INDEX FOR LAB WORKS

Lab No.		**Title**	**Submission Date**	**Signature**	**Remarks**
1.	**a.**	Greatest Common Divisor (GCD) of two numbers	2081-07-27		
	b.	Find nth Fibonacci number			
	c.	Sequential Search			
	d.	Bubble Sort			
	e.	Selection Sort			
	f.	Insertion Sort			
	g.	Min and Max finding in a list (minmax algorithm)			
2.		Binary search using divide and conquer approach			
3.		Maximum and minimum element using divide and conquer approach			
4.	**a.**	Heap Sort			
	b.	Quick Sort			
	c.	Randomized quick Sort			
	d.	Merge Sort			
5.		Job Sequencing with deadline using greedy algorithm			
6.		Floyd Warshall’s algorithm using dynamic programming approach			
7.		Matrix Chain Multiplication using dynamic programming algorithm			

LAB WORKS

1. Write iterative algorithm for the following problem, write program in C/C++. Also analyze their complexity.

a. Greatest Common Divisor (GCD)

Algorithm:

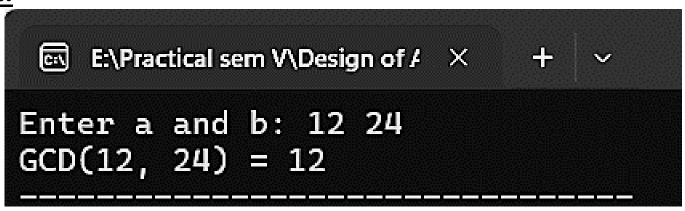
Step 1: Start
Step 2: Read any two numbers, say a and b
Step 3: If b == 0, return the value of a as the answer and terminate
Step 4: If a == 0, return the value of b as the answer and terminate
Step 5: Divide a by b and assign the value of the remainder to r
Step 6: Assign the value of b to a and the value of r to b, then go to step 3
Step 7: Stop.

Source Code:

```
#include<stdio.h>
void GCD(int a, int b) {
    if(a==0)
    printf("GCD(%d, %d) = %d",a, b, b);
    else if(b==0)
    printf("GCD(%d, %d) = %d",a, b, a);
    else{
            printf("GCD(%d, %d) = ",a, b);
            while(b!=0) {
                    int r=a%b;
                    a=b;
                    b=r;
            } printf("%d",a); } }

main() {
    int a, b;
    printf("Enter a and b: ");
    scanf("%d %d", &a, &b);
    GCD(a, b);
    return 0;
}
```

Output:



b. Fibonacci Number

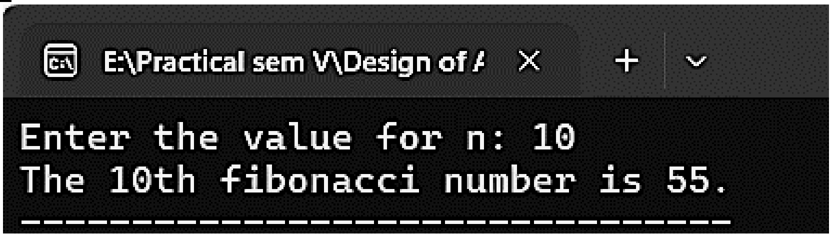
Algorithm:

Step 1: Start
Step 2: Set first = 0, second = 1
Step 3: Read term of Fibonacci number, say n
Step 4: Set I = 2
Step 5: While(i <= n)
5.1: Set temp = first + second
5.2: Set first = second
5.3: Set second = temp
5.4: Increment I by 1 as i++
Step 6: Print temp as required Fibonacci number
Step 7: Stop.

