Searching and Sorting

There are basically two aspects of computer programming. One is data organization also commonly called as data structures. Till now we have seen about data structures and the techniques and algorithms used to access them. The other part of computer programming involves choosing the appropriate algorithm to solve the problem. Data structures and algorithms are linkedeachother.Afterdeveloping programmingtechniques torepresent information, it is logical to proceed to manipulate it. This chapter introduces this important aspect of problem solving.

Searchingisusedtofindthelocation whereanelementisavailable.Therearetwo types of search techniques. They are:

1. Linearorsequential search
2. Binarysearch

Sorting allows an efficient arrangement of elements within a given data structure. It isa way in which the elements are organized systematically for some purpose. For example, a dictionary in which words is arranged in alphabetical order and telephone director in which the subscriber names are listed in alphabetical order. There are many sorting techniques out of which we study the following.

1. Bubblesort
2. Quicksort
3. Selectionsortand
4. Heap sort

Therearetwotypesofsortingtechniques:

1. Internalsorting
2. Externalsorting

If all the elements to be sorted are present in the main memory then such sorting is called **internal sorting** on the other hand, if some of the elements to be sorted are kept on the secondary storage, it is called **external sorting**. Here we study only internal sorting techniques.

Linear Search:

This is the simplest of all searching techniques. In this technique, an ordered or unordered list will besearched one by one fromthe beginning until the desiredelement is found. If the desired element is found in the list then the search is successful otherwise unsuccessful.

Suppose there are 'n' elements organized sequentially on a List. The number of comparisons required to retrieve an element from the list, purely depends on where the element is stored in the list. If it is the first element, one comparison will do; if it is second element two comparisons are necessary and so on. On an average you need [(n+1)/2] comparison's to search an element. If search is not successful, you would need 'n' comparisons.

The time complexity of linear search is ***O(n)***.

Algorithm:

Let array a[n] stores n elements. Determine whether element 'x' is present or not.

```
linsrch(a[n],x)
{
        index=0;
        flag =0;
        while(index<n) do
        {
                if(x==a[index])
                {
                        flag=1;
                        break;
                }
                index++;
        }
        if(flag==1)
                printf("Data found at %d position", index);
        else
                printf("data not found");

}
```

Example1:

Suppose we have the following unsorted list: 45,39,8,54,77, 38, 24,16,4,7, 9,20 If we are

searching for:

45, we'll look at 1 element before success
39, we'll look at 2 elements before success
8, we'll look at 3 elements before success
54, we'll look at 4 elements before success
77, we'll look at 5 elements before success
38 we'll look at 6 elements before success
24, we'll look at 7 elements before success
16, we'll look at 8 elements before success
4, we'll look at 9 elements before success
7, we'll look at 10 elements before success
9, we'll look at 11 elements before success
20, we'll look at 12 elements before success

For any element not in the list, we'll look at 12 elements before failure.

Example2:

Letusillustratelinearsearchonthefollowing9elements:

Index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
Elements	-15	-6	0	7	9	23	54	82	101

Searchingdifferentelementsisasfollows:

1. Searchingforx=7Search successful, datafoundat 3rdposition.

2. Searchingforx=82Searchsuccessful,data foundat 7thposition.

3. Searchingforx=42Searchun-successful,datanotfound.

Anon-recursiveprogramforLinearSearch:

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

main()
{
        intnumber[25],n,data,i,flag=0; clrscr();
        printf("\nEnterthenumberofelements:"); scanf("%d",
        &n);
        printf("\nEntertheelements:"); for(i
        = 0; i < n; i++)
                scanf("%d",&number[i]);
        printf("\nEntertheelementtobeSearched:");
        scanf("%d", &data);
        for(i=0;i<n;i++)
        {
                if(number[i]==data)
                {
                        flag=1;
                        break;
                }
        }
        if(flag==1)
                printf("\nDatafoundatlocation:%d",i+1);
         else
                printf("\nDatanotfound");
}
```

ARecursiveprogramforlinearsearch:

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

voidlinear_search(inta[],intdata,intposition,int n)
{
        if(position<n)
```

```
        {
                if(a[position]==data)
                        printf("\nDataFoundat%d",position);
                else
                        linear_search(a,data,position+1,n);
        }
        else
                printf("\nDatanotfound");
}

voidmain()
{
        inta[25],i,n,data;
        clrscr();
        printf("\nEnterthenumberofelements:"); scanf("%d",
        &n);
        printf("\nEntertheelements:"); for(i
        = 0; i < n; i++)
        {
                scanf("%d",&a[i]);
        }
        printf("\nEntertheelementtobeseached:");
        scanf("%d", &data);
        linear_search(a,data,0,n); getch();
}
```

