ALGORITHMS LABORATORY
[CS-2098]
Individual Work

Lab. No.- 1 Date. 16/07/2021

Roll Number:	1905083	Branch/Section:	CS-2
Name in Capital:	AMIL UTKARSH GUPTA		

Program No: 1.1

Program Title:
Write a program to store random numbers into an array of n integers and then find out the smallest and largest number stored in it. n is the user input.

Input/Output Screenshots:
RUN-1:

```
PS C:\Algo Lab\Lab-1> gcc .\q1.c
.\q1.c: In function 'main':
.\q1.c:12:14: warning: implicit declaration of function 'rand' [-Wimplicit-function-declaration]
   12 |         a[i]=rand();
      |              ^~~~
PS C:\Algo Lab\Lab-1> ./a
Enter number of elements
5
Array:
41      18467   6334    26500   19169
Smallest number = 41
Largest number = 26500
```

Source code

```
#include<stdio.h>

int main()
{
   int n, i, s, l;
   printf("Enter number of elements\n");
   scanf("%d",&n);
   int a[n];
   printf("Array:\n");
   for(i=0;i<n;i++)
   {
      a[i]=rand();
      printf("%d\t",a[i]);
   }

   printf("\n");

   s=l=a[0];
```

```
    for(i=0;i<n;i++)
    {
        if(a[i] < s)
        {
            s=a[i];
        }
        else if(a[i] > l)
        {
            l=a[i];
        }
    }

    printf("Smallest number = %d \nLargest number = %d\n", s, l);

    return 0;
}
```

Conclusion/Observation

Hence we have made an algorithm to insert n random values into an array and find the largest and smallest value.

Program No: 1.2

Program Title:

Write a program to store random numbers into an array of n integers, where the array must contain some duplicates. Do the following:

a) Find out the total number of duplicate elements.
b) Find out the most repeating element in the array.

Input/Output Screenshots:

RUN-1:

```
PS C:\Algo Lab\Lab-1> gcc .\q2.c
.\q2.c: In function 'insert_random_duplicates':
.\q2.c:8:16: warning: implicit declaration of function 'abs' [-Wimplicit-function-declaration]
    8 |         a[i] = abs(rand()%(n-1));
      |                ^~~
.\q2.c:8:20: warning: implicit declaration of function 'rand' [-Wimplicit-function-declaration]
    8 |         a[i] = abs(rand()%(n-1));
      |                    ^~~~
PS C:\Algo Lab\Lab-1> ./a
Enter number of elements
5
1       3       2       0       1
Number of duplicate elements = 1
Element with maximum duplicates = 1
```

Source code

```
#include<stdio.h>

void insert_random_duplicates(int *a, int n)
{
   int i;
   for(i=0;i<n;i++)
   {
      a[i] = abs(rand()%(n-1));
      printf("%d\t",a[i]);
   }
   printf("\n");
}

int count_and_replace(int *a, int n, int e)
{
   int i, count=0;
   for(i=0; i<n; i++)
   {
      if(a[i] == e)
      {
         count++;
         a[i] = -1;
```

```
        }
    }
    return count;
}

void count_duplicates(int *a, int n)
{
    int i, j, max_dup=0, max_dup_element=a[0], count, count_dup=0, e;
    for(i=0;i<n;i++)
    {
        if(a[i] >= 0)
        {
            e=a[i]; //Doing this because a[i] will become -1 after count_and_replace()
            count = count_and_replace(a, n, e);
            if(count>1)
            {
                count_dup++;
                if(count >  max_dup)
                {
                    max_dup_element = e;
                }
            }
        }
    }
    printf("Number of duplicate elements = %d \nElement with maximum duplicates = %d\n", count_dup,
max_dup_element);
}

int main()
{
    int n;
    printf("Enter number of elements\n");
    scanf("%d",&n);
    int a[n];
    insert_random_duplicates(a, n);
    count_duplicates(a,n);
    return 0;
}
```

Conclusion/Observation

Hence, we have developed algorithms to store n random integers into an array and find the number of duplicate elements and most repeating element.

Program No: 1.3

Program Title:

Write a program to rearrange the elements of an array of n integers such that all even numbers are followed by all odd numbers. How many different ways you can solve this problem. Write your approaches & strategy for solving this problem.

