If ARR[2] > ARR[4] then Swap (ARR[2], ARR[4])
⋮
Step N-2: If ARR[2] > ARR[N] then swap (ARR[2], ARR[N])
⋮

Pass N-1: Step 1: If ARR[N-1] > ARR[N] then swap (ARR[N-1], ARR[N])

* Algorithm for Selection Sort :—

Algorithm: SELECTION_SORT (ARR, N)

1- Repeat step 2 to 3 for A = 1 to N-1.
2- Repeat step 3 for B = A+1 to N:
3- If ARR[A] > ARR[B] then:
Swap (ARR[A], ARR[B]).
[End of if structure]
[End of step 2 loop]
[End of step 1 loop]
4. Exit