BINARYSEARCH

If we have 'n' records which have been ordered by keys so that $x_1 < x_2 < \ldots < x_n$ .Whenwe are given a element 'x', binary search is used to find the corresponding element from the list. In case 'x' is present, we have to determine a value 'j' such that a[j] = x (successful search). If 'x' is not in the list then j is to set to zero (un successfulsearch).

InBinarysearchwejumpintothemiddleofthefile,wherewefindkeya[mid],and compare 'x' with a[mid]. If x = a[mid] then the desired record has been found. If x < a[mid] then 'x' must be in that portion of the file that precedes a[mid]. Similarly, if a[mid] > x, then further search is only necessary in that part of the file which follows a[mid].

Ifweuserecursiveprocedureoffindingthemiddlekeya[mid]oftheun-searched portion of a file, theneveryun-successfulcomparisonof'x' with a[mid] will eliminate roughlyhalftheun-searchedportionfromconsideration.

Since the array size is roughly halved after each comparison between 'x' and a[mid], and since an array of length 'n' can be halved only about $\log_2 n$ timesbeforereachinga trivial length, the worst case complexity of Binary search is about $\log_2 n$.

Algorithm:

Let array a[n] of elements in increasing order, $n \geq 0$, determine and ifwhether 'x' is present, so, set j such that x = a[j] else return 0.

```
binsrch(a[],n,x)
{
        low = 1; high = n;
        while(low≤high)do
        {
                mid=(low+high)/2 if
                (x < a[mid])
                        high =mid –1;
                else if (x > a[mid])
                        low=mid+1;
                else return mid;
        }
        return0;
 }
```

*low*and*high*are integer variables such that each time through the loop either 'x' is found or *low* is increased by at least one or *high* is decreased by at least one. Thus we havetwo sequences ofintegers approaching each other andeventually *low* will become greater than *high* causing termination in a finite number of steps if 'x' is not present.

Example1:

Letusillustratebinarysearchonthefollowing12elements:

Index	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*
Elements	4	7	8	9	16	20	24	38	39	45	54	77

If wearesearchingforx=4:(Thisneeds3comparisons) low = 1,
high = 12, mid = 13/2 = 6, check 20
low=1,high=5,mid=6/2=3,check8
low=1,high=2,mid=3/2=1,check4,***found***

If wearesearchingforx=7:(Thisneeds4comparisons) low = 1,
high = 12, mid = 13/2 = 6, check 20
low=1,high=5,mid=6/2=3,check8
low=1,high=2,mid=3/2=1,check4
low=2,high=2,mid=4/2=2,check7,***found***

If wearesearchingforx=8:(Thisneeds2comparisons) low = 1,
high = 12, mid = 13/2 = 6, check 20
low=1,high=5,mid=6/2=3,check8,***found***

If wearesearchingforx=9:(Thisneeds3comparisons) low = 1,
high = 12, mid = 13/2 = 6, check 20
low=1,high=5,mid=6/2=3,check8
low=4,high=5,mid=9/2=4,check9,***found***

If wearesearchingforx =16:(Thisneeds4comparisons) low =
1, high = 12, mid = 13/2 = 6, check 20
low= 1,high=5,mid=6/2=3,check8
low=4,high=5,mid=9/2=4,check9
low=5,high=5,mid=10/2=5,check16,***found***

If wearesearchingforx=20:(Thisneeds1comparison) low = 1,
high = 12, mid = 13/2 = 6, check 20, ***found***

