NAME: AMAN RAYCHAUDHURI

DIVISION: B2
ROLL NO.: 2401094 (B2-30)
BRANCH: AIDS
FE

Experiment No. 1

Aim:
Write a program to read N random numbers and evaluate the sum and average of the same

Theory:

1. Input:

The program first asks the user how many random numbers (N) they want to input. Then, it uses a loop to read N numbers. Each number is stored in the variable num.

2. Sum Calculation:

The program keeps a running total of the sum in the sum variable.

3. Average Calculation:

Once all the numbers are entered, the program calculates the average by dividing the total sum by N.

4. Output:

The program prints both the sum and the average of the numbers.

Program:

```
#include <stdio.h>

int main()
  { int N;
  double sum = 0.0, average;

  printf("Enter the number of random numbers: ");
  scanf("%d", &N);

  for (int i = 1; i <= N; i++)
    { double num;
    printf("Enter number %d:
    ", i); scanf("%lf", &num);
    sum += num;
  }

  average = sum / N;

  printf("The sum of the numbers is: %.2lf\n", sum);
  printf("The average of the numbers is: %.2lf\n",
  average);

  return 0;
}
```

NAME: AMAN RAYCHAUDHURI

DIVISION: B2
ROLL NO.: 2401094 (B2-30)
BRANCH: AIDS
FE

Output:

```
Enter the number of random numbers: 6
Enter number 1: 5
Enter number 2: 7
Enter number 3: 6
Enter number 4: 7
Enter number 5: 3
Enter number 6: 4
The sum of the numbers is: 32.00
The average of the numbers is: 5.33


...Program finished with exit code 0
Press ENTER to exit console.
```

Conclusion:

This program reads N random numbers from the user, computes the sum, and calculates the average. It demonstrates basic input handling, loops, and arithmetic operations in C. The program ensures the user can handle any number of values dynamically based on the input.

NAME: AMAN RAYCHAUDHURI

DIVISION: B2
ROLL NO.: 2401094 (B2-30)
BRANCH: AIDS
FE

Experiment No. 2

AIM

Program to find the biggest of an array of integers

Theory:

1. Input:

The program first asks for the number of elements in the array.
It then takes input for each integer and stores it in the array arr[].

2. Finding the Largest Element:

We initialize biggest with the first element of the array (arr[0]).
A loop is used to iterate through the array. If any element is greater than the current biggest, it updates biggest.

3. Output:

The program prints the largest number in the array.

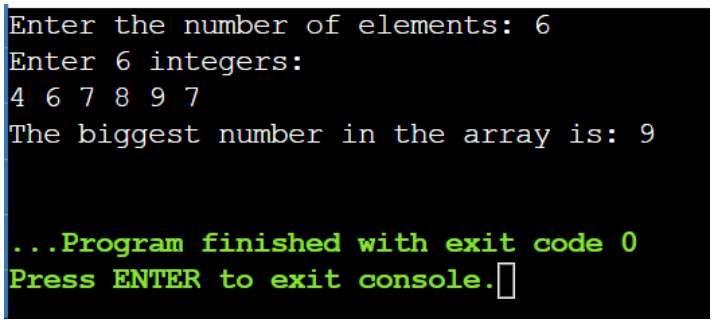
Program:

```
#include
<stdio.h> int
main() {
   int n;

   printf("Enter the number of elements: ");
   scanf("%d", &n);

   int arr[n];

   printf("Enter %d
   integers:\n", n); for (int i = 0;
   i < n; i++) {
      scanf("%d", &arr[i]);
   }

       int biggest =
            arr[0];
   for (int i = 1; i < n; i++)
    { if (arr[i] > biggest) {
        biggest = arr[i];
      }
   }

   printf("The biggest number in the array is: %d\n", biggest);

   return 0;
}
```

NAME: AMAN RAYCHAUDHURI

DIVISION: B2
ROLL NO.: 2401094 (B2-30)
BRANCH: AIDS
FE

Output:



Conclusion:

This program efficiently finds the largest number in an array by iterating through the array once. It uses a simple comparison to track the largest number, demonstrating how loops and conditional checks work together in C.

NAME: AMAN RAYCHAUDHURI

DIVISION: B2
ROLL NO.: 2401094 (B2-30)
BRANCH: AIDS
FE

Experiment
No. 3
AIM

Program to find the Smallest of an array of integers

Theory:

1. Input:

The program first asks for the number of elements in the array.
It then takes input for each integer and stores it in the array
arr[].

2. Finding the Smallest Element:

The smallest variable is initialized with the first element of the array (arr[0]).
A loop is used to iterate through the array. If any element is smaller than the current smallest, it updates smallest.

3. Output:

The program prints the smallest number in the array.

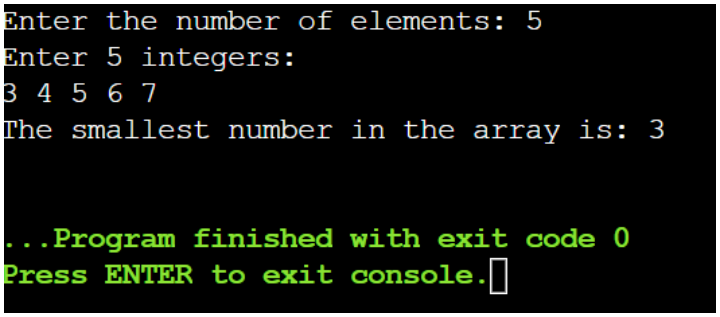
Program:

```
#include
<stdio.h> int
main() {
   int n;

   printf("Enter the number of elements: ");
   scanf("%d", &n);

   int arr[n];

   printf("Enter %d
   integers:\n", n); for (int i = 0;
   i < n; i++) {
      scanf("%d", &arr[i]);
   }


   int smallest = arr[0];
   for (int i = 1; i < n;
   i++) {
      if (arr[i] < smallest)
         { smallest =
         arr[i];
      }
   }

   printf("The smallest number in the array is: %d\n", smallest);

   return 0;
```

NAME: AMAN RAYCHAUDHURI

DIVISION: B2
ROLL NO.: 2401094 (B2-30)
BRANCH: AIDS
FE

Output:



Conclusion:

This program finds the smallest number in an array by comparing each element with the current smallest value. It efficiently tracks the smallest number using a simple loop and comparison. This approach can be used for any size of the array as long as it contains integers.

NAME: AMAN RAYCHAUDHURI

DIVISION: B2
ROLL NO.: 2401094 (B2-30)
BRANCH: AIDS
FE

Experiment No. 4

Aim: Write Program to find the biggest and smallest element and interchange them in the initialized array

Theory:

1. Input:

The program asks for the number of elements in the array (n). It then reads n integers from the user to initialize the array.

2. Finding the Biggest and Smallest:

The program initializes biggest and smallest to the first element of the array. It then loops through the array to find the actual biggest and smallest values and their corresponding indices.

3. Interchanging the Biggest and Smallest:

The program swaps the values at the indices of biggest and smallest using a temporary variable temp.

4. Output:

The program prints the modified array after the biggest and smallest elements have been interchanged.

