Q1. Implement Euclid's algorithm to find GCD (15265, 15)and calculate the number of times mod and assignment operations will be required.

```
#include <stdio.h>
#include<conio.h>
int gcd(int a, int b, int* modCount, int* assignCount) {
   while (b != 0) {
      (*assignCount)++;
      int temp = b;
      b = a % b;
      (*modCount)++;
      a = temp;
   }
   return a;
}
void main() {
   int num1 = 15265;
   int num2 = 15;
   int modCount = 0;
   int assignCount = 0;
   int result = gcd(num1, num2, &modCount, &assignCount);
    printf("GCD of %d and %d is %d\n", num1, num2, result);
   printf("Number of mod operations: %d\n", modCount);
   printf("Number of assignment operations: %d\n", assignCount);

   getch();
}
```

Q2(i). Implement Horner's rule for evaluating polynomial P(x)= $6x$ 6+$5x$ 5 + $4x$ 4 -$3x$ 3 + $2x$ 2 + $8x$ − 7 at x = 3. Calculate how many times (i) multiplications and addition operations will take (ii) how many times the loop will iterate

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

```
double horner(double coeff[], int degree, double x, int* multCount, int*
addCount, int* loopCount) {
    double result = coeff[0];
    int i;
    for (int i = 1; i <= degree; i++) {
        result = result * x + coeff[i];
        (*multCount)++;
        (*addCount)++;
        (*loopCount)++;
    }
    return result;
}
void main() {
    double coeff[] = {66, 55, 44, -33, 22, 8, -7};
    int degree = 6;  // Highest power of x
    double x = 3;
    int multCount = 0, addCount = 0, loopCount = 0;


    double result = horner(coeff, degree, x, &multCount, &addCount,
&loopCount);


    printf("P(x) = 66 + 55x + 44x^2 - 33x^3 + 22x^4 + 8x^5 - 7 evaluated at x =
3 is %lf\n", result);
    printf("Number of multiplications: %d\n", multCount);
    printf("Number of additions: %d\n", addCount);
    printf("Number of loop iterations: %d\n", loopCount);


  getch();
}
```

Q3.Implement multiplication of two matrices A[4,4] and B[4,4] and calculate (i) how many times the innermost and the outermost loops will run (ii) total number of multiplications and additions in computing the multiplication

```
#include <stdio.h>
#include<conio.h>
void multiplyMatrices(int a[4][4], int b[4][4], int result[4][4], int* multCount, int* addCount, int* innerLoopCount, int* outerLoopCount) {
int i,j,k;
for (i = 0; i < 4; i++) {
        (*outerLoopCount)++;
        for (j = 0; j < 4; j++) {
            result[i][j] = 0;
            for (k = 0; k < 4; k++) {
                (*innerLoopCount)++;
                result[i][j] += a[i][k] * b[k][j];
                (*multCount)++;
                (*addCount)++;
            }
        }
    }
}
int i,j,k;
void main() {
    int a[4][4] = {
        {1, 2, 3, 4},
        {5, 6, 7, 8},
        {9, 10, 11, 12},
        {13, 14, 15, 16}
    };
    int b[4][4] = {
        {16, 15, 14, 13},
        {12, 11, 10, 9},
        {8, 7, 6, 5},
        {4, 3, 2, 1}
```

```
    };
    int result[4][4];
    int multCount = 0, addCount = 0, innerLoopCount = 0, outerLoopCount = 0;

    multiplyMatrices(a, b, result, &multCount, &addCount, &innerLoopCount,
&outerLoopCount);

    printf("Resultant Matrix:\n");
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            printf("%d ", result[i][j]);
        }
        printf("\n");
    }

    printf("\nNumber of outer loop iterations: %d\n", outerLoopCount);
    printf("Number of inner loop iterations: %d\n", innerLoopCount);
    printf("Total number of multiplications: %d\n", multCount);
    printf("Total number of additions: %d\n", addCount);

   getch();
}
```

Q4. Implement Bubble Sort algorithm for the following list of numbers: 55 25 15 40 60 35 17 65 75 10 Calculate (i) a number of exchange operations (ii) a number of times comparison operations (iii) a number of times the inner and outer loops will iterate?

```
#include <stdio.h>
#include<conio.h>
void bubbleSort(int arr[], int n, int* exchangeCount, int* comparisonCount, int*
innerLoopCount, int* outerLoopCount) {
```

```
    int temp,i,j;
    for (i = 0; i < n-1; i++) {
        (*outerLoopCount)++;
        for (j = 0; j < n-i-1; j++) {
            (*innerLoopCount)++;
            (*comparisonCount)++;
            if (arr[j] > arr[j+1]) {
                temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
                (*exchangeCount)++;
            }
        }
    }
}
int i;
int main() {
    int arr[] = {55, 25, 15, 40, 60, 35, 17, 65, 75, 10};
    int n = sizeof(arr)/sizeof(arr[0]);
    int exchangeCount = 0, comparisonCount = 0, innerLoopCount = 0,
outerLoopCount = 0;

    bubbleSort(arr, n, &exchangeCount, &comparisonCount, &innerLoopCount,
&outerLoopCount);

    printf("Sorted array: \n");
    for (i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }

    printf("\n\nNumber of exchanges: %d\n", exchangeCount);
    printf("Number of comparisons: %d\n", comparisonCount);
```

```
    printf("Number of inner loop iterations: %d\n", innerLoopCount);
    printf("Number of outer loop iterations: %d\n", outerLoopCount);

    getch();
}
```