If wearesearchingforx =24:(Thisneeds3comparisons) low =
1, high = 12, mid = 13/2 = 6, check 20
low=7,high=12,mid=19/2=9,check39
low=7,high=8,mid=15/2=7,check24,***found***

If wearesearchingforx =38:(Thisneeds4comparisons) low =
1, high = 12, mid = 13/2 = 6, check 20
low=7,high=12,mid=19/2=9,check39
low=7,high=8,mid=15/2=7,check24
low=8,high=8,mid=16/2=8,check38,***found***

If wearesearchingforx =39:(Thisneeds2comparisons) low =
1, high = 12, mid = 13/2 = 6, check 20
low=7,high=12,mid=19/2=9,check39,***found***

If wearesearchingfor x =45:(Thisneeds4comparisons) low =
1, high = 12, mid = 13/2 = 6, check 20
low=7,high=12,mid=19/2=9,check39
low=10,high=12,mid=22/2=11,check 54
low=10,high =10,mid =20/2=10,check45,***found***

If wearesearchingforx =54:(Thisneeds3comparisons) low =
1, high = 12, mid = 13/2 = 6, check 20
low=7,high=12,mid=19/2=9,check39
low=10,high =12,mid =22/2=11,check54,***found***

If wearesearchingforx =77:(Thisneeds4comparisons) low =
1, high = 12, mid = 13/2 = 6, check 20
low=7,high=12,mid=19/2=9,check39
low=10,high=12,mid=22/2=11,check 54
low=12,high =12,mid =24/2=12,check77,***found***

Thenumberofcomparisonsnecessarybysearch element:

20–requires1comparison;
8 and39–requires2 comparisons;
4,9,24,54–requires3comparisonsand
7,16,38,45,77–requires4comparisons

Summingthecomparisons,neededtofindalltwelveitemsanddividingby12,yielding 37/12 or approximately 3.08 comparisons per successful search on the average.

Example2:

Letusillustratebinarysearchonthefollowing9elements:

Index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
Elements	-15	-6	0	7	9	23	54	82	101

Solution:

Thenumberofcomparisonsrequiredforsearchingdifferentelementsisasfollows:

1. Ifwearesearchingforx=101:(Numberofcomparisons=4)

low	high	mid
1	9	5
6	9	7
8	9	8
9	9	9
		found

2. Searchingforx=82:(Numberofcomparisons=3)

low	high	mid
1	9	5
6	9	7
8	9	8
		found

3. Searchingforx=42:(Numberofcomparisons=4)

low	high	mid
1	9	5
6	9	7
6	6	6
7	6	notfound

4. Searchingforx=-14:(Numberofcomparisons=3)

low	high	mid
1	9	5
1	4	2
1	1	1
2	1	not found

Continuing in this manner the number of element comparisons needed to find each of nine elements is:

Index	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
Elements	-15	-6	0	7	9	23	54	82	101
Comparisons	*3*	*2*	*3*	*4*	*1*	*3*	*2*	*3*	*4*

No element requires more than 4 comparisons to be found. Summing the comparisons needed to find all nine items and dividing by 9, yielding 25/9 or approximately 2.77 comparisons per successful search on the average.

Therearetenpossiblewaysthatanun-successfulsearchmayterminatedepending upon the value of x.

Ifx<a(1),a(1)<x<a(2),a(2)<x<a(3),a(5)<x<a(6),a(6)<x<a(7)ora(7)<x<a(8)thealgorithmrequires3elementcomparisonstodetermine that 'x' is not present.Forall oftheremainingpossibilitiesBINSRCHrequires4 element comparisons. Thus

the average number of element comparisons for an unsuccessful search is:

(3+3+3 +4+4+3 +3 +3 +4+4) /10 =34/10= 3.4

TimeComplexity:

Thetimecomplexityofbinarysearchin asuccessfulsearchisO(logn) andforan unsuccessful search is O(log n).

