Program 8:

Design and implement C/C++ program to find a subset of a given set S={S1, S2, ……..., Sn} of n positive integers whose sum is equal to a given positive integer d.

Algorithm:

```
Algorithm SumOfSub(s, k, r)
//Find all subsets of w[1…n] that sum to m. The values of x[j], 1<= j < k, have already
 //been determined. s=∑ j=1 to k-1 w[j]*x[j]  and r =∑ j=k to n w[j]. The w[j]'s are in ascending order.

{
    x[k] ← 1  //generate left child
    if (s+w[k] = m)
            write (x[1...n])  //subset found
    else if ( s + w[k]+w[k+1] <= m)
            SumOfSub( s + w[k], k+1, r-w[k])
    //Generate right child
    if( (s + r - w[k] >= m) and (s + w[k+1] <= m) )
    {
            x[k] ← 0
            SumOfSub( s, k+1, r-w[k] )
    }
}
```

Code:

```
#include<stdio.h>

int x[10],w[10],count,d;

void sum_of_subsets(int s, int k, int rem)

{

x[k]=1;

if(s+w[k]==d)

{

printf("subset=%d\n",++count);

for(int i=0;i<=k;i++)

if(x[i]==1)

printf("%d ",w[i]);
```

```
printf("\n");
}
else
if(s+w[k]+w[k+1]<=d)
sum_of_subsets(s+w[k],k+1,rem-w[k]);
if((s+rem-w[k]>=d)&&(s+w[k+1])<=d)
{
x[k]=0;
sum_of_subsets(s,k+1,rem-w[k]);
}
}
int main()
{
int sum=0,n;
printf("enter number of elements:");
scanf("%d",&n);
printf("enter the elements in increasing order:");
for(int i=0;i<n;i++)
{
scanf("%d",&w[i]);
sum=sum+w[i];
}
printf("enter the sum:");
scanf("%d",&d);
if((sum<d) || (w[0]>d))
printf("No subset possible\n");
else
sum_of_subsets(0,0,sum);
}
```

Output:

```
sru-ubuntu@srujani-Ubuntu-VirtualBox:~$ gcc p8.c
sru-ubuntu@srujani-Ubuntu-VirtualBox:~$ ./a.out
enter number of elements:5
enter the elements in increasing order:1 2 3 4 5
enter the sum:10
subset=1
1 2 3 4
subset=2
1 4 5
subset=3
2 3 5
sru-ubuntu@srujani-Ubuntu-VirtualBox:~$
```

Program 9:

Design and implement C/C++ program to sort a given set of n integer elements using selection sort method and compute its time complexity. Run the program for varied values of

n>5000 and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

Algorithm:

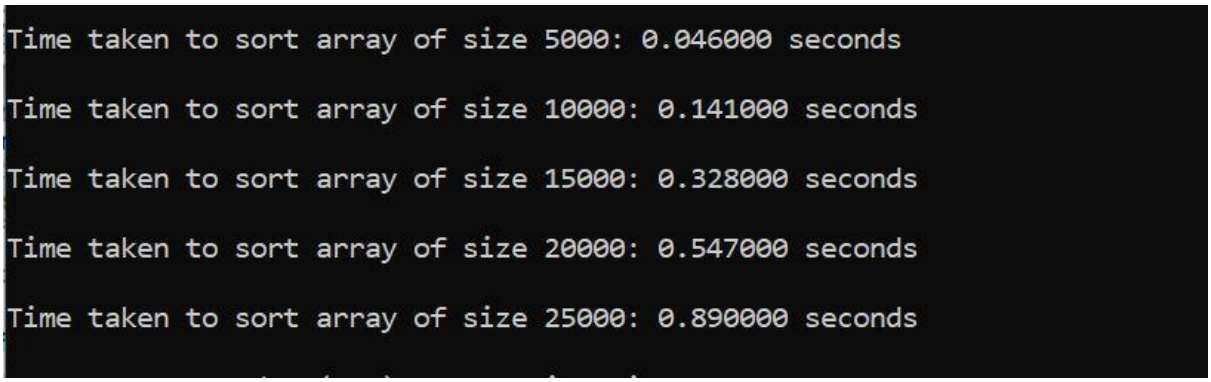
ALGORITHM *SelectionSort*$(A[0..n-1])$
//Sorts a given array by selection sort
//Input: An array $A[0..n-1]$ of orderable elements
//Output: Array $A[0..n-1]$ sorted in ascending order
for $i \leftarrow 0$ **to** $n-2$ **do**
    $min \leftarrow i$
    for $j \leftarrow i+1$ **to** $n-1$ **do**
        if $A[j] < A[min]$ $min \leftarrow j$
    swap $A[i]$ and $A[min]$

Code:

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
void selectionSort(int arr[], int n)
{
int i,j,min_idx;
for(i=0;i<n-1;i++)
{
min_idx=i;
for(j=i+1;j<n;j++)
{
if(arr[j]<arr[min_idx])
{
min_idx=j;
}
}
```