Input/Output Screenshots:

RUN-1:

```
PS C:\Algo Lab\Lab-1> gcc .\q3.c
.\q3.c: In function 'insert_random_positive':
.\q3.c:27:16: warning: implicit declaration of function 'abs' [-Wimplicit-function-declaration]
   27 |         a[i] = abs(rand());
      |                ^~~
.\q3.c:27:20: warning: implicit declaration of function 'rand' [-Wimplicit-function-declaration]
   27 |         a[i] = abs(rand());
      |                    ^~~~
.\q3.c: At top level:
.\q3.c:33:1: warning: return type defaults to 'int' [-Wimplicit-int]
   33 | sort_even_odd(int *a, int n) //Basically, we need to swap the first odd element with the last odd element
      | ^~~~~~~~~~~~
.\q3.c:53:1: warning: return type defaults to 'int' [-Wimplicit-int]
   53 | display_array(int *a, int n)
      | ^~~~~~~~~~~~~
PS C:\Algo Lab\Lab-1> ./a
Enter no. of elements
10
41      18467   6334    26500   19169   15724   11478   29358   26962   24464
24464   26962   6334    26500   29358   15724   11478   19169   18467   41
```

Source code

```
// Approaches and strategies I could think of:

//1. Keep searching for even numbers and inserting them at the start of the array
//OR keep searching for odd numbers and inserting them at the end of the array

//2. Make a new array. Traverse the original array looking for even numbers and put them in the new
//array.
//Then traverse the original array looking for odd numbers and put them in the new array.

//3. Make a new array. Traverse the original array once. Whenever an even number is encountered, insert
//it at the
//start of the new array. Whenever an odd number is encountered, insert it at the end of the new array.

//4. THE ONE THAT I AM USING IN THIS PROGRAM. Take two variables, one will traverse the
//array forward from the
//beginning. The other will go backwards from the end. In each iteration, we use the forward-moving
//variable to find
//the first odd element. If it is found, the backwards-moving variable is used to find the last even
//element.
//The two elements are swapped. We repeat this while the forward-moving variable is at an index less
//than the
//backwards-moving variable.
```

```
//(In other words, while the first odd number exists before the last even number.)
//This is the method of least complexity that I could think of.

#include<stdio.h>

void insert_random_positive(int *a, int n)
{
   int i;
   for(i=0;i<n;i++)
   {
      a[i] = abs(rand());
      printf("%d\t",a[i]);
   }
   printf("\n");
}

sort_even_odd(int *a, int n) //Basically, we need to swap the first odd element with the last odd element
                        //as long as the former appears before the latter
{
   int forward_move=-1, backward_move=n-1, temp;
   while( forward_move < backward_move )
   {
      forward_move++;
      if(a[forward_move] % 2 == 1)
      {
         while( (a[backward_move] % 2 != 0) && ( forward_move < backward_move ) )
         {
            backward_move--;
         }
         temp = a[forward_move];
         a[forward_move] = a[backward_move];
         a[backward_move] = temp;
      }
   }
}

display_array(int *a, int n)
{
   int i;
   for(i=0; i<n; i++)
   {
      printf("%d\t", a[i]);
   }
   printf("\n");
}

int main()
{
```

```
    int n;
    printf("Enter no. of elements\n");
    scanf("%d",&n);
    int a[n];
    insert_random_positive(a, n);
    sort_even_odd(a,n);
    display_array(a,n);
    return 0;
}
```

Conclusion/Observation

Hence, we have developed an algorithm to rearrange an integer array in the desired order.

Program No: 1.4

Program Title:
Write a program that takes three variable (a, b, c) as separate parameters and rotates the values stored so that value a goes to be, b, b to c and c to a by using SWAP(x,y) function that swaps/exchanges the numbers x & y.

Input/Output Screenshots:
RUN-1:

```
PS C:\Algo Lab\Lab-1> gcc .\q4.c
PS C:\Algo Lab\Lab-1> ./a
Enter:
a  b  c
1  2  3
Values after rotating are:
a       b       c
3       1       2
```

Source code

```
#include<stdio.h>

void SWAP(int *x, int *y)
{
   *x = *x + *y;
   *y = *x - *y;
   *x = *x - *y;
}

int main()
{
   int a, b, c;
   printf("Enter:\na  b  c\n");
   scanf("%d%d%d",&a,&b,&c);
   SWAP(&a, &b);
   SWAP(&a, &c);
   printf("Values after rotating are: \na\tb\tc \n%d\t%d\t%d\n", a, b, c);
   return 0;
}
```

Conclusion/Observation

Hence, we have developed an algorithm to rotate the three values using only the function SWAP.

Program No: 1.5

Program Title:

Let A be n*n square matrix array. WAP by using appropriate user defined functions for the following:

a) Find the number of nonzero elements in A
b) Find the sum of the elements above the leading diagonal.
c) Display the elements below the minor diagonal.
d) Find the product of the diagonal elements.

Input/Output Screenshots:

RUN-1:

```
PS C:\Algo Lab\Lab-1> gcc .\q5.c
PS C:\Algo Lab\Lab-1> ./a
Enter the size of the square matrix
3
Enter matrix elements
1 0 0
5 3 0
0 4 2
Number of non-zero elements = 5

Sum of elements above leading diagonal = 0

Elements below the minor diagonal are:
0       4       2
Product of diagonal elements (leading and minor) = 0
```

Source code

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

int num_non_zero(int **a, int n) //Counts the number of non-zero elements in the matrix
{
   int i, j, count=0;
   for(i=0; i<n; i++)
   {
      for(j=0; j<n; j++)
      {
         if( a[i][j] != 0 )
         {
            count++;
```

```
            }
        }
    }
    return count;
}

int find_sum_above_leading(int **a, int n) //Finds the sum of the elements above the leading diagonal
{
    int i, j, sum=0;
    for(i=0; i<n; i++)
    {
        for(j=i+1; j<n; j++)
        {
            sum += a[i][j];
        }
    }

    return sum;
}

void display_elements_below_minor(int **a, int n) //Displays the elements below the minor diagonal
{
    int i, j;
    for(i=1; i<n; i++)
    {
        for(j=n-i; j<n; j++)
        {
            printf("%d\t",a[i][j]);
        }
    }
    printf("\n");
}

int find_product_of_diagonals(int **a, int n) //Finds the product of diagonal elements
{
    int i, prod=1;
    for(i=0; i<n; i++)
    {
        prod *= a[i][i];
        if( (i != (n/2)) || (n%2==0) ) //The diagonals will intersect if n is odd,
                                       // in which case we avoid multiplying the intersecting element twice
        {
            prod *= a[i][n-i-1];
        }
    }
    return prod;
}
```

```
int main()
{
   int n, i , j;
   printf("Enter the size of the square matrix\n");
   scanf("%d", &n);
   int **a = (int**) malloc( sizeof(int*) * n );

   printf("Enter matrix elements\n");
   for(i=0; i<n; i++)
   {
      a[i] = (int*) malloc( sizeof(int) * n );
      for(j=0; j<n; j++)
      {
         scanf("%d", &a[i][j]);
      }
   }

   printf("Number of non-zero elements = %d\n\n", num_non_zero(a, n));
   printf("Sum of elements above leading diagonal = %d\n\n", find_sum_above_leading(a, n));
   printf("Elements below the minor diagonal are:\n");
   display_elements_below_minor(a, n);
   printf("Product of diagonal elements (leading and minor) = %d\n", find_product_of_diagonals(a, n));

   return 0;
}
```

Conclusion/Observation

Hence, we have developed an algorithm to meet the required specifications.

Program No: 1.6

Program Title:

Write a program to find out the second smallest and second largest element stored in an array of n integers. n is the user input. The array takes input through random number generation within a given range. How many different ways you can solve this problem. Write your approaches & strategy for solving this problem.

Input/Output Screenshots:

RUN-1:

```
PS C:\Algo Lab\Lab-1> gcc .\q6.c
.\q6.c: In function 'insert_random_within_range':
.\q6.c:21:17: warning: implicit declaration of function 'rand' [-Wimplicit-function-declaration]
   21 |         a[i] = (rand() % (u - l + 1)) + l;
      |                 ^~~~
PS C:\Algo Lab\Lab-1> ./a
Enter no. of elements
5
Enter range
Lower_limit     Upper_limit
10              100
Array is:
51      95      65      29      69
Second smallest element = 51
Second largest element = 69
```

Source code

```
//Approaches and strategies I could think of:

//1. Sort the array(lets say in ascending order), find the first value that is not equal to a[0]
//while traversing forward from the start and the first value that is not equal to a[n-1] while traversing
//backwards
//from the end. These are the second smallest and second largest element respectively.
//If one of the values is found, we know that the other will exist as well. So, we check for it and print the
//results accordingly.

