Program 1:Write a program to sort a list of N elements using Selection Sort Technique.

#include<stdio.h>//This line includes the standard input-output library, which allows the program to use functions like printf and scanf.

#include<conio.h>//This line includes the console input-output library, which is often used for functions like clrscr() and getch()

```
int selection_sort(int a[10],int n);

void main()

{

 int a[10],n,i;

 clrscr();//avilable in conio .h library

 printf("enter n\n");//asking users to enter the number of elements to sort

 scanf("%d",&n);//reads the integer input and stores it in the n variable

 printf("enter an array\n");prompts users to enter the array elements

 for(i=0;i<n;i++)//loop iterates from n-n-1

 scanf("%d",&a[i]);//storing in a variable a

 selection_sort(a,n);//This line calls the selection_sort function, passing the array a and the number of elements n to sort the array.

 printf("sorted array is\n");

 for(i=0;i<n;i++)//sorting an array and print element

 printf("%d\n",a[i]);

 getch();

}

int selection_sort(int a[20],int n)

{

 int i,j,temp,pos;//pos will track the array index value of min element

 for(i=0;i<n-1;i++)//This loop iterates from 0 to n-2, meaning it will run for n-1 iterations. Each iteration will find the smallest element in the unsorted portion of the array.

{
```

```
    pos=i;//array pos assigning
      for(j=i+1;j<n;j++)//helps to check unsorted array to find which is smallest elemnt
 {
  if(a[j]<a[pos])
  pos=j;
 }
 temp=a[pos];
 a[pos]=a[i];
 a[i]=temp;
}
return 0;
}
```

Algorithm for Selection sort

Algorithm SelectionSort (a [], n)

//Purpose: Sort the given elements using selection sort

//Inputs:

n- The number of items present in the array

a- The item to be sorted are present in the array.

//Outputs:

a- contains the sorted list.

for i ← 0 to n-1 do

pos ← i

for j ← i+1 to n do

//Assume ith element as smallest

// Find the position of the smallest item

if (a[j]<a[pos])

pos ← j;

end for

temp ← a[pos]

a[pos] ← a[i]

a[i] ← temp

end for

// exchange ith item with smallest element

Iteration Breakdown

First Iteration (i = 0)

Initialization: pos = 0 (value is 34).

Inner Loop:

j = 1: Compare 12 with 34 → Update pos = 1.

j = 2: Compare 5 with 12 → Update pos = 2.

j = 3: Compare 67 with 5 → No change.

j = 4: Compare 23 with 5 → No change.

j = 5: Compare 89 with 5 → No change.

j = 6: Compare 45 with 5 → No change.

Swap: Swap a (34) with a (5).

Array after first iteration: {5, 12, 34, 67, 23, 89, 45}.

Second Iteration (i = 1)

Initialization: pos = 1 (value is 12).

Inner Loop:

j = 2: Compare 34 with 12 → No change.

j = 3: Compare 67 with 12 → No change.

j = 4: Compare 23 with 12 → No change.

j = 5: Compare 89 with 12 → No change.

j = 6: Compare 45 with 12 → No change.

Swap: No swap needed since pos remains 1.

Array after second iteration: {5, 12, 34, 67, 23, 89, 45}.

Third Iteration (i = 2)

Initialization: pos = 2 (value is 34).

Inner Loop:

j = 3: Compare 67 with 34 → No change.

j = 4: Compare 23 with 34 → Update pos = 4.

j = 5: Compare 89 with 23 → No change.

j = 6: Compare 45 with 23 → No change.

Swap: Swap a (34) with a (23).

Array after third iteration: {5, 12, 23, 67, 34, 89, 45}.

Fourth Iteration (i = 3)

Initialization: pos = 3 (value is 67).

Inner Loop:

j = 4: Compare 34 with 67 → Update pos = 4.

j = 5: Compare 89 with 34 → No change.

j = 6: Compare 45 with 34 → No change.

Swap: Swap a (67) with a (34).

Array after fourth iteration: {5, 12, 23, 34, 67, 89, 45}.

Fifth Iteration (i = 4)

Initialization: pos = 4 (value is 67).

Inner Loop:

j = 5: Compare 89 with 67 → No change.

j = 6: Compare 45 with 67 → Update pos = 6.

Swap: Swap a (67) with a (45).

Array after fifth iteration: {5, 12, 23, 34, 45, 89, 67}.

Sixth Iteration (i = 5)

Initialization: pos = 5 (value is 89).

