

UNIT - V

Sorting:

Sorting is a technic to rearrange element in a list in the form of ascending or descending order.

The sorting classified in 2 types.

- (i) Internal sort (or) selection sort
- (ii) External sort

(i) Internal sort:

* The method use only primary memory implement a sorting process.

* All data elements only used in main memory not in secondary memory.

* This ~~type~~ type of sorting is called internal sort.

An Internal sort derived in 3 types.

- (i) Insertion sort
- (ii) Selection sort
- (iii) Exchange sort (or) Bubble sort

Insertion sort - using shell sort algorithm.

Selection sort - using heap sort algorithm.

Bubble sort - using Quick sort algorithm.

(ii) External sort:

The sorting operation will be performed the inserting data elements are only used in secondary memory not in a primary memory.

Ex: Merge Sort

Insertion Sort:

* Insertion sort very simplest algorithm to be sort an array elements.

* This type of sorting method ordering a elements if input based on values. The main idea of insertion sort to prepare each time of inserting values.

* There is no prepare place in saved memory.

* A current values to be insert the data will be repeat interchange the value place this is called

Insertion Sort.

12 5 7 3 10 15

3 5 7 10 12 15

12 5

(1) insertion

5 12

(2) insertion

5 7

12

3

(3) insertion

3 5

7

10

12

15

(5) insertion

```
#include <iostream.h>
```

```
#include <conio.h>
```

```
void main()
```

```
{ int a[10], i, j, k, n;
```

```
clrscr();
```

```
cout << "How many elements you want to sort? In";
```

```
cin >> n;
```

```
cout<<"\n Enter the elements into an array: \n";
```

```
for (i=0; i<n; i++)
```

```
cin>>a[i];
```

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

```
{
```

```
    k = a[i];
```

```
    for (j=i; j>0 && k<a[j]; j--)
```

```
        a[j+1] = a[j];
```

```
        a[j+1] = k;
```

```
}
```

```
cout<<"\n\n Element after Sorting: \n";
```

```
for (i=0; i<n; i++)
```

```
cout<<" "a[i];
```

```
getch();
```

```
3.
```

The insertion sort does not exchange the element. The element places only move similar values insertion. The following steps are using insertion sort.

Step 1: Insertion all data elements in a array.

Step 2: To check the new insert value work some conditions to the previous values less than or greater than to perform the array index is 0 not in -1.

Step 3: To change the memory address of array index directly moved in 1 place to another place not an interchange operation.

Heap sort :-

is a sorting technic is a comparison based model. normally heap sort is a binary tree structure format. It is similar to selection sort. The binary tree sort maximum element is stored in complete binary tree format. 2 different approaches are using heap sort.

1. Binary tree format

2. Array format

Array representation in heap sort :

Since binary heap is perform complete binary tree format. Is easy to represent an array is more efficient.

Heap sort :

Algorithm :

Step 1: Built a maths heap from the input data.

Step 2: The largest value in a array is stored in root to replace at each time in left or right to move the data.

Step 3: Repeat above step is the size of Heap.

Shell sort:

Is mainly some variation of insertion sort. we moved element one position to another position to involved many times. The idea of shell sort is exchange the item based on large value.

Advantages of Shell sort:-

- * More efficient for medium size data.
- * More complex to better result of worst & average case.

How Shell sort works:

The following example of given data to perform shell sort.

This example consists of 8 element to perform virtual subset operation in 4 intervals.

{35, 14} {33, 19} {42, 27} {10, 44}

$$\text{Size} = \frac{8}{4} = \frac{\text{ele.}}{\text{subset}}$$

{12, 13} {14, 15} {16, 17} {18, 19}

$\overset{1}{\{12\}}, \overset{2}{14}, \overset{3}{16}, \overset{4}{18}\}$ $\overset{5}{\{13\}}, \overset{6}{15}, \overset{7}{17}, \overset{8}{19}\}$ 2 intervals, 2 subset

$\{12, 14, 16, 18, \overset{13}{15}, 17, 19\}$ 1 interval, 1 subset.

{12, 13, 14, 15, 16, 17, 18, 19}

Bucket Sort

* It works by Partitioning the ele. stored into buckets & return the result.

* Buckets are assigned based on each ele. in search key finally concatenate the buckets return in single array.

Difference between other sorting algorithm:

* The buckets each data hold 1 ele. use a count & duplicate ele. are eliminated.

This is better for radix sort.

Time complexity is reduced in more no. of buckets is distributed evenly.

Buckets requires extra space to increase space consumption & decrease the time complexity.

The following eg. is construct 6 inputs to store each data.