```
int temp=arr[min_idx];
arr[min_idx]=arr[i];
arr[i]=temp;
}
}
int main()
{
int n,i;
clock_t start, end;
double cpu_time_used;
int sizes[]={5000,10000,15000,20000,25000};
for(i=0;i<sizeof(sizes)/sizeof(sizes[0]);i++)
{
n=sizes[i];
int arr[n];
srand(time(NULL));
for(int j=0;j<n;j++)
{
arr[j]=rand();
}
start=clock();
selectionSort(arr, n);
end=clock();
cpu_time_used=((double)(end-start)) / CLOCKS_PER_SEC;
printf("\n Time taken to sort array of size %d: %f seconds\n", n, cpu_time_used);
}
return 0;
}
```

Output:



Program 10:

Design and implement C/C++ program to sort a given set of n integer elements using Quick Sort method and compute its time complexity. Run the program for varied values of n>5000

and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

Algorithm:

ALGORITHM *Quicksort*$(A[l..r])$
//Sorts a subarray by quicksort
//Input: A subarray $A[l..r]$ of $A[0..n-1]$, defined by its left and right indices
// l and r
//Output: Subarray $A[l..r]$ sorted in nondecreasing order
if $l < r$
  $s \leftarrow$ *Partition*$(A[l..r])$ //s is a split position
  Quicksort$(A[l..s-1])$
  Quicksort$(A[s+1..r])$

ALGORITHM *Partition*$(A[l..r])$
//Partitions a subarray by using its first element as a pivot
//Input: A subarray $A[l..r]$ of $A[0..n-1]$, defined by its left and right
// indices l and r $(l < r)$
//Output: A partition of $A[l..r]$, with the split position returned as
// this function's value
$p \leftarrow A[l]$
$i \leftarrow l$; $j \leftarrow r+1$
repeat
  repeat $i \leftarrow i+1$ **until** $A[i] \geq p$
  repeat $j \leftarrow j-1$ **until** $A[j] \leq p$
  swap$(A[i], A[j])$
until $i \geq j$
swap$(A[i], A[j])$ //undo last swap when $i \geq j$
swap$(A[l], A[j])$
return j

Code:

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/time.h>
#include<time.h>
void fnGenRandInput(int[], int);
void fnDispArray(int[], int);
int fnPartition(int[], int, int);
void fnQuickSort(int [], int, int);
```

```
void fnSwap(int*, int*);
void fnSwap(int *a, int *b)
{
int t=*a;
*a=*b;
*b=t;
}
int main(int argc, char **argv)
{
FILE *fp;
struct timeval tv;
double dStart, dEnd;
int iaArr[500000],iNum,i,iChoice;
for(;;)
{
printf("\n1.Plot the Graph\n2.QuickSort\n3.Exit");
printf("\nEnter your choice\n");
scanf("%d",&iChoice);
switch(iChoice)
{
case 1:
fp=fopen("QuickPlot.dat","w");
for(i=100;i<100000;i+=100)
{
fnGenRandInput(iaArr,i);
gettimeofday(&tv,NULL);
dStart=tv.tv_sec+(tv.tv_usec/1000000.0);
fnQuickSort(iaArr,0,i-1);
gettimeofday(&tv,NULL);
dEnd=tv.tv_sec+(tv.tv_usec/1000000.0);
```

```
fprintf(fp,"%d\t%lf\n",i,dEnd-dStart);
}
fclose(fp);
printf("\nData File generated and stored in file<QuickPlot.dat>.\n Use a plotting utility\n");
break;
case 2:
printf("\nEnter the number of elements to sort\n");
scanf("%d",&iNum);
printf("\nUnsorted Array\n");
fnGenRandInput(iaArr,iNum);
fnDispArray(iaArr,iNum);
fnQuickSort(iaArr,0,iNum-1);
printf("\nSorted Array\n");
fnDispArray(iaArr,iNum);
break;
case 3:
exit(0);
}
}
return 0;
}
int fnPartition(int a[], int l, int r)
{
int i,j;
int p;
p=a[l];
i=l;
j=r+1;
do
{
```

```
do{i++;}
while(a[i]<p);
do{j--;}
while(a[j]>p);
fnSwap(&a[i],&a[j]);
}
while(i<j);
fnSwap(&a[i],&a[j]);
fnSwap(&a[l],&a[j]);
return j;
}
void fnQuickSort(int a[], int l, int r)
{
int s;
if(l<r)
{
s=fnPartition(a,l,r);
fnQuickSort(a,l,s-1);
fnQuickSort(a,s+1,r);
}
}
void fnGenRandInput(int X[], int n)
{
srand(time(NULL));
for(int i=0;i<n;i++)
{
X[i]=rand()%10000;
}
}
void fnDispArray(int X[], int n)
```

```
{
for(int i=0;i<n;i++)
printf("%5d\n",X[i]);
}
```

Output:

```
Enter the number of elements to sort
4

Unsorted Array
   581
  3498
  4832
  4542

Sorted Array
   581
  3498
  4542
  4832

1.Plot the Graph
2.QuickSort
3.Exit
Enter your choice
1

Data File generated and stored in file < QuickPlot.dat >.
 Use a plotting utility

1.Plot the Graph
2.QuickSort
3.Exit
```