Anon-recursiveprogramforbinarysearch:

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

main()
{
        intnumber[25],n,data,i,flag=0,low,high,mid;
        clrscr();
        printf("\nEnterthenumberofelements:"); scanf("%d",
        &n);
        printf("\nEntertheelementsinascendingorder:"); for(i
        = 0; i < n; i++)
                scanf("%d",&number[i]);
        printf("\nEntertheelementtobesearched:");
        scanf("%d", &data);
        low=0;high=n-1;
        while(low <= high)
        {
                mid = (low + high)/2;
                if(number[mid]==data)
                {
                        flag=1; break;

                }
                else
                {       if(data<number[mid])
                                high=mid- 1;
                        else
                                low=mid+1;
                }
        }
        if(flag==1)
                printf("\nDatafoundatlocation:%d",mid+1);
        else
                printf("\nDataNotFound");
}
```

Arecursiveprogramforbinarysearch:

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

voidbin_search(inta[],intdata,intlow,int high)
{
        int mid;
        if(low <=high)
        {
                mid=(low+high)/2;
                if(a[mid] == data)
                        printf("\nElementfoundatlocation:%d",mid+1);
                else
                {
                        if(data< a[mid])
                                bin_search(a,data,low,mid-1);
                        else
```

```
                        bin_search(a,data,mid+1,high);
                }
        }
        else
                printf("\nElementnotfound");
}
voidmain()
{
        inta[25],i,n,data;
        clrscr();
        printf("\nEnterthenumberofelements:"); scanf("%d",
        &n);
        printf("\nEntertheelementsinascendingorder:"); for(i
        = 0; i < n; i++)
                scanf("%d",&a[i]);
        printf("\nEntertheelementtobesearched:"); scanf("%d",
        &data);
        bin_search(a,data,0,n-1); getch();
}
```

BubbleSort:

The bubble sort is easy to understand and program. The basic idea of bubble sort is to pass through the file sequentially several times. In each pass, we compare each element in the file with its successori.e., X[i] with X[i+1] andinterchange two element when they are not in proper order. We will illustrate this sorting technique by taking a specific example. Bubble sort is also called as exchange sort.

Example:

Considerthearrayx[n]whichisstoredinmemoryasshownbelow:

X[0]	X[1]	X[2]	X[3]	X[4]	X[5]
33	44	22	11	66	55

Supposewewantourarraytobestoredin ascendingorder.Thenwepassthroughthe array 5 times as described below:

Pass1:(firstelementiscomparedwithallotherelements).

We compareX[i] andX[i+1] fori = 0,1,2,3, and4, andinterchangeX[i]andX[i+1] if X[i] > X[i+1]. The process is shown below:

X[0]	X[1]	X[2]	X[3]	X[4]	X[5]	Remarks
33	44	22	11	66	55	
	22	44				
		11	44			
			44	66		
				55	66	
33	22	11	44	55	66	

Thebiggestnumber66ismovedto(bubbledup)therightmost positioninthearray.

Pass2:(secondelementiscompared).

We repeat the same process, but this time we don't include X[5] into our comparisons. i.e., we compare X[i] withX[i+1] for i=0, 1, 2, and 3and interchange X[i] and X[i+1]if X[i] > X[i+1]. The process is shown below:

X[0]	X[1]	X[2]	X[3]	X[4]	Remarks
33	22	11	44	55	
22	33				
	11	33			
		33	44		
			44	55	
22	11	33	44	55	

Thesecondbiggestnumber55ismovednowtoX[4].

Pass3:(thirdelementiscompared).

We repeatthe same process, but this time weleave both X[4] andX[5]. By doingthis, we move the third biggest number 44 to X[3].

X[0]	X[1]	X[2]	X[3]	Remarks
22	11	33	44	
11	22			
	22	33		
		33	44	
11	22	33	44	

Pass4:(fourthelementiscompared).

We repeat the process leaving X[3], X[4], andX[5]. By doingthis, we movethe fourth biggest number 33 to X[2].

X[0]	X[1]	X[2]	Remarks
11	22	33	
11	22		
	22	33	

Pass5:(fifthelementiscompared).

We repeat the process leaving X[2], X[3], X[4], and X[5]. By doing this, we move the fifth biggest number 22 to X[1]. At this time, we will have the smallest number 11 in X[0]. Thus, we see that we can sort the array of size 6 in 5 passes.

Foranarrayofsizen,werequired(n-1)passes.