	3	5	7	3	4	6
	2	1	1	2	1	1
Using	2	2	1	1	1	1
count	3	3	4	5	6	7

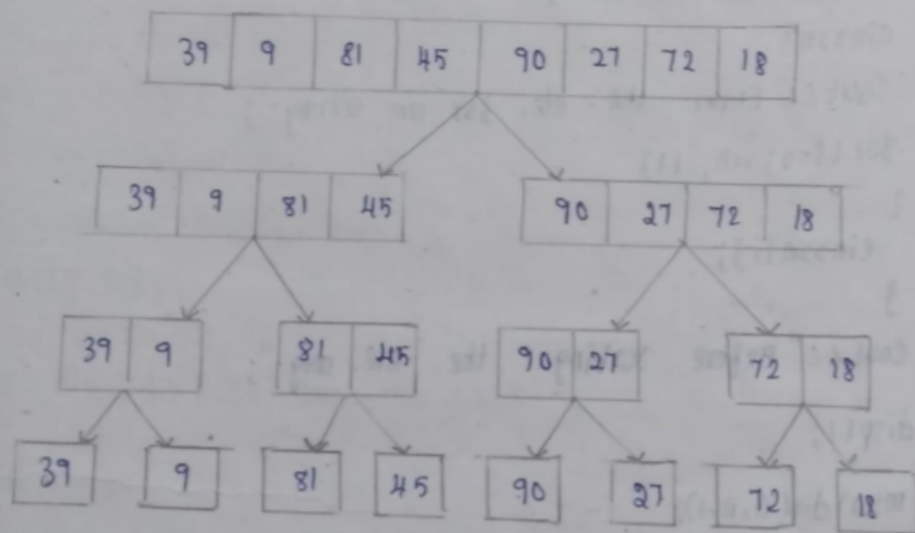
Merge sort : based on Divide & Conquer

* The basic concept of merge sort is it $\div$ es the list into 2 smaller sub-lists of $\approx$ = size. Recursively repeat this procedure till only 1 ele. is left in the sub-list.

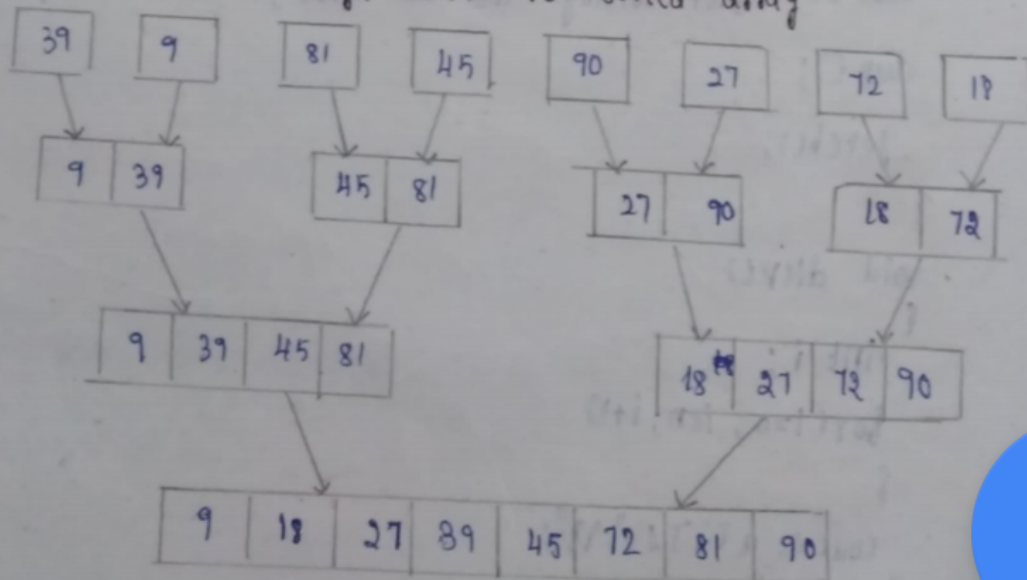
* After this, various sorted sub-lists are merged to form sorted parent list. This process goes on recursively till the original sorted list arrives.

Sort in ascending order:

Divide the array



Merge ele. to sorted array



* very effective for worst cases because this algo.
has lower time complexity for worst case also.

```
#include <iostream.h>
void disp();
void mergesort (int, int, int);
void msortdiv (int, int);
int a [50], n;
void main()
{
    int i;
    clrscr();
    cout << "Enter n value:";
    cin >> n;
    cout << "Enter the ele. for an array:";
    for (i=0; i<n; i++)
    {
        cin >> a[i];
    }
    cout << "Before sorting the ele. are:";
    disp();
    msortdiv(0, n-1);
    cout << "After sorting the ele. are:";
    disp();
    getch();
}

void disp()
{
    int i;
    for (i=0; i<n; i++)
    {
        cout << a[i] << " ";
    }
}
```

```
void mergesort (int low, int mid, int high)
```

```
{  
    int t[50], i, j, k;
```

```
    i = low;
```

```
    j = mid + 1;
```

```
    k = low;
```

```
    while (i <= mid && j <= high)
```

```
    {  
        if (a[i] >= a[j])
```

```
            t[k++] = a[j++];
```

```
    }  
    else
```

```
        t[k++] = a[i++];
```

```
    }
```

```
    while (i <= mid)
```

```
        t[k++] = a[i++];
```

```
    while (j <= high)
```

```
        t[k++] = a[j++];
```

```
    for (i = low; i <= high; i++)
```

```
        a[i] = t[i];
```

```
}
```

```
void msortdiv (int low, int high)
```

```
{  
    int mid;
```

```
    if (low != high)
```

```
    {
```

```
        mid = (low + high) / 2;
```

```
        msortdiv (low, mid);
```

```
        msortdiv (mid + 1, high);
```

```
        mergesort (low, mid, high);
```

```
    }
```

```
}
```