Source Code:

```
#include<stdio.h>
fibonacci(n) {
        int i, first=0, second=1, temp=1;
        for (i=2; i<=n; i++) {
                temp=first+second;
                first=second;
                second=temp;
        } return temp; }
main() {
        int n;
        printf("Enter the value for n: ");
        scanf("%d", &n);
        printf("The %dth fibonacci number is %d.", n, fibonacci(n));
        return 0;
}
```

Output:



c. Sequential Search

Algorithm:

Step 1: start
Step 2: Read the search element
Step 3: Compare the search element with the first element in the list
Step 4: If both are matching, then display “Given element found!!!” and terminate the program
Step 5: If both are not matching, then compare the search element with the next element in the list
Step 6: Repeat steps 4 and 5 until the search element is compared with the last element in the list
Step 7: If the last element in the list is also doesn’t match, then display “Element not found!!!” and terminate the program
Step 8: Stop.

Source Code:

```
#include<stdio.h>
int sequentialSearch(int E[], int n, int key) {
        int i;
        for (i=0; i<n; i++) {
                if (E[i]==key)
        return i; }
  return -1; }

int main() {
        int i, n, E[n], key;
        printf("Enter total number of elements: ");
        scanf("%d", &n);
        printf("Enter %d elements: ", n);
        for(i=0; i<n; i++)
                scanf("%d", &E[i]);
        printf("Enter the key to be searched: ");
        scanf("%d", &key);
        printf("The searched element '%d' is found at index '%d'.",key,  sequentialSearch(E, n, key));
        return 0;
}
```

Output:

```
E:\Practical sem V\Design of /

Enter total number of elements: 10
Enter 10 elements: 1 5 6 8 9 4 7 2 3 0
Enter the key to be searched: 7
The searched element '7' is found at index '6'.
--------------------------------
```

d. Bubble Sort

Algorithm:

Step 1: Start
Step 2: For the first iteration, compare all the elements n. For the subsequent runs, compare (n-1) (n-2) and so on
Step 3: Compare each element with its right side neighbor
Step 4: Swap the smallest element to the left
Step 5: Keep repeating steps 1-3 until the wholw list is covered
Step 6: Stop.

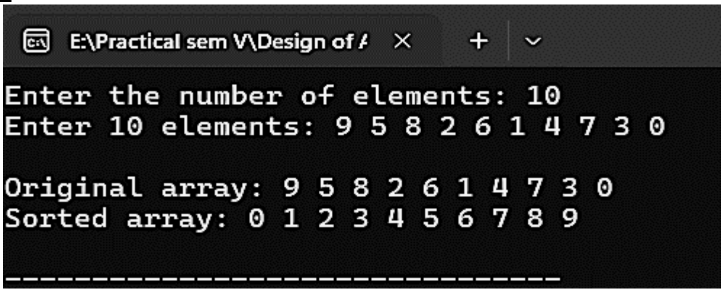
Source Code:

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

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

int main()
{
  int i, n;
  printf("Enter the number of elements: ");
  scanf("%d", &n);
  int arr[n];
  printf("Enter %d elements: ", n);
  for (i = 0; i < n; i++)
    scanf("%d", &arr[i]);
  printf("\nOriginal array: ");
  print_array(arr, n);
  bubble_sort(arr, n);
  printf("Sorted array: ");
  print_array(arr, n);
  return 0;
}
```

Output:



e. Selection Sort

Algorithm:

Step 1: Start
Step 2: Consider the first element to be sorted and the rest to be unsorted
Step 3: Assume the first element to be the smallest element
Step 4: Check if the first element is smaller than each of the other elements:
  4.1: If yes, do nothing
  4.2: If no, choose the other smaller element as minimum and repeat step 3
Step 5: After completion of one iteration through the list, swap the smallest element with the first element of the list
Step 6: Now consider the second element in the list to be the smallest and so on till all the elements in the list are covered
Step 7: Stop.

Source Code:

```
#include <stdio.h>
void selection_sort(int arr[], int n) {
  int i, j, min_idx, temp;
  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;
    }
    temp = arr[min_idx];
    arr[min_idx] = arr[i];
    arr[i] = temp;
  } }
void print_array(int arr[], int n) {
  int i;
  for (i = 0; i < n; i++)
    printf("%d ", arr[i]);
  printf("\n");
}
int main() {
  int i, n;
  printf("Enter the number of elements: ");
  scanf("%d", &n);
  int arr[n];
  printf("Enter %d elements: ", n);
  for (i = 0; i < n; i++)
    scanf("%d", &arr[i]);
  printf("\nOriginal array: ");
  print_array(arr, n);
  selection_sort(arr, n);
  printf("Sorted array: ");
  print_array(arr, n);
  return 0;
}
```

Output:

```
E:\Practical sem V\Design of /

Enter the number of elements: 10
Enter 10 elements: 9 0 8 1 7 3 4 2 5 6

Original array: 9 0 8 1 7 3 4 2 5 6
Sorted array: 0 1 2 3 4 5 6 7 8 9

--------------------------------
```

f. Insertion Sort

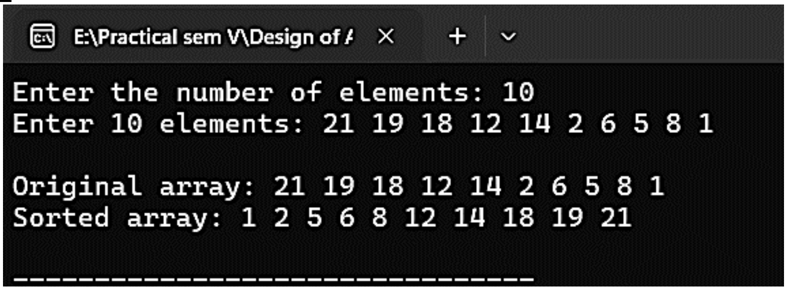
Algorithm:

Step 1: Start
Step 2: Consider the first element to be sorted and the rest to be unsorted
Step 3: Compare with the second element
  3.1: If the second element < the first element, insert the element in the correct position of the sorted portion
  3.2: Else, leave it as it's
Step 4: Repeat 2 and 3 until all elements are sorted
Step 5: Stop.

Source Code:

```c
#include <stdio.h>
void selection_sort(int arr[], int n) {
  int i, j, min_idx, temp;
  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;
    }
    temp = arr[min_idx];
    arr[min_idx] = arr[i];
    arr[i] = temp;
  } }
void print_array(int arr[], int n) {
  int i;
  for (i = 0; i < n; i++)
    printf("%d ", arr[i]);
  printf("\n");
}
int main() {
  int i, n;
  printf("Enter the number of elements: ");
  scanf("%d", &n);
  int arr[n];
  printf("Enter %d elements: ", n);
  for (i = 0; i < n; i++)
    scanf("%d", &arr[i]);
  printf("\nOriginal array: ");
  print_array(arr, n);
  selection_sort(arr, n);
  printf("Sorted array: ");
  print_array(arr, n);
  return 0;
}
```

Output:



g. Min-Max Algorithm

Algorithm:

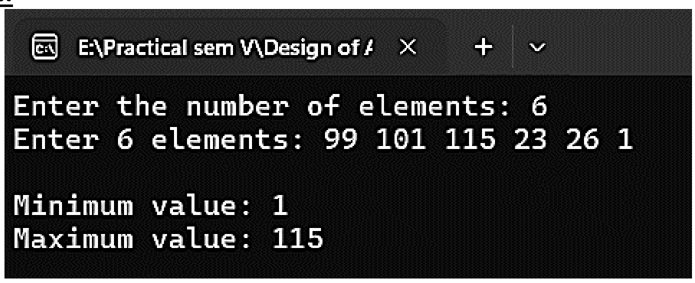
```
Max-Min-Element(numbers[])
Max = numbers[0]
Min = numbers[0]
For i = 1 to n do
        If numbers[i] > max then
                Max = numbers[i]
        If numbers[i] < min then
                Min = numbers[i]
Return(max, min)
```

Source Code:

```
#include <stdio.h>
void find_min_max(int arr[], int n, int *min, int *max) {
        int i;
  *min = *max = arr[0];
  for (i = 1; i < n; i++) {
    if (arr[i] < *min) {
      *min = arr[i];
    }
        else if (arr[i] > *max) {
      *max = arr[i];
    } } }

int main() {
  int i, n;
  printf("Enter the number of elements: ");
  scanf("%d", &n);
  int arr[n];
  printf("Enter %d elements: ", n);
  for (i = 0; i < n; i++)
    scanf("%d", &arr[i]);
  int min, max;
  find_min_max(arr, n, &min, &max);
  printf("\nMinimum value: %d\n", min);
  printf("Maximum value: %d\n", max);
  return 0;
}
```

Output:



	GCD	Fibonacci Numbers	Linear Search	Bubble Sort	Selection Sort	Insertion Sort	Min-Max Algorithm
Time Complexity	**O**(n)	**O**(n)	**O**(n)	**O**(n^2)	**O**(n^2)	**O**(n^2)	**O**(n)
Space Complexity	**O**(1)	**O**(1)	**O**(n)	**O**(1)	**O**(1)	**O**(1)	**O**(1)

2. Write algorithm and program to search an item in ordered list using binary search as divide and conquer Approach.

Algorithm:

Step 1: start
Step 2: Read the search element from the user
Step 3: Find the middle element in the sorted list
Step 4: Compare the search element with the middle element in the sorted list
Step 5: If both are matching, then display "Given element found!!!" and terminate the program
Step 6: If both are not matching, then check whether the search element is smaller or larger than the middle element
Step 7: If the search element is smaller than the middle element, then repeat 3, 4, 5 and 6 for the left sub-list of the middle element
Step 8: If the search element is larger than the middle element, then repeat 3, 4, 5 and 6 for the right sub-list of the middle element
Step 9: Repeat the same process until we find the search element in the list or until sub list contains only one element
Step 10: If that element also doesn't match with the search element, then display "Element not found!!!" and terminate the program
Step 11: Stop.

Source Code:

```
#include <stdio.h>
int binary_search(int arr[], int target, int low, int high) {
   if (low > high)
      return -1; // not found
   int mid = (low + high) / 2;
   if (arr[mid] == target) {
      return mid; // found
   }
         else if (arr[mid] < target) {
      return binary_search(arr, target, mid + 1, high);
   }
         else {
      return binary_search(arr, target, low, mid - 1);
} }
int main() {
   int n;
   printf("Enter the number of elements: ");
   scanf("%d", &n);
   int arr[n];
   printf("Enter %d elements in sorted order: ", n);
   for (int i = 0; i < n; i++)
      scanf("%d", &arr[i]);
   int target;
   printf("Enter the target element: ");
   scanf("%d", &target);
   int result = binary_search(arr, target, 0, n - 1);
   if (result != -1) {
      printf("The searched element '%d' is found at index '%d'.", target, result);
```

```
  }
          else {
    printf("Element not found!!!\n");
  }
  return 0;
}
```

Output:

```
E:\Practical sem V\Design of A
Enter the number of elements: 10
Enter 10 elements in sorted order: 21 23 25 29 35 46 58 68 78 79
Enter the target element: 78
The searched element '78' is found at index '8'.
--------------------------------
```

3. Write algorithm and program to find the maximum and minimum element in a list of numbers using divide and conquer approach.

Algorithm:

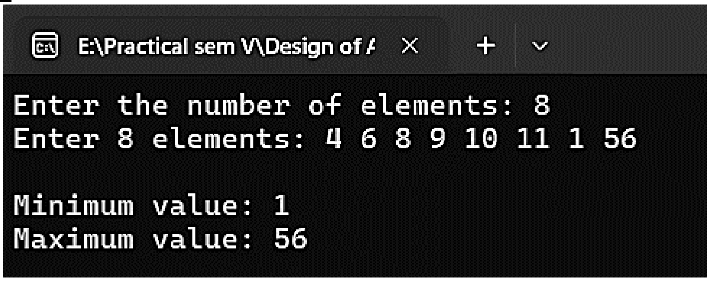
```
Max - Min(x, y)
if y – x ≤ 1 then
        return (max(numbers[x],numbers[y]),min((numbers[x],numbers[y]))
else
        (max1, min1):= maxmin(x, ⌊((x + y)/2)⌋)
        (max2, min2):= maxmin(⌊((x + y)/2) + 1)⌋,y)
return (max(max1, max2), min(min1, min2))
```

Source Code:

```c
#include <stdio.h>
void find_min_max(int arr[], int low, int high, int *min, int *max) {
  if (low == high) {
    *min = *max = arr[low];
  }
      else if (low + 1 == high) {
    if (arr[low] < arr[high]) {
      *min = arr[low];
      *max = arr[high];
    }
      else {
      *min = arr[high];
      *max = arr[low];
    }
  }
      else {
    int mid = (low + high) / 2;
    int min1, max1, min2, max2;
    find_min_max(arr, low, mid, &min1, &max1);
    find_min_max(arr, mid + 1, high, &min2, &max2);
    *min = (min1 < min2) ? min1 : min2;
    *max = (max1 > max2) ? max1 : max2;
  }
}
int main() {
  int i, n;
  printf("Enter the number of elements: ");
  scanf("%d", &n);
  int arr[n];
  printf("Enter %d elements: ", n);
```

```
    for (i = 0; i < n; i++)
        scanf("%d", &arr[i]);
    int min, max;
    find_min_max(arr, 0, n - 1, &min, &max);
    printf("\nMinimum value: %d\n", min);
    printf("Maximum value: %d\n", max);
    return 0;
}
```

Output:



4. Write algorithms and program using divide and conquer approach for following sorting algorithms

a. Heap Sort

Algorithms:

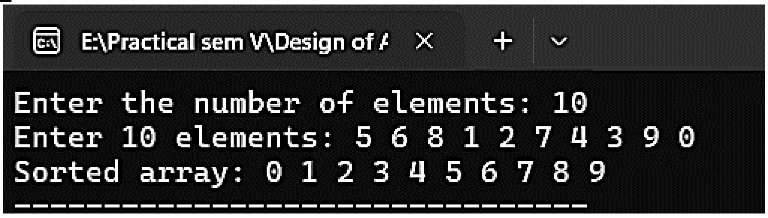
```
HeapSort(A){
        BuildHeap(A);
    N = length[A];
    For(i = n; i >= 2; i--){
        Swap(A[1], A[n]);
                n = n-1;
                Heapify(A, 1);
        }
}
```

Source Code:

```
#include <stdio.h>
void heapify(int arr[], int n, int i) {
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;
    if (left < n && arr[left] > arr[largest])
        largest = left;
    if (right < n && arr[right] > arr[largest])
        largest = right;
    if (largest!= i) {
        int temp = arr[i];
        arr[i] = arr[largest];
        arr[largest] = temp;
        heapify(arr, n, largest);
    }
}
void heapSort(int arr[], int n) {
        int i;
    for (i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);
    for (i = n - 1; i >= 0; i--) {
        int temp = arr[0];
        arr[0] = arr[i];
        arr[i] = temp;
```

```
        heapify(arr, i, 0);
    } }
int main() {
    int i, n;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter %d elements: ", n);
    for (i = 0; i < n; i++)
        scanf("%d", &arr[i]);
    heapSort(arr, n);
    printf("Sorted array: ");
    for (i = 0; i < n; i++)
        printf("%d ", arr[i]);
    return 0;
}
```

Output:



b. Quick Sort

Algorithm:

Step 1: Start
Step 2: Choose a pivot
Step 3: Set a left pointer and right pointer
Step 4: Compare the left pointer element (lelement) with the pivot and the right pointer element (relement) with the pivot
Step 5: Check if lelement < pivot and relement > pivot:
    5.1: If yes, increment the left pointer and decrement the right pointer
    5.2: If not, swap the lelement and relement
Step 6: When left >= right, swap the pivot with either left or right pointer
Step 7: Repeat steps 2-5 on the left half and the right half of the list till the entire list is sorted
Step 8: Stop.

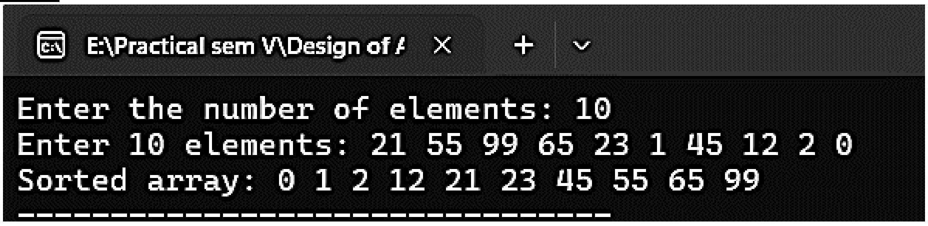
Source Code:

```
#include <stdio.h>
int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int j, i = low - 1;
    for (j = low; j < high; j++) {
        if (arr[j] < pivot) {
            i++;
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        } }
    int temp = arr[i + 1];
    arr[i + 1] = arr[high];
    arr[high] = temp;
    return i + 1;
}
void quickSort(int arr[], int low, int high) {
    if (low < high) {
```

```
      int pivot = partition(arr, low, high);
      quickSort(arr, low, pivot - 1);
      quickSort(arr, pivot + 1, high);
    } }

int main() {
    int i, n;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter %d elements: ", n);
    for (i = 0; i < n; i++)
        scanf("%d", &arr[i]);
    quickSort(arr, 0, n - 1);
    printf("Sorted array: ");
    for (i = 0; i < n; i++)
        printf("%d ", arr[i]);
    return 0;
}
```

Output:



c. Randomized quick Sort

Algorithm:

```
int partition_left(int arr[], int low, int high) {
        int pivot = arr[high];
        int i = low; // Place for swapping
        for (int j = low; j < high; j++) {
                if (arr[j] <= pivot) {
                swap(&arr[i], &arr[j]);
                i++;
                }
        }
        swap(&arr[i], &arr[high]);
        return i; }
int partition_right(int arr[], int low, int high) {
        int r = RandomNumber(low, high); // Randomly choose pivot index
        swap(&arr[r], &arr[high]);
        return partition_left(arr, low, high); }
void quicksort(int arr[], int low, int high) {
        if (low < high) {
        int pivot = partition_right(arr, low, high);
        quicksort(arr, low, pivot - 1);
        quicksort(arr, pivot + 1, high);
        }
}
```

Source Code:

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

```c
void swap(int* a, int* b) {
   int temp = *a;
   *a = *b;
   *b = temp;
}
// Function to partition the array
int partition(int arr[], int low, int high) {
   // Choose a random pivot index
   int pivot_index = low + rand() % (high - low + 1);
   int pivot = arr[pivot_index];
   // Move the pivot element to the end of the array
   swap(&arr[pivot_index], &arr[high]);
   int i = low - 1, j;
   for (j = low; j < high; j++) {
      if (arr[j] < pivot) {
         i++;
         swap(&arr[i], &arr[j]);
      }
   }
   // Move the pivot element to its final sorted position
   swap(&arr[i + 1], &arr[high]);
   return i + 1;
}
// Function to perform randomized quick sort
void randomized_quick_sort(int arr[], int low, int high) {
   if (low < high) {
      int pivot = partition(arr, low, high);
      // Recursively sort the two sub-arrays
      randomized_quick_sort(arr, low, pivot - 1);
      randomized_quick_sort(arr, pivot + 1, high);
   }
}
// Function to print the array
void print_array(int arr[], int size) {
            int i;
   for (i = 0; i < size; i++)
      printf("%d ", arr[i]);
   printf("\n");
}
int main() {
   int i, size;
   printf("Enter the size of the array: ");
   scanf("%d", &size);
   int* arr = (int*)malloc(size * sizeof(int));
   printf("Enter the elements of the array: ");
   for (i = 0; i < size; i++)
      scanf("%d", &arr[i]);
   // Seed the random number generator
   srand(time(NULL));
   printf("\nOriginal array: ");
   print_array(arr, size);
   randomized_quick_sort(arr, 0, size - 1);
   printf("Sorted array: ");
   print_array(arr, size);
   free(arr);
   return 0;
}
```

Output:

```
E:\Practical sem V\Design of /

Enter the size of the array: 10
Enter the elements of the array: 85 76 23 54 45 36 61 2 1 6

Original array: 85 76 23 54 45 36 61 2 1 6
Sorted array: 1 2 6 23 36 45 54 61 76 85
```

d. Merge Sort

Algorithm:

Step 1: Start
Step 2: Split the unsorted list into two groups recursively until there is one element per group
Step 3: Compare each of the elements and then sort and group them
Step 4: Repeat step 2 until the whole list is merged and sorted in the process
Step 5: Stop.

Source Code:

```
#include <stdio.h>
// Function to merge two sub-arrays
void merge(int arr[], int left, int mid, int right) {
   int i, j, k;
   int n1 = mid - left + 1;
   int n2 = right - mid;
   // Create temporary arrays
   int left_arr[n1], right_arr[n2];
   // Copy data to temporary arrays
   for (i = 0; i < n1; i++)
      left_arr[i] = arr[left + i];
   for (j = 0; j < n2; j++)
      right_arr[j] = arr[mid + 1 + j];
   // Merge the temporary arrays
   i = 0;
   j = 0;
   k = left;
   while (i < n1 && j < n2) {
      if (left_arr[i] <= right_arr[j]) {
         arr[k] = left_arr[i];
         i++;
      }
else{
         arr[k] = right_arr[j];
         j++;
      }
      k++;
   }
   // Copy the remaining elements of left_arr[], if there are any
   while (i < n1) {
      arr[k] = left_arr[i];
      i++;
      k++;
   }
   // Copy the remaining elements of right_arr[], if there are any
   while (j < n2) {
      arr[k] = right_arr[j];
      j++;
      k++;
   }
```

```
}
// Function to perform merge sort
void merge_sort(int arr[], int left, int right) {
  if (left < right) {
    int mid = left + (right - left) / 2;
    // Recursively sort the two halves
    merge_sort(arr, left, mid);
    merge_sort(arr, mid + 1, right);
    // Merge the two sorted halves
    merge(arr, left, mid, right);
  }
}
// Function to print the array
void print_array(int arr[], int size){
       int i;
  for (i = 0; i < size; i++)
    printf("%d ", arr[i]);
  printf("\n");
}
int main() {
  int i, size;
  printf("Enter the size of the array: ");
  scanf("%d", &size);
  int arr[size];
  printf("Enter the elements of the array: ");
  for (i = 0; i < size; i++)
    scanf("%d", &arr[i]);
  printf("\nOriginal array: ");
  print_array(arr, size);
  merge_sort(arr, 0, size - 1);
  printf("Sorted array: ");
  print_array(arr, size);
  return 0;
}
```

Output:

```
E:\Practical sem V\Design of /

Enter the size of the array: 15
Enter the elements of the array: 212 11 5 63 512 983 654 223 75 15 65 0 9 1452 1212

Original array: 212 11 5 63 512 983 654 223 75 15 65 0 9 1452 1212
Sorted array: 0 5 9 11 15 63 65 75 212 223 512 654 983 1212 1452
```

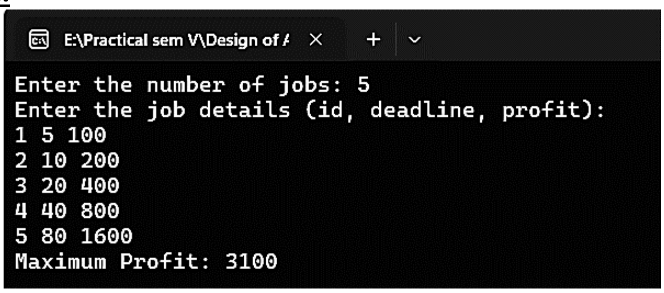
5. Write a program to implement problem Job Sequencing with deadline using greedy algorithm

Source Code:

```
#include <stdio.h>
// Structure to represent a job
typedef struct {
  int id;
  int deadline;
  int profit;
}Job;
// Function to compare two jobs based on their profits
int compare(const void *a, const void *b) {
  Job *job1 = (Job *)a;
  Job *job2 = (Job *)b;
  return job2->profit - job1->profit;
```