Program:

```
#include
<stdio.h> int
main() {
   int n;

   printf("Enter the number of elements: ");
   scanf("%d", &n);

   int arr[n];

   // Initialize the array
   printf("Enter %d
   integers:\n", n); for (int i = 0;
   i < n; i++) {
      scanf("%d", &arr[i]);
   }

   int biggest = arr[0], smallest =
   arr[0]; int biggestIndex = 0,
   smallestIndex = 0;

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

NAME: AMAN RAYCHAUDHURI

DIVISION: B2
ROLL NO.: 2401094 (B2-30)
BRANCH: AIDS
FE

```
        if (arr[i] > biggest)
            { biggest = arr[i];
            biggestIndex = i;
        }
        if (arr[i] < smallest)
            { smallest =
            arr[i];
            smallestIndex =
            i;
        }
    }

    int temp =
    arr[biggestIndex];
    arr[biggestIndex] =
    arr[smallestIndex];
    arr[smallestIndex] = temp;

    printf("\nArray after swapping the biggest and smallest
    elements:\n"); for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }

    printf("\n");

    return 0;
}
```

Output:

```
Enter the number of elements: 3
Enter 3 integers:
2 3 4

Array after swapping the biggest and smallest elements:
4 3 2


...Program finished with exit code 0
Press ENTER to exit console.
```

Conclusion:

This program identifies the biggest and smallest elements in an array, then swaps them. It efficiently handles the task by first finding the indices of the largest and smallest elements and then performing the swap. This approach works well for arrays of any size

NAME: AMAN RAYCHAUDHURI

DIVISION: B2
ROLL NO.: 2401094 (B2-30)
BRANCH: AIDS
FE

Experiment No. 5

Aim:
Ascending and descending using selection sort

Theory:

1. selectionSortAscending():

Finds the smallest element in the unsorted portion of the array and swaps it with the first unsorted element.
Repeats the process for all elements except the last one.

2. selectionSortDescending():

Finds the largest element in the unsorted portion of the array and swaps it with the first unsorted element.
Repeats the process for all elements except the last one.

3. printArray():

A utility function to print the array elements.

4. main():

The main function asks for the size of the array, takes input, sorts the array in ascending and descending order, and prints the results.

Program:

```
#include <stdio.h>

void selectionSortAscending(int arr[], int n)
  { for (int i = 0; i < n - 1; i++) {
     int minIndex = i;
     for (int j = i + 1; j < n; j++)
        { if (arr[j] <
        arr[minIndex]) {
           minIndex = j;
        }
     }

     if (minIndex != i) {
        int temp =
        arr[i];
        arr[i] = arr[minIndex];
        arr[minIndex] =
        temp;
     }
  }
}void selectionSortDescending(int arr[], int n) {
 for (int i = 0; i < n - 1; i++)
```

NAME: AMAN RAYCHAUDHURI

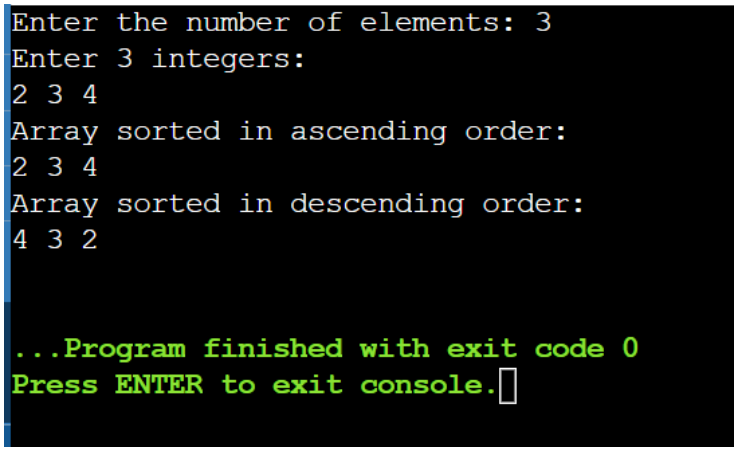
DIVISION: B2
ROLL NO.: 2401094 (B2-30)
BRANCH: AIDS
FE

```
int maxIndex = i;
        for (int j = i + 1; j < n; j++) {
            if (arr[j] > arr[maxIndex])
            {
                maxIndex = j;
            }
        }

        if (maxIndex != i) {
            int temp = arr[i];
            arr[i] = arr[maxIndex];
            arr[maxIndex] =
            temp;
        }
    }
}

void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main()
    { int n;

    printf("Enter the number of elements: ");
    scanf("%d", &n);

    int arr[n];
    printf("Enter %d
    integers:\n", n); for (int i = 0;
    i < n; i++) {
        scanf("%d", &arr[i]);
    }

    selectionSortAscending(arr, n);
    printf("Array sorted in
    ascending order:\n");
    printArray(arr, n);

    selectionSortDescending(arr, n);
    printf("Array sorted in
    descending order:\n");
    printArray(arr, n);

    return 0;
}
```

NAME: AMAN RAYCHAUDHURI

DIVISION: B2
ROLL NO.: 2401094 (B2-30)
BRANCH: AIDS
FE

Output:



Conclusion:

This program demonstrates how Selection Sort works for both ascending and descending orders. The sorting is performed by finding the minimum or maximum element in the unsorted part of the array and swapping it with the first unsorted element. This method works for both small and large arrays efficiently within the time complexity

NAME: AMAN RAYCHAUDHURI

DIVISION: B2
ROLL NO.: 2401094 (B2-30)
BRANCH: AIDS
FE

Experiment No. 6

Aim:
Merge two sorted arrays

Theory:

1. mergeSortedArrays():

This function takes two sorted arrays (arr1 and arr2), their sizes (n1 and n2), and an empty array (merged) to store the result.

It compares elements from both arrays, and places the smaller element into the merged array.

Once one array is exhausted, it copies the remaining elements of the other array into merged.

2. printArray():

A utility function to print the array elements.

3. main():
The main function asks for the sizes and elements of the two sorted arrays.
It then calls mergeSortedArrays() to merge the two arrays and prints the result.

Program:

```
#include <stdio.h>
void mergeSortedArrays(int arr1[], int n1, int arr2[], int n2, int
merged[]) { int i = 0, j = 0, k = 0;

    while (i < n1 && j < n2)
       { if (arr1[i] < arr2[j]) {
          merged[k++] =
          arr1[i++];
       } else {
          merged[k++] = arr2[j++];
       }
    }
    while (i < n1) {
       merged[k++]
       = arr1[i++];
    }
    while (j < n2) {
       merged[k++]
       = arr2[j++];
    }
 }
 void printArray(int arr[], int n){
    for (int i = 0; i < n; i++) {
printf("%d ", arr[i]);
```

NAME: AMAN RAYCHAUDHURI

DIVISION: B2
ROLL NO.: 2401094 (B2-30)
BRANCH: AIDS
FE

```
    }
    printf("\n");
}
int main() {
    int n1,
    n2;

    printf("Enter the number of elements in the first array: ");
    scanf("%d", &n1);
    int arr1[n1];
    printf("Enter %d elements for the first sorted array:\n",
    n1); for (int i = 0; i < n1; i++) {
        scanf("%d", &arr1[i]);
    }
    printf("Enter the number of elements in the second array: ");
    scanf("%d", &n2);
    int arr2[n2];
    printf("Enter %d elements for the second sorted array:\n",
    n2); for (int i = 0; i < n2; i++) {
        scanf("%d", &arr2[i]);
    }
    int merged[n1 + n2];

    mergeSortedArrays(arr1, n1, arr2, n2, merged);

    printf("Merged sorted array:\n");
    printArray(merged, n1 + n2);

    return 0;
}
```

Output:

```
Enter the number of elements in the first array: 2
Enter 2 elements for the first sorted array:
2 3
Enter the number of elements in the second array: 3
Enter 3 elements for the second sorted array:
4 5 6
Merged sorted array:
2 3 4 5 6


...Program finished with exit code 0
Press ENTER to exit console.
```

Conclusion:

This program demonstrates how to merge two sorted arrays into a single sorted array. The merging process compares elements from both arrays and places them in the correct order, making it efficient with a time complexity of , where n1 and n2 are the sizes of the input arrays.