//2. THE ONE THAT I AM USING IN THIS PROGRAM. Take four variables, for finding the smallest,
//largest, second smallest
//and second largest elements. Traverse once through the array and keep updating the values of these
variables
//as necessary. Check for the existence of the required values as mentioned in method 1 and print the
//results
//accordingly.

#include<stdio.h>

void insert_random_within_range(int *a, int n, int l, int u)
{
   int i;
```

```
    for(i=0; i<n; i++)
    {
        a[i] = (rand() % (u - l + 1)) + l;
    }
}

int find_second_largest_smallest(int *a, int n)
{
    int i, sm= a[0], ssm= a[0], lar= a[0], slar= a[0], exists=0;
    for(i=1; i<n; i++)
    {
        if(a[i] < sm) //New smallest number found, so the current smallest number will now be the second
//smallest
        {
            ssm = sm;
            sm = a[i];
            exists=1;
        }
        else if((a[i] > sm) && (a[i] < ssm)) //Any number between the smallest and second smallest
                                    // is the new second smallest element
        {
            ssm=a[i];
            exists=1;
        }

        //If the second smallest number exists, then we can be sure that the second largest number also
//exists
        //So we don't need to write exists=1 in the following blocks

        if(a[i] > lar) //New largest number found, so the current largest number will now be the second
largest
        {
            slar = lar;
            lar = a[i];
        }

        else if((a[i] < lar) && (a[i] > slar)) //Any number between the largest and second largest
                                    // is the new second largest element
        {
            slar=a[i];
        }
    }

    if(exists)
    {
        printf("Second smallest element = %d \nSecond largest element = %d\n", ssm, slar);
        return 1;
    }
```

```
    else
    {
        return 0;
    }
}

int main()
{
    int i, n, l, u;
    printf("Enter no. of elements\n");
    scanf("%d", &n);
    int a[n];
    printf("Enter range\nLower_limit\tUpper_limit\n");
    scanf("%d%d", &l, &u);
    insert_random_within_range(a, n, l, u);

    printf("Array is:\n");
    for(i=0; i<n; i++)
    {
        printf("%d\t",a[i]);
    }
    printf("\n");

    if(! find_second_largest_smallest(a, n)) //The function will return 0(false) if the required elements are
//not found
    {
        printf("Second largest and second smallest elements do not exist\n");
    }
    return 0;
}
```

Conclusion/Observation

Hence, we have developed an algorithm to find the second smallest and second largest elements in an array.

Program No: 1.7

Program Title:
Write a program to swap pair of elements of an array of n integers from starting. If n is odd, then last number will be remain unchanged.

Input/Output Screenshots:
RUN-1:

```
PS C:\Algo Lab\Lab-1> gcc .\q7.c
PS C:\Algo Lab\Lab-1> ./a
Enter number of elements
5
Enter array elements
1 2 3 4 5
2       1       4       3       5
```

Source code

```
#include<stdio.h>

int main()
{
   int n, i, temp;

   // Input the array
   printf("Enter number of elements\n");
   scanf("%d", &n);
   int a[n];
   printf("Enter array elements\n");
   for(i=0; i<n; i++)
   {
      scanf("%d", &a[i]);
   }

   //Swap pairs of elements
   for(i=1; i<n; i += 2)
   {
      temp = a[i];
      a[i] = a[i-1];
      a[i-1] = temp;
   }

   //Display the array
   for(i=0; i<n; i++)
   {
```

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

    return 0;
}
```

Conclusion/Observation

Hence, we have developed an algorithm to swap pairs of elements in an array in the desired fashion.