ProgramforBubbleSort:

```
#include <stdio.h>
#include<conio.h>
voidbubblesort(intx[],int n)
{
        int i, j,temp;
        for(i=0; i<n; i++)
        {
                for(j=0; j<n-i-1; j++)
                {
                        if(x[j]>x[j+1])
                        {
                                temp = x[j];
                                x[j] = x[j+1];
                                x[j+1]=temp;
                        }
                }
        }
}

main()
{
        inti,n,x[25]; clrscr();
        printf("\nEnterthenumberofelements:"); scanf("%d",
        &n);
        printf("\nEnterData:");
        for(i = 0; i < n ; i++)
                scanf("%d",&x[i]);
        bubblesort(x, n);
        printf("\nArrayElementsaftersorting:"); for (i
        = 0; i < n; i++)
                printf("%5d",x[i]);
}
```

TimeComplexity:

The bubble sort method of sorting an array of size n requires (n-1) passes and (n-1) comparisonsoneachpass.Thusthetotalnumberofcomparisonsis(n-1)*(n-1)=n^2-2n +1, which is O(n^2). Therefore bubble sortis veryinefficient when there are more elements to sorting.

SelectionSort:

Selection sort will not require no more than n-1 interchanges. Suppose x is an array of sizen stored in memory. The selection sort algorithm firstselects the smallest element in the array x and place it at array position 0;then it selects thenext smallest element in the arrayx and placeit at array position1.It simply continues this procedureuntilit places the biggest element in the last position of the array.

The array is passed through (n-1) times and the smallest element is placed in its respective position in the array as detailed below:

Pass1: Findthelocationjofthesmallestelementin thearrayx [0], x[1],…… x[n-1], andtheninterchangex[j]withx[0].Thenx[0]issorted.

Pass 2: Leave the first element and find the location j of the smallest element in the sub-array x[1], x[2], ... x[n-1], and then interchange x[1] with x[j]. Then x[0], x[1] are sorted.

Pass 3: Leave the first two elements and find the location j of the smallest element in the sub-array x[2], x[3], ... x[n-1], and then interchange x[2] with x[j]. Then x[0], x[1], x[2] are sorted.

Pass (n-1): Find the location j of the smaller of the elements x[n-2] and x[n-1], and then interchange x[j] and x[n-2]. Then x[0], x[1], ... x[n-2] are sorted. Of course, during this pass x[n-1] will be the biggest element and so the entire array is sorted.

TimeComplexity:

In general we prefer selection sort in case where the insertion sort or the bubble sort requires exclusive swapping. In spite of superiority of the selection sort over bubblesort and the insertion sort (there is significant decrease in run time), its efficiency is also $O(n^2)$ for n data items.

Example:

Letusconsiderthefollowingexamplewith9elementstoanalyzeselectionSort:

1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*Remarks*
65	70	75	80	50	60	55	85	45	findthefirstsmallest element
i								j	swap a[i]& a[j]
45	70	75	80	50	60	55	85	65	find the second smallestelement
	i			j					swapa[i]anda[j]
45	**50**	75	80	70	60	55	85	65	Find the third smallestelement
		i				j			swapa[i]anda[j]
45	**50**	**55**	80	70	60	75	85	65	Find the fourth smallest element
			i		j				swapa[i]anda[j]
45	**50**	**55**	**60**	70	80	75	85	65	Find the fifth smallest element
				i				j	swapa[i]anda[j]
45	**50**	**55**	**60**	**65**	80	75	85	70	Findthesixthsmallestelement
					i			j	swapa[i]anda[j]
45	**50**	**55**	**60**	**65**	**70**	75	85	80	Findtheseventhsmallestelement
						ij			swapa[i]anda[j]
45	**50**	**55**	**60**	**65**	**70**	**75**	85	80	Find the eighth smallestelement
							i	J	swapa[i]anda[j]
45	**50**	**55**	**60**	**65**	**70**	**75**	**80**	**85**	Theouterloopends.

Non-recursiveProgramforselectionsort:

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

voidselectionSort(intlow,inthigh); int

a[25];

int main()
{
        intnum,i=0;
        clrscr();
        printf("Enterthenumberofelements:");
        scanf("%d", &num);
        printf("\nEntertheelements:\n"); for(i=0; i
        < num; i++)
                scanf("%d",&a[i]);
        selectionSort(0,num-1 );
        printf("\nTheelementsaftersortingare:"); for(
        i=0; i< num; i++ )
                printf("%d     ", a[i]);
        return0;
}

voidselectionSort(intlow,inthigh)
{
        inti=0,j=0,temp=0,minindex; for(
        i=low; i <= high; i++ )
        {
                minindex=i;
                for(j=i+1;j<=high;j++)
                {
                        if(a[j]<a[minindex] )
                                minindex= j;
                }
                temp=a[i];
                a[i] = a[minindex];
                a[minindex]=temp;
        }
}
```

RecursiveProgramforselectionsort:

```
#include<stdio.h>#i
nclude<conio.h>

int x[6]={77,33,44,11,66};
selectionSort(int);

main()
{
        inti,n=0;
        clrscr();
        printf("ArrayElementsbeforesorting:"); for
        (i=0; i<5; i++)
```

```
            printf("%d",x[i]);
        selectionSort(n);                   /*callselectionsort*/
        printf ("\n Array Elements after sorting: ");
        for(i=0;i<5;i++)
            printf("%d",x[i]);
}

selectionSort(intn)
{
        intk,p,temp,min; if
        (n== 4)
            return(-1);
        min=x[n]; p
        = n;
        for(k=n+1;k<5;k++)
        {
            if(x[k]<min)
            {
                min=x[k]; p
                = k;
            }
        }
        temp = x[n];        /*interchangex[n]andx[p]*/
        x[n] = x[p];
        x[p]=temp;
        n++ ;
        selectionSort(n);
}
```

Quick Sort:

The quick sort was invented by Prof. C. A. R. Hoare in the early 1960's. It was one of thefirstmostefficientsortingalgorithms.Itisanexampleofa classof algorithms that work by "divide and conquer" technique.

The quick sort algorithm partitions the original array by rearranging it into two groups. The first group contains those elements less than some arbitrary chosen value taken from the set, and the second group contains those elements greater than or equal to the chosen value. The chosen value is known as the *pivot* element. Once the arrayhas been rearranged in this way with respect to the *pivot*, the same partitioning procedure is recursively applied to each of the two subsets. When all the subsets have been partitioned and rearranged, the original array is sorted.

The function partition() makes use of two pointers up and down which are moved toward each other in the following fashion:

1. Repeatedlyincrease thepointer 'up'untila[up]>= pivot.
2. Repeatedly decrease the pointer 'down' until a[down] <= pivot.
3. Ifdown>up,interchangea[down]witha[up] 4.

   Repeat the steps 1, 2 and 3 till the 'up' pointer crosses the 'down' pointer. If 'up' pointer crosses 'down' pointer, the position for pivotisfoundandplace pivot element in 'down' pointer position.

Theprogramusesarecursivefunctionquicksort().Thealgorithmofquicksortfunction sorts all elements in an array 'a' between positions 'low' and 'high'.

1. Itterminateswhentheconditionlow>=highissatisfied.Thiscondition will be satisfied only when the array is completely sorted.

2. Herewechoose the first element as the 'pivot'. So, pivot = x[low]. Now it callsthepartitionfunctiontofindtheproperpositionj oftheelement x[low] i.e.pivot.Thenwewillhavetwosub-arraysx[low],x[low+1],…… x[j-1] andx[j+1],x[j+2],……x[high].

3. Itcallsitselfrecursivelytosorttheleftsub-arrayx[low],x[low+1],..... x[j-1] between positionslow andj-1 (wherejisreturned bythepartition function).

4. Itcallsitselfrecursivelytosorttherightsub-arrayx[j+1],x[j+2],..x[high] between positions j+1 and high.

Thetimecomplexityofquicksortalgorithmisof***O(nlogn)***.

Algorithm

Sorts the elements a[p], ... ,a[q] which reside in the global array a[n] into ascending order.The a[n +1]is consideredtobe definedandmust be greater than all elements in a[n]; a[n + 1] = +