5)Implement Fractional Knapsack algorithm and find out optimal result for the following problem instances: (P1, P2, P3, P4, P5) = (20, 30,40, 32, 55) (W1, W2, W3, W4, W5) = (5, 8, 10 12, 15) Maximum Knapsack Capacity = 20 using c program

```
#include <stdio.h>
#include<conio.h>
struct Item {
    int weight;
    int profit;
    double ratio;
};

void calculateRatio(struct Item arr[], int n) {
int i;
    for (i = 0; i < n; i++) {
        arr[i].ratio = (double) arr[i].profit / arr[i].weight;
    }
}

void sortByRatio(struct Item arr[], int n) {
int i,j;
    for (i = 0; i < n-1; i++) {
        for (j = i+1; j < n; j++) {
            if (arr[i].ratio < arr[j].ratio) {
                struct Item temp = arr[i];
```

```
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
  }
}

double fractionalKnapsack(struct Item arr[], int n, int capacity) {
  int currentWeight = 0;
  double totalProfit = 0.0;
  int i;
  calculateRatio(arr, n);
  sortByRatio(arr, n);



  for (i = 0; i < n; i++) {
    if (currentWeight + arr[i].weight <= capacity) {
      currentWeight += arr[i].weight;
      totalProfit += arr[i].profit;
    } else {
      int remainingWeight = capacity - currentWeight;
      totalProfit += arr[i].ratio * remainingWeight;
      break;
    }
  }

  return totalProfit;
}

void main() {
```

```
    struct Item items[] = {
        {5, 20, 0},
        {8, 30, 0},
        {10, 40, 0},
        {12, 32, 0},
        {15, 55, 0}
    };
    int n = sizeof(items) / sizeof(items[0]);
    int capacity = 20;

    double maxProfit = fractionalKnapsack(items, n, capacity);
    printf("Maximum profit in the knapsack of capacity %d is %lf\n", capacity, maxProfit);

    getch();
}
```

6) Implement Fractional Knapsack algorithm and find out optimal result for the following problem instances: Q1 (P1, P2, P3, P4, P5, P6, P7) = (15, 5, 20, 8, 7, 20, 6) (W1, W2, W3, W4,W5,W6,W7) = (3, 4, 6, 8, 2, 2, 3) Maximum Knapsack Capacity = 18

```
#include <stdio.h>
#include<conio.h>
struct Item {
    int weight;
    int profit;
    double ratio;
};

void calculateRatio(struct Item arr[], int n) {
int i;
```

```
 for (i = 0; i < n; i++) {
        arr[i].ratio = (double) arr[i].profit / arr[i].weight;
    }
}

void sortByRatio(struct Item arr[], int n) {
int i,j;
    for (i = 0; i < n-1; i++) {
        for (j = i+1; j < n; j++) {
            if (arr[i].ratio < arr[j].ratio) {
                struct Item temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
    }
}

double fractionalKnapsack(struct Item arr[], int n, int capacity) {
    int currentWeight = 0;
    double totalProfit = 0.0;
int i;
    calculateRatio(arr, n);
    sortByRatio(arr, n);



    for (i = 0; i < n; i++) {
        if (currentWeight + arr[i].weight <= capacity) {
            currentWeight += arr[i].weight;
            totalProfit += arr[i].profit;
```

```
        } else {
            int remainingWeight = capacity - currentWeight;
            totalProfit += arr[i].ratio * remainingWeight;
            break;
        }
    }

    return totalProfit;
}

void main() {
    struct Item items[] = {
        {3, 15, 0},
        {4, 5, 0},
        {6, 20, 0},
        {8, 8, 0},
        {2, 7, 0},
        {2, 20, 0},
        {3, 6, 0}
    };
    int n = sizeof(items) / sizeof(items[0]);
    int capacity = 18;

    double maxProfit = fractionalKnapsack(items, n, capacity);
    printf("Maximum profit in the knapsack of capacity %d is %lf\n", capacity,
maxProfit);

    getch();
}
```

7)Implement the task scheduling algorithm on your system to minimize the total amount of time spent in the system for the following problem

: Job Service Time

1 5

2 10

3 7

4 8 using C program

```
#include <stdio.h>
#include<conio.h>
// Structure to hold the details of a job
struct Job {
    int id;
    int serviceTime;
    int waitingTime;
    int turnaroundTime;
};

// Function to sort jobs by service time (for SJF scheduling)
void sortJobsByServiceTime(struct Job jobs[], int n) {
    int i,j;
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (jobs[i].serviceTime > jobs[j].serviceTime) {
                struct Job temp = jobs[i];
                jobs[i] = jobs[j];
                jobs[j] = temp;
            }
        }
    }
}
```

```
// Function to calculate waiting time and turnaround time
void calculateTimes(struct Job jobs[], int n) {
    int totalWaitingTime = 0, totalTurnaroundTime = 0,i;
    jobs[0].waitingTime = 0;  // First job has zero waiting time
    jobs[0].turnaroundTime = jobs[0].serviceTime;

    for (i = 1; i < n; i++) {
        jobs[i].waitingTime = jobs[i - 1].waitingTime + jobs[i - 1].serviceTime;
        jobs[i].turnaroundTime = jobs[i].waitingTime + jobs[i].serviceTime;

        totalWaitingTime += jobs[i].waitingTime;
        totalTurnaroundTime += jobs[i].turnaroundTime;
    }

    printf("Job\tService Time\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\n", jobs[i].id, jobs[i].serviceTime,
jobs[i].waitingTime, jobs[i].turnaroundTime);
    }

    printf("\nAverage Waiting Time: %.2f\n", (float)totalWaitingTime / n);
    printf("Average Turnaround Time: %.2f\n", (float)totalTurnaroundTime / n);
}

void main() {
    int n = 4;
    struct Job jobs[] = {
        {1, 5, 0, 0},
        {2, 10, 0, 0},
        {3, 7, 0, 0},
        {4, 8, 0, 0}
```

```
    };

    // Sort jobs by service time for SJF scheduling
    sortJobsByServiceTime(jobs, n);

    // Calculate waiting time and turnaround time
    calculateTimes(jobs, n);

    getch();
}
```

8) Implement a recursive binary search algorithm on your system to search for a number 100 in the following array of integers. Show the processes step by step:
10 35 40 45 50 55 60 65 70 100

Draw recursive calls to be made in this problem

```
#include <stdio.h>
#include<conio.h>
// Recursive binary search function
int binarySearch(int arr[], int low, int high, int target) {
   int i;
    if (low > high) {
        return -1;  // Base case: number not found
    }

    int mid = (low + high) / 2;
    printf("Searching in range [%d, %d] with mid index %d (value = %d)\n",
low, high, mid, arr[mid]);

    if (arr[mid] == target) {
        return mid;  // Target found at mid
```

```
    } else if (arr[mid] < target) {
        return binarySearch(arr, mid + 1, high, target);  // Search in the right half
    } else {
        return binarySearch(arr, low, mid - 1, target);  // Search in the left half
    }
}

void main() {
    int arr[] = {10, 35, 40, 45, 50, 55, 60, 65, 70, 100};
    int size = sizeof(arr) / sizeof(arr[0]);
    int target = 100;
   int result;
    clrscr();
    printf("Searching for %d in the array...\n", target);
    result = binarySearch(arr, 0, size - 1, target);

    if (result != -1) {
        printf("Number %d found at index %d.\n", target, result);
    } else {
        printf("Number %d not found in the array.\n", target);
    }

    getch();
}
```

9) Implement Quick Sort's algorithm on your machine to do sorting of the following list of elements 12 20 22 16 25 18 8 10 6 15 Show step by step processes. using C program

```
#include <stdio.h>
#include<conio.h>
// Function to swap two elements
```

```
void swap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

// Partition function to place pivot element at correct position
int partition(int array[], int low, int high) {
  int j;
    int pivot = array[high];  // Choose the last element as pivot
    int i = low - 1;  // Index of smaller element

    for (j = low; j < high; j++) {
        if (array[j] < pivot) {
            i++;
            swap(&array[i], &array[j]);
        }
    }
    swap(&array[i + 1], &array[high]);
    return (i + 1);
}

// Quick Sort function
void quickSort(int array[], int low, int high) {
    if (low < high) {
        int pi = partition(array, low, high);  // pi is the partition index

        // Recursively sort elements before and after partition
        quickSort(array, low, pi - 1);
        quickSort(array, pi + 1, high);
    }
```

```
}

// Function to print the array
void printArray(int array[], int size) {
 int i;
    for (i = 0; i < size; i++)
        printf("%d ", array[i]);
    printf("\n");
}

// Main function to test Quick Sort
void main() {
    int array[] = {12, 20, 22, 16, 25, 18, 8, 10, 6, 15};
    int n = sizeof(array) / sizeof(array[0]);

    printf("Original array: ");
    printArray(array, n);

    quickSort(array, 0, n - 1);

    printf("Sorted array: ");
    printArray(array, n);

    getch();
}
```