Program No: 1.8

Program Title:

Write a program to display an array of n integers (n>1), where at every index of the array should contain the product of all elements in the array except the element at the given index. Solve this problem by taking single loop and without an additional array.
Input Array : 3 4 5 1 2
Output Array :40 30 24 120 60

Input/Output Screenshots:

RUN-1:

```
PS C:\Algo Lab\Lab-1> gcc .\q8.c
PS C:\Algo Lab\Lab-1> ./a
Enter number of elements
5
Enter array elements
3 4 5 1 2
40      30      24      120     60
```

Source code

```
#include<stdio.h>

int main()
{
   int n, i, p;

   // Input the array
   printf("Enter number of elements\n");
   scanf("%d", &n);
   int a[n];
   printf("Enter array elements\n");
   for(i=0; i<n; i++)
   {
      scanf("%d", &a[i]);
   }

   // Find output array
   p = 1;
   for(i=0; i<n; i++)
   {
      p *= a[i];
   }
   for(i=0; i<n; i++)
   {
```

```
        a[i] = p / a[i];
    }

    // Print output array
    for(i=0; i<n; i++)
    {
        printf("%d\t", a[i]);
    }
    printf("\n");

    return 0;
}
```

Conclusion/Observation

Hence, we have developed an algorithm to convert the array as required.

Program No: 1.9

Program Title:

Write a program using a function for computing ⌞√n⌟ for any positive integer. Besides assignment and comparison, your algorithm may only use the four basic arithmetic operations.

Hints: In number theory, the integer square root (isqrt) of a positive integer n is the positive integer m which is the greatest integer less than or equal to the square root of n, isqrt(n)=⌞√n⌟

Input/Output Screenshots:

RUN-1:

```
PS C:\Algo Lab\Lab-1> gcc .\q9.c
PS C:\Algo Lab\Lab-1> ./a
Enter a number
10
isqrt(10) = 3
```

Source code

```
#include<stdio.h>

int main()
{
   int n, i;
   printf("Enter a number\n");
   scanf("%d", &n);
   for(i=0; i<n; i++)
   {
      if((i * i <= n) && ((i+1) * (i+1) > n)) // Can also use (n/i == i), but that will use the limitations of
                                              // int datatype instead of logic
      {
         printf("isqrt(%d) = %d\n", n ,i);
      }
   }

   return 0;
}
```

Conclusion/Observation

Hence, we have developed an algorithm to find the integer square root of a number.

Program No: 1.10

Program Title:

Assume that you are given a rudimentary programming language which contains only four operators, viz., +, −, abs and div. + and − have their usual meanings, while div(a, b) returns the quotient of a/b and abs(a) returns the absolute value of a. Write a program to solve this problem by using a function max(a, b) that takes two integers a and b as input and returns the maximum of the two. Note that you can only use the operators provided; in particular, the constructs "if", "while", or "for" are not available.

Input/Output Screenshots:

RUN-1:

```
PS C:\Algo Lab\Lab-1> gcc .\q10.c
.\q10.c: In function 'max':
.\q10.c:5:12: warning: implicit declaration of function 'div' [-Wimplicit-function-declaration]
    5 |     return div((a + b + abs(a - b)), 2); //Basically, we add the difference(absolute) to the smaller value,
      |            ^~~
.\q10.c:5:25: warning: implicit declaration of function 'abs' [-Wimplicit-function-declaration]
    5 |     return div((a + b + abs(a - b)), 2); //Basically, we add the difference(absolute) to the smaller value,
      |                         ^~~
PS C:\Algo Lab\Lab-1> ./a
Enter 2 integers
35 99
Maximum of the two integers is 99
```

Source code

```
#include<stdio.h>

int max(int a, int b)
{
   return div((a + b + abs(a - b)), 2); //Basically, we add the difference(absolute) to the smaller value,
                                        // thus making it equal to the bigger value. When we add them,
                                        // we get twice the bigger value, so dividing by 2 gives us the required
                                        // solution
}

int main()
{
   int a, b;
   printf("Enter 2 integers\n");
   scanf("%d%d", &a, &b);
   printf("Maximum of the two integers is %d\n", max(a, b));
   return 0;
}
```

Conclusion/Observation

Hence, we have developed an algorithm to find the maximum of two integers using only the specified operations.