```
quicksort (p,q)
{
      if(p < q)then
      {
           callj=PARTITION(a,p,q+1);          //jisthepositionofthepartitioningelement
           call quicksort(p, j –1);
           callquicksort(j+1,q);
       }
}

partition(a,m,p)
{
       v = a[m]; up = m; down = p;          // a[m] is the partition element
       do
       {
              repeat
                     up =up +1;
              until(a[up]≥v);

              repeat
                     down=down-1;
              until(a[down]≤v);
              if (up<down)thencallinterchange(a,up,
        down); } while (up ≥ down);

       a[m]=a[down];
       a[down] = v;
       return (down);
}
```

```
interchange(a,up,down)
{
        p = a[up];
        a[up]=a[down];
        a[down] = p;
}
```

Example:

Select first element as the pivot element. Move 'up' pointer from left to right in search of an element larger than pivot. Move the 'down' pointer from right to left in search of anelementsmallerthanpivot.Ifsuchelementsarefound,theelementsareswapped.

This process continues till the 'up' pointer crosses the 'down' pointer. If 'up' pointer crosses 'down' pointer, the position for pivot is found and interchange pivot and element at 'down' position.

Letusconsiderthefollowingexamplewith13elementstoanalyzequicksort:

1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*Remarks*
38	08	16	06	79	57	24	56	02	58	04	70	45	
pivot				up						down			swap up& down
pivot				04						79			
pivot					up			down					swap up& down
pivot					02			57					
pivot						down	up						swappivot &down
(24	08	16	06	04	02)	**38**	(56	57	58	79	70	45)	
pivot					down	up							swappivot &down
(02	08	16	06	04)	**24**								
pivot, down	up												swappivot &down
02	(08	16	06	04)									
	pivot	up		down									swap up& down
	pivot	04		16									
	pivot		down	Up									
	(06	04)	**08**	(16)									swappivot &down
	pivot	down	up										
	(04)	**06**											swappivot &down
	04 pivot, down, up												
				16 pivot, down, up									
(02	**04**	**06**	**08**	**16**	**24)**	38							

							(56	57	58	79	70	45**)**	
							pivot	up				down	swap up & down
							pivot	45				57	
							pivot	down	up				swappivot &down
							(45)	**56**	(58	79	70	57)	
							45 pivot, down, up						swappivot &down
									(58 pivot	79 up	70	57) down	swap up& down
										57		79	
										down	up		
									(57)	**58**	(70	79)	swap pivot &down
									57 pivot, down, up				
											(70	79)	
											pivot, down	up	swappivot &down
											70		
												79 pivot, down, up	
							(45	**56**	**57**	**58**	**70**	**79)**	
02	**04**	**06**	**08**	**16**	**24**	**38**	**45**	**56**	**57**	**58**	**70**	**79**	

RecursiveprogramforQuickSort:

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

voidquicksort(int,int); int
partition(int, int);
voidinterchange(int,int);
int array[25];

int main()
{
        intnum,i=0; clrscr();
        printf("Enterthenumberofelements:"); scanf(
        "%d", &num);
        printf("Entertheelements:");
        for(i=0; i < num; i++)
                scanf("%d",&array[i]);
        quicksort(0, num -1);
        printf("\nTheelementsaftersortingare:");
```

```
        for(i=0;i<num;i++)
                printf("%d",array[i]); return
        0;
}

voidquicksort(intlow,inthigh)
{
        int
        pivotpos;if(low
        <high)
        {
                pivotpos=partition(low,high+1);
                quicksort(low, pivotpos - 1);
                quicksort(pivotpos + 1, high);
        }
}

intpartition(intlow,inthigh)
{
        intpivot=array[low];
        intup=low,down=high;

        do
        {
                do
                        up =up +1;
                while(array[up]<pivot);

                do
                        down = down - 1;
                while(array[down]>pivot);

                if(up<down)
                        interchange(up,down);

        } while(up < down);
        array[low]=array[down];
        array[down] = pivot;
        return down;
}

voidinterchange(inti,intj)
{
        int temp;
        temp = array[i];
        array[i]=array[j];
        array[j] = temp;
}
```