Inner Loop:

j = 6: Compare 67 with 89 → Update pos = 6.

Swap: Swap a (89) with a (67).

Array after sixth iteration: {5, 12, 23, 34, 45, 67, 89}.

End of Sorting

After all iterations, the array is sorted:

text

{5, 12, 23, 34, 45, 67, 89}

Example Input with Negative Values

Let's say the user inputs the following elements to sort:

text

Enter n:

6

Enter an array:

64

-25

12

-22

11

-5

Initial Array

The initial array is:

text

[64, -25, 12, -22, 11, -5]

Step-by-Step Sorting Process

1. First Iteration (i = 0)

- **Current Array**: [64, -25, 12, -22, 11, -5]
- **Unsorted Portion**: [64, -25, 12, -22, 11, -5]
- **Find Minimum**:
    - Compare 64 (index 0) with -25 (index 1) → -25 is smaller.
    - Compare -25 with 12 (index 2) → -25 remains smaller.
    - Compare -25 with -22 (index 3) → -25 remains smaller.
    - Compare -25 with 11 (index 4) → -25 remains smaller.
    - Compare -25 with -5 (index 5) → -25 remains smaller.
- **Minimum Found**: -25 (index 1)
- **Swap**: Swap -25 with 64 (index 0).

Array After First Iteration:

[-25, 64, 12, -22, 11, -5]

2. Second Iteration (i = 1)

- **Current Array**: [-25, 64, 12, -22, 11, -5]
- **Unsorted Portion**: [64, 12, -22, 11, -5]
- **Find Minimum**:
    - Compare 64 (index 1) with 12 (index 2) → 12 is smaller.
    - Compare 12 with -22 (index 3) → -22 is smaller.
    - Compare -22 with 11 (index 4) → -22 remains smaller.
    - Compare -22 with -5 (index 5) → -22 remains smaller.
- **Minimum Found**: -22 (index 3)
- **Swap**: Swap -22 with 64 (index 1).

Array After Second Iteration:

text

[-25, -22, 12, 64, 11, -5]

3. Third Iteration (i = 2)

- **Current Array**: [-25, -22, 12, 64, 11, -5]
- **Unsorted Portion**: [12, 64, 11, -5]
- **Find Minimum**:
    - Compare 12 (index 2) with 64 (index 3) → 12 remains smaller.
    - Compare 12 with 11 (index 4) → 11 is smaller.
    - Compare 11 with -5 (index 5) → -5 is smaller.
- **Minimum Found**: -5 (index 5)
- **Swap**: Swap -5 with 12 (index 2).

Array After Third Iteration:

text

[-25, -22, -5, 64, 11, 12]

4. Fourth Iteration (i = 3)

- **Current Array**: [-25, -22, -5, 64, 11, 12]
- **Unsorted Portion**: [64, 11, 12]
- **Find Minimum**:
    - Compare 64 (index 3) with 11 (index 4) → 11 is smaller.

    - Compare 11 with 12 (index 5) → 11 remains smaller.
- **Minimum Found**: 11 (index 4)
- **Swap**: Swap 11 with 64 (index 3).

Array After Fourth Iteration:

text

[-25, -22, -5, 11, 64, 12]

5. Fifth Iteration (i = 4)

- **Current Array**: [-25, -22, -5, 11, 64, 12]
- **Unsorted Portion**: [64, 12]
- **Find Minimum**:
    - Compare 64 (index 4) with 12 (index 5) → 12 is smaller.
- **Minimum Found**: 12 (index 5)
- **Swap**: Swap 12 with 64 (index 4).

Array After Fifth Iteration:

text

[-25, -22, -5, 11, 12, 64]

Final Sorted Array

After completing all iterations, the final sorted array is:

text

[-25, -22, -5, 11, 12, 64]

Example 1: SELECTION SORT In this sorting method first find the smallest element in the list and exchange that with first element in the list. Find the second smallest element in the list and exchange with second element of the list and so on. Finally all the elements will be arranged in ascending order. Since, the next least item is selected and exchanged appropriately so that elements are finally sorted, this technique is called Selection sort.

Overview of Selection Sort

The **Selection Sort** algorithm sorts an array by repeatedly finding the minimum element from the unsorted portion and moving it to the beginning. It works as follows:

1. Start with the first element of the array.
2. Search for the smallest element in the remaining unsorted portion of the array.
3. Swap the smallest found element with the first element of the unsorted portion.

4. Move the boundary of the sorted and unsorted portions one element to the right.
5. Repeat steps until the entire array is sorted.