```
}
// Function to find the maximum profit
void jobSequencing(Job jobs[], int n) {
        int i, j;
  // Sort the jobs based on their profits
  qsort(jobs, n, sizeof(Job), compare);
  // Create a boolean array to track the deadlines
  int deadlines[10] = {0};
  // Initialize the maximum profit
  int maxProfit = 0;
  // Iterate through the sorted jobs
  for (i = 0; i < n; i++) {
    // Find a deadline that can accommodate the current job
    for (j = jobs[i].deadline; j >= 1; j--) {
      if (!deadlines[j]) {
        deadlines[j] = 1;
        maxProfit += jobs[i].profit;
        break;
      } } }
  printf("Maximum Profit: %d\n", maxProfit);
}
int main() {
  int i, n;
  printf("Enter the number of jobs: ");
  scanf("%d", &n);
  Job jobs[n];
  printf("Enter the job details (id, deadline, profit):\n");
  for (i = 0; i < n; i++)
    scanf("%d %d %d", &jobs[i].id, &jobs[i].deadline, &jobs[i].profit);
  jobSequencing(jobs, n);
  return 0;
}
```

Output:



6. Implement the Floyd- Warshall's algorithm to compute all pair shortest path problem using dynamic programming approach.

Source Code:

```
#include <iostream>
#include <climits>
int main() {
  int V;
  std::cout << "Enter the number of vertices: ";
  std::cin >> V;
  int graph[V][V];
  std::cout << "Enter the adjacency matrix:" << std::endl;
  for (int i = 0; i < V; i++) {
    for (int j = 0; j < V; j++) {
      std::cin >> graph[i][j];
```

```
          if (graph[i][j] == 0 && i != j) {
            graph[i][j] = INT_MAX;
          } } }
    std::cout << "\nOriginal graph:" << std::endl;
    for (int i = 0; i < V; i++) {
      for (int j = 0; j < V; j++) {
        std::cout << graph[i][j] << " ";
      }
      std::cout << std::endl;
    }
    for (int k = 0; k < V; k++) {
      for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
          if (graph[i][k] + graph[k][j] < graph[i][j])
            graph[i][j] = graph[i][k] + graph[k][j];
        } } }
    std::cout << "\nShortest distances between all pairs of vertices:" << std::endl;
    for (int i = 0; i < V; i++) {
      for (int j = 0; j < V; j++) {
        std::cout << graph[i][j] << " ";
      }
      std::cout << std::endl;
    }
    return 0;
}
```

Output:

```
D:\Practical sem V\Design of

Enter the number of vertices: 3
Enter the adjacency matrix:
0 4 11
6 0 2
3 10000000000000000000 0

Original graph:
0 4 11
6 0 2
3 2147483647 0

Shortest distances between all pairs of vertices:
0 4 6
5 0 2
3 7 0
```

7. Write a program to implement Matrix Chain Multiplication using dynamic programming algorithm.

Source Code:

```
#include <stdio.h>
#include <limits.h>
int MatrixChainOrder(int p[], int n) {
  int m[100][100];
  int i, j, k, L, q;
  for (i = 1; i <= n; i++)
    m[i][i] = 0;
        for (L = 2; L < n; L++) {
    for (i = 1; i < n - L + 1; i++) {
      j = i + L - 1;
      m[i][j] = INT_MAX;
      for (k = i; k <= j - 1; k++) {
```

```
            q = m[i][k] + m[k + 1][j] + p[i - 1] * p[k] * p[j];
            if (q < m[i][j])
                m[i][j] = q;
        } } }
    return m[1][n - 1];
}
int main() {
    int i, n;
    printf("Enter the number of matrices: ");
    scanf("%d", &n);
    int p[n + 1];
    printf("Enter the dimensions of the matrices:\n");
    for (i = 0; i <= n; i++) {
        scanf("%d", &p[i]);
    }
    printf("\nThe optimal multiplication cost is %d.", MatrixChainOrder(p, n + 1));
    return 0;
}
```

Output: