VIT
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Lab Assignment - I, Winter Semester 2023-24

**Course Code** : BCSE204P **Slot** : L9 + L10

**Course Name:** Design and Analysis of Algorithm **Marks** : 10

1. **To find the kth smallest and kth largest element in a list of elements without sorting the elements.**

**CODE:**

```
#include <stdio.h>

void findKthSmallestAndLargest(int arr[], int n, int k) {
   // Initialize variables for kth smallest and largest
   int kthSmallest, kthLargest;

   // Iterate through the array to find kth smallest
   for (int i = 0; i < n; i++) {
      int smallerCount = 0;
      for (int j = 0; j < n; j++) {
         if (arr[j] < arr[i]) {
            smallerCount++;
         }
      }
      if (smallerCount == k - 1) {
         kthSmallest = arr[i];
         break;
      }
   }

   // Iterate through the array to find kth largest
   for (int i = 0; i < n; i++) {
      int largerCount = 0;
      for (int j = 0; j < n; j++) {
         if (arr[j] > arr[i]) {
            largerCount++;
```

```
            }
        }
        if (largerCount == k - 1) {
            kthLargest = arr[i];
            break;
        }
    }

    // Print the results
    printf("The %dth smallest element is: %d\n", k, kthSmallest);
    printf("The %dth largest element is: %d\n", k, kthLargest);
}

int main() {
    // Input the size of the array
    int n;
    printf("Enter the size of the array: ");
    scanf("%d", &n);

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

    // Input the value of k
    int k;
    printf("Enter the value of k: ");
    scanf("%d", &k);

    // Find and print the kth smallest and kth largest element
    findKthSmallestAndLargest(arr, n, k);

    return 0;
}
```



**CODE SCREENSHOT:**

main.c

Save Run

```c
//21BCE0169

#include <stdio.h>

void findKthSmallestAndLargest(int arr[], int n, int k) {
    // Initialize variables for kth smallest and largest
    int kthSmallest, kthLargest;

    // Iterate through the array to find kth smallest
    for (int i = 0; i < n; i++) {
        int smallerCount = 0;
        for (int j = 0; j < n; j++) {
            if (arr[j] < arr[i]) {
                smallerCount++;
            }
        }
        if (smallerCount == k - 1) {
            kthSmallest = arr[i];
            break;
        }
    }

    // Iterate through the array to find kth largest
    for (int i = 0; i < n; i++) {
        int largerCount = 0;
        for (int j = 0; j < n; j++) {
            if (arr[j] > arr[i]) {
                largerCount++;
```

Output

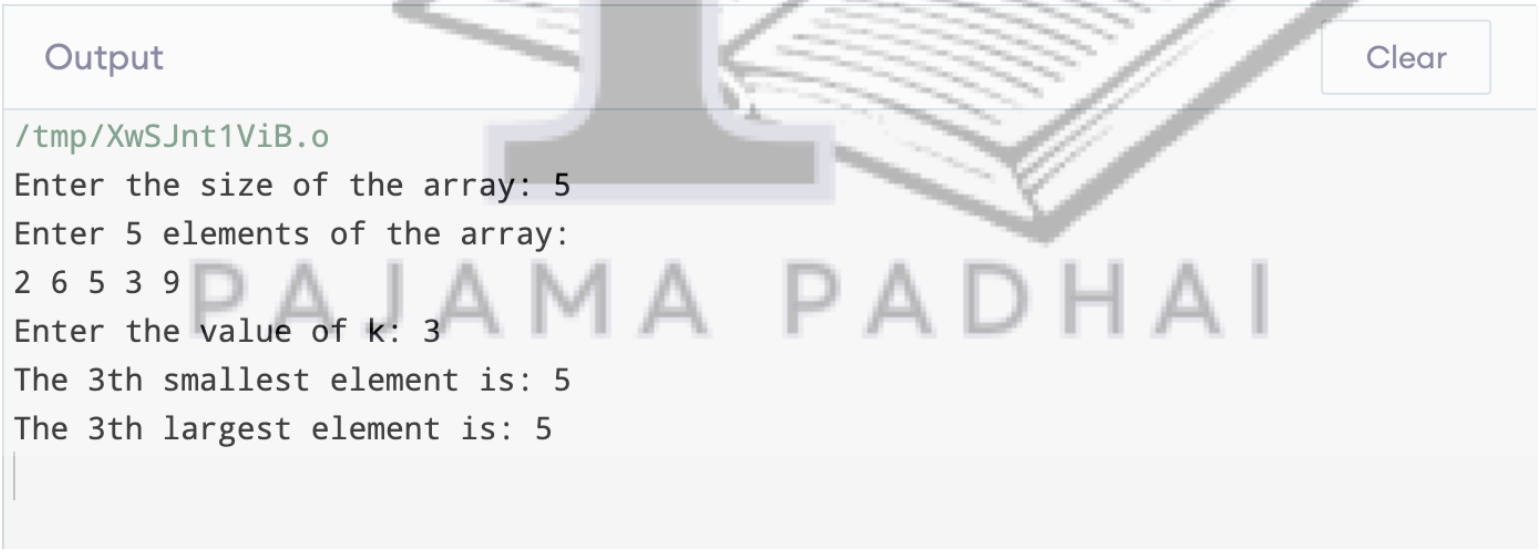
Clear

```
/tmp/XwSJnt1ViB.o
Enter the size of the array: 5
Enter 5 elements of the array:
2 6 5 3 9
Enter the value of k: 3
The 3th smallest element is: 5
The 3th largest element is: 5
```

**OUTPUT:**

Enter the size of the array: 5
Enter 5 elements of the array:
2 6 5 3 9
Enter the value of k: 3
The 3th smallest element is: 5
The 3th largest element is: 5

**OUTPUT SCREENSHOT:**



2. **To search for an element using recursive linear or binary search based on the option given by the user. The number of comparisons taken to find the element also must be found.**

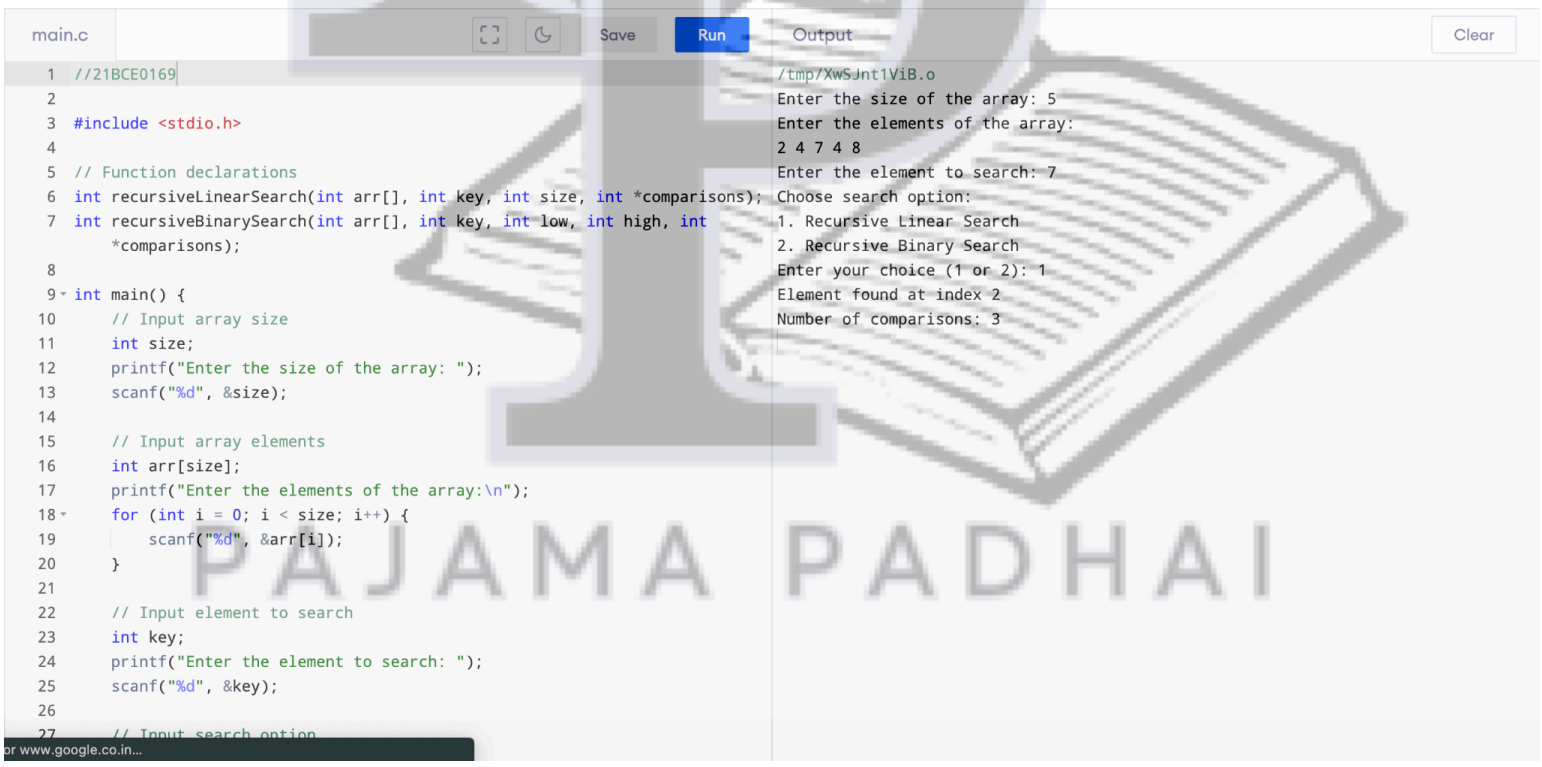
**CODE:**

```
#include <stdio.h>

// Function declarations
int recursiveLinearSearch(int arr[], int key, int size, int *comparisons);
int recursiveBinarySearch(int arr[], int key, int low, int high, int *comparisons);

int main() {
    // Input array size
    int size;
    printf("Enter the size of the array: ");
    scanf("%d", &size);

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

    // Input element to search
    int key;
    printf("Enter the element to search: ");
    scanf("%d", &key);

    // Input search option
    int option;
    printf("Choose search option:\n");
    printf("1. Recursive Linear Search\n");
    printf("2. Recursive Binary Search\n");
    printf("Enter your choice (1 or 2): ");
    scanf("%d", &option);

    // Variables to store the number of comparisons
    int comparisons = 0;

    // Perform search based on user's choice
    int index;
    switch (option) {
```

```
        case 1:
            index = recursiveLinearSearch(arr, key, size, &comparisons);
            break;
        case 2:
            index = recursiveBinarySearch(arr, key, 0, size - 1, &comparisons);
            break;
        default:
            printf("Invalid choice\n");
            return 1; // Exit with an error code
    }

    // Output the results
    if (index != -1) {
        printf("Element found at index %d\n", index);
        printf("Number of comparisons: %d\n", comparisons);
    } else {
        printf("Element not found\n");
        printf("Number of comparisons: %d\n", comparisons);
    }

    return 0;
}

// Function for recursive linear search
int recursiveLinearSearch(int arr[], int key, int size, int *comparisons) {
    if (size == 0) {
        // Base case: element not found
        return -1;
    }

    if (arr[size - 1] == key) {
        // Base case: element found
        (*comparisons)++;
        return size - 1;
    }

    // Recursive case
    (*comparisons)++;
    return recursiveLinearSearch(arr, key, size - 1, comparisons);
}

// Function for recursive binary search
int recursiveBinarySearch(int arr[], int key, int low, int high, int *comparisons) {
```

```
    if (low > high) {
        // Base case: element not found
        return -1;
    }

    int mid = (low + high) / 2;

    if (arr[mid] == key) {
        // Base case: element found
        (*comparisons)++;
        return mid;
    } else if (arr[mid] > key) {
        // Recursive case: search in the left half
        (*comparisons)++;
        return recursiveBinarySearch(arr, key, low, mid - 1, comparisons);
    } else {
        // Recursive case: search in the right half
        (*comparisons)++;
        return recursiveBinarySearch(arr, key, mid + 1, high, comparisons);
    }
}
```

**CODE SCREENSHOT:**



**OUTPUT:**

**Recursive Linear Search:**

Enter the size of the array: 5
Enter the elements of the array:
2 4 7 4 8
Enter the element to search: 7
Choose search option:
1. Recursive Linear Search
2. Recursive Binary Search
Enter your choice (1 or 2): 1
Element found at index 2
Number of comparisons: 3

**Recursive Binary Search:**

Enter the size of the array: 5
Enter the elements of the array:
4 7 2 4 9
Enter the element to search: 4
Choose search option:
1. Recursive Linear Search
2. Recursive Binary Search
Enter your choice (1 or 2): 2
Element found at index 3
Number of comparisons: 2

**OUTPUT SCREENSHOT:**

Recursive Linear Search:

Output Clear

```
/tmp/XwSJnt1ViB.o
Enter the size of the array: 5
Enter the elements of the array:
2 4 7 4 8
Enter the element to search: 7
Choose search option:
1. Recursive Linear Search
2. Recursive Binary Search
Enter your choice (1 or 2): 1
Element found at index 2
Number of comparisons: 3
```

Recursive Binary Search:

Output Clear

```
/tmp/XwSJnt1ViB.o
Enter the size of the array: 5
Enter the elements of the array:
4 7 2 4 9
Enter the element to search: 4
Choose search option:
1. Recursive Linear Search
2. Recursive Binary Search
Enter your choice (1 or 2): 2
Element found at index 3
Number of comparisons: 2
```



3. **Greedy Approach :**

**i) Dijkstra's Single Source Shortest Path Algorithm**

**CODE:**

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

```
#define MAX_VERTICES 100

// Function declarations
void dijkstra(int graph[MAX_VERTICES][MAX_VERTICES], int vertices, int source);

int main() {
    int vertices;

    // Input the number of vertices
    printf("Enter the number of vertices: ");
    scanf("%d", &vertices);

    // Input the adjacency matrix
    int graph[MAX_VERTICES][MAX_VERTICES];
    printf("Enter the adjacency matrix (use -1 for infinity):\n");
    for (int i = 0; i < vertices; i++) {
        for (int j = 0; j < vertices; j++) {
            scanf("%d", &graph[i][j]);
        }
    }

    // Input the source vertex
    int source;
    printf("Enter the source vertex: ");
    scanf("%d", &source);

    // Run Dijkstra's algorithm
    dijkstra(graph, vertices, source);

    return 0;
}

void dijkstra(int graph[MAX_VERTICES][MAX_VERTICES], int vertices, int source) {
    int distance[vertices]; // Array to store the distance from source to each vertex
    int visited[vertices];   // Array to keep track of visited vertices

    // Initialize distance and visited arrays
    for (int i = 0; i < vertices; i++) {
        distance[i] = INT_MAX;
        visited[i] = 0;
    }
```

```
    // Distance from source to itself is always 0
    distance[source] = 0;

    // Find shortest paths
    for (int count = 0; count < vertices - 1; count++) {
        // Find the vertex with the minimum distance value
        int minDistance = INT_MAX;
        int minIndex = -1;

        for (int v = 0; v < vertices; v++) {
            if (!visited[v] && distance[v] < minDistance) {
                minDistance = distance[v];
                minIndex = v;
            }
        }

        // Mark the selected vertex as visited
        visited[minIndex] = 1;

        // Update distance values of the adjacent vertices
        for (int v = 0; v < vertices; v++) {
            if (!visited[v] && graph[minIndex][v] != -1 && distance[minIndex] != INT_MAX &&
                distance[minIndex] + graph[minIndex][v] < distance[v]) {
                distance[v] = distance[minIndex] + graph[minIndex][v];
            }
        }
    }

    // Print the shortest distances
    printf("Shortest distances from the source vertex %d:\n", source);
    for (int i = 0; i < vertices; i++) {
        if (distance[i] == INT_MAX) {
            printf("Vertex %d is not reachable from the source\n", i);
        } else {
            printf("Vertex %d: %d\n", i, distance[i]);
        }
    }
}
```



**CODE SCREENSHOT:**

main.c

Save Run

```c
//21BCE0169

#include <stdio.h>
#include <limits.h>

#define MAX_VERTICES 100

// Function declarations
void dijkstra(int graph[MAX_VERTICES][MAX_VERTICES], int vertices, int
    source);

int main() {
    int vertices;

    // Input the number of vertices
    printf("Enter the number of vertices: ");
    scanf("%d", &vertices);

    // Input the adjacency matrix
    int graph[MAX_VERTICES][MAX_VERTICES];
    printf("Enter the adjacency matrix (use -1 for infinity):\n");
    for (int i = 0; i < vertices; i++) {
        for (int j = 0; j < vertices; j++) {
            scanf("%d", &graph[i][j]);
        }
    }

```

Output Clear

```
/tmp/qxNWIXKHGC.o
Enter the number of vertices: 3
Enter the adjacency matrix (use -1 for infinity):
0 2 5
-1 0 9
-1 -1 -1
Enter the source vertex: 0
Shortest distances from the source vertex 0:
Vertex 0: 0
Vertex 1: 2
Vertex 2: 5
```

**OUTPUT:**

Enter the number of vertices: 3
Enter the adjacency matrix (use -1 for infinity):
0 2 5
-1 0 9
-1 -1 -1
Enter the source vertex: 0
Shortest distances from the source vertex 0:
Vertex 0: 0
Vertex 1: 2
Vertex 2: 5

**OUTPUT SCREENSHOT:**

Output Clear

```
/tmp/qxNWIXKHGC.o
Enter the number of vertices: 3
Enter the adjacency matrix (use -1 for infinity):
0 2 5
-1 0 9
-1 -1 -1
Enter the source vertex: 0
Shortest distances from the source vertex 0:
Vertex 0: 0
Vertex 1: 2
Vertex 2: 5
```

**ii) Fractional Knapsack Problem for an airport permitted baggage scenario.**

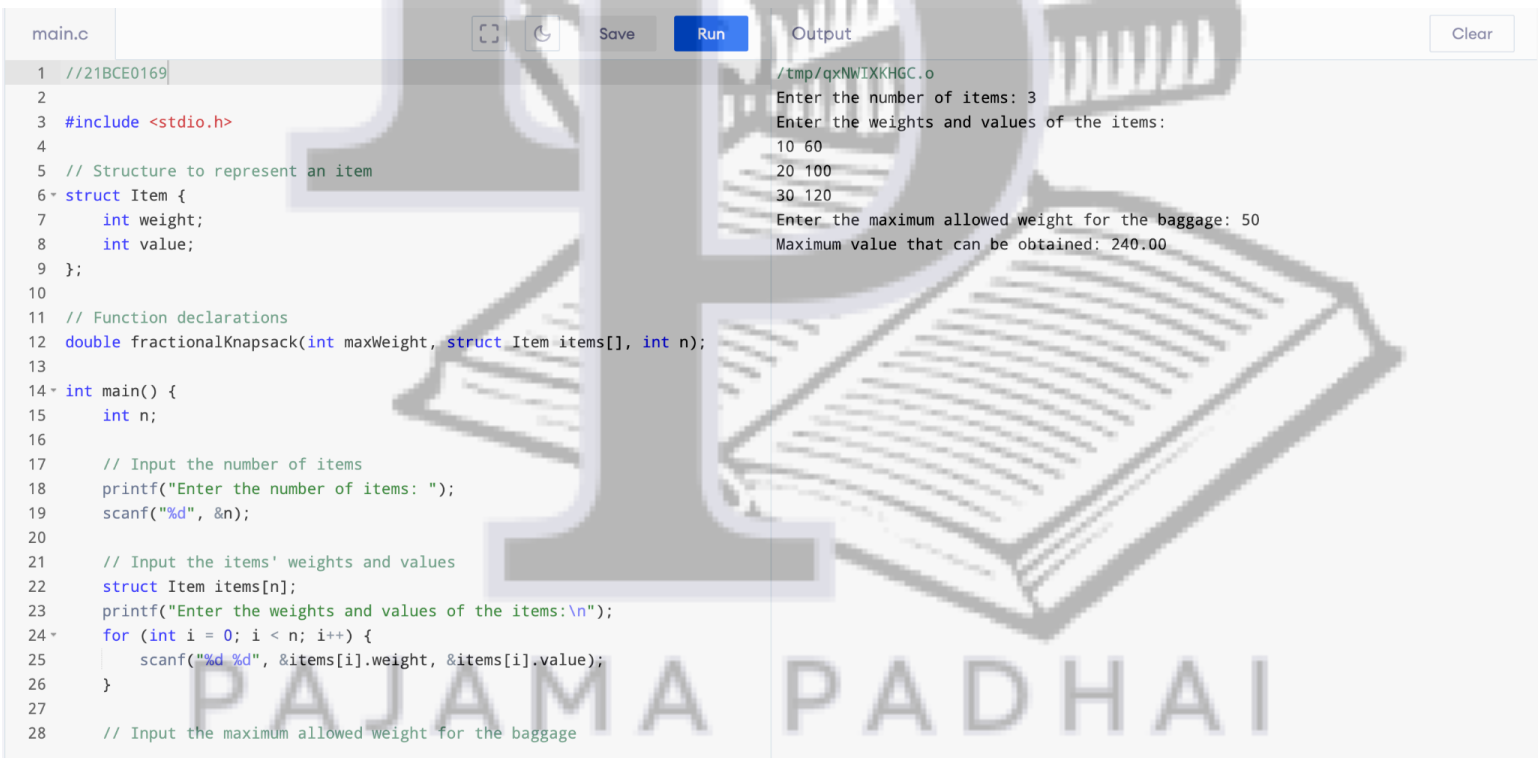
**CODE:**

```
#include <stdio.h>

// Structure to represent an item
struct Item {
    int weight;
    int value;
};

// Function declarations
double fractionalKnapsack(int maxWeight, struct Item items[], int n);

int main() {
    int n;

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

    // Input the items' weights and values
    struct Item items[n];
    printf("Enter the weights and values of the items:\n");
```

```
    for (int i = 0; i < n; i++) {
        scanf("%d %d", &items[i].weight, &items[i].value);
    }

    // Input the maximum allowed weight for the baggage
    int maxWeight;
    printf("Enter the maximum allowed weight for the baggage: ");
    scanf("%d", &maxWeight);

    // Calculate the maximum value using fractional knapsack
    double maxValue = fractionalKnapsack(maxWeight, items, n);

    // Print the result
    printf("Maximum value that can be obtained: %.2f\n", maxValue);

    return 0;
}

// Function to solve the fractional knapsack problem
double fractionalKnapsack(int maxWeight, struct Item items[], int n) {
    // Calculate the value-to-weight ratios for each item
    double ratios[n];
    for (int i = 0; i < n; i++) {
        ratios[i] = (double)items[i].value / items[i].weight;
    }

    // Sort items based on value-to-weight ratio in descending order
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (ratios[j] < ratios[j + 1]) {
                // Swap items
                double tempRatio = ratios[j];
                ratios[j] = ratios[j + 1];
                ratios[j + 1] = tempRatio;

                struct Item tempItem = items[j];
                items[j] = items[j + 1];
                items[j + 1] = tempItem;
            }
        }
    }

    // Initialize variables
```

```
    double totalValue = 0.0;
    int remainingWeight = maxWeight;

    // Iterate through sorted items and add to the baggage
    for (int i = 0; i < n; i++) {
        if (items[i].weight <= remainingWeight) {
            // Take the entire item
            totalValue += items[i].value;
            remainingWeight -= items[i].weight;
        } else {
            // Take a fraction of the item
            totalValue += ratios[i] * remainingWeight;
            break;
        }
    }

    return totalValue;
}
```

**CODE SCREENSHOT:**



```
main.c                                   Save   Run   Output                    Clear

1  //21BCE0169
2
3  #include <stdio.h>
4
5  // Structure to represent an item
6  struct Item {
7      int weight;
8      int value;
9  };
10
11 // Function declarations
12 double fractionalKnapsack(int maxWeight, struct Item items[], int n);
13
14 int main() {
15     int n;
16
17     // Input the number of items
18     printf("Enter the number of items: ");
19     scanf("%d", &n);
20
21     // Input the items' weights and values
22     struct Item items[n];
23     printf("Enter the weights and values of the items:\n");
24     for (int i = 0; i < n; i++) {
25         scanf("%d %d", &items[i].weight, &items[i].value);
26     }
27
28     // Input the maximum allowed weight for the baggage
```

```
/tmp/qxNWIXKHGC.o
Enter the number of items: 3
Enter the weights and values of the items:
10 60
20 100
30 120
Enter the maximum allowed weight for the baggage: 50
Maximum value that can be obtained: 240.00
```

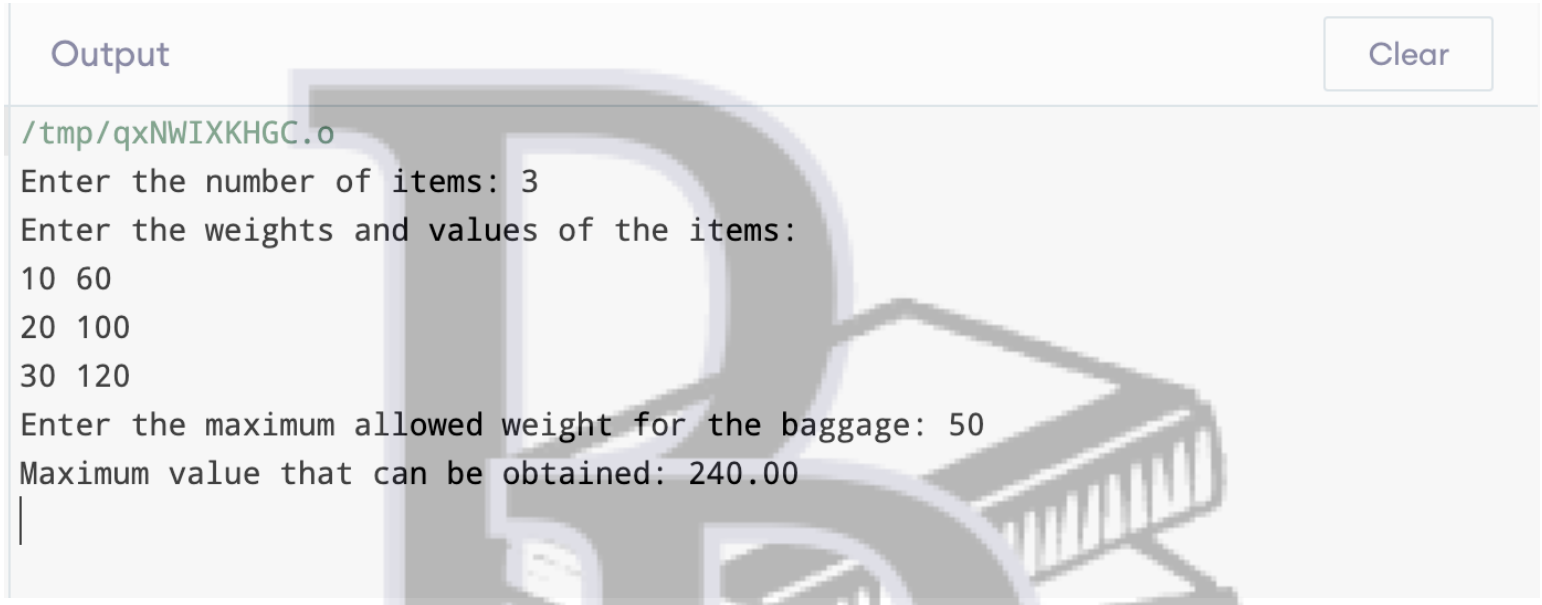


**OUTPUT:**

Enter the number of items: 3
Enter the weights and values of the items:
10 60

20 100
30 120
Enter the maximum allowed weight for the baggage: 50
Maximum value that can be obtained: 240.00

**OUTPUT SCREENSHOT:**




**4. Divide and Conquer Approach:**

**i) Quick Sort or Merge Sort to sort the details of mobile phones based on their RAM capacity**

**Quick Sort:**

**CODE:**

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

// Structure to represent mobile phone details
struct MobilePhone {
    char name[50];
    int ram;
    float price;
};

// Function declarations
void quickSort(struct MobilePhone phones[], int low, int high);
int partition(struct MobilePhone phones[], int low, int high);
void swap(struct MobilePhone *a, struct MobilePhone *b);
```

```
int main() {
    int n;

    // Input the number of mobile phones
    printf("Enter the number of mobile phones: ");
    scanf("%d", &n);

    // Input mobile phone details
    struct MobilePhone phones[n];
    printf("Enter the details of each mobile phone (name, RAM, price):\n");
    for (int i = 0; i < n; i++) {
        scanf("%s %d %f", phones[i].name, &phones[i].ram, &phones[i].price);
    }

    // Sort mobile phones based on RAM using Quick Sort
    quickSort(phones, 0, n - 1);

    // Display the sorted details
    printf("\nMobile phones sorted based on RAM capacity:\n");
    printf("%-20s %-10s %-10s\n", "Name", "RAM (GB)", "Price");
    for (int i = 0; i < n; i++) {
        printf("%-20s %-10d %-10.2f\n", phones[i].name, phones[i].ram, phones[i].price);
    }

    return 0;
}

// Function to perform Quick Sort on mobile phones based on RAM
void quickSort(struct MobilePhone phones[], int low, int high) {
    if (low < high) {
        // Find the pivot index such that elements smaller than the pivot
        // are on the left, and elements greater are on the right
        int pivotIndex = partition(phones, low, high);

        // Recursively sort the subarrays on both sides of the pivot
        quickSort(phones, low, pivotIndex - 1);
        quickSort(phones, pivotIndex + 1, high);
    }
}

// Function to partition the array and return the pivot index
int partition(struct MobilePhone phones[], int low, int high) {
```

```
    int pivot = phones[high].ram; // Choose the last element as the pivot
    int i = low - 1; // Index of the smaller element

    for (int j = low; j <= high - 1; j++) {
        // If the current element is smaller than or equal to the pivot
        if (phones[j].ram <= pivot) {
            i++;
            // Swap phones[i] and phones[j]
            swap(&phones[i], &phones[j]);
        }
    }

    // Swap phones[i + 1] and phones[high] to place the pivot in the correct position
    swap(&phones[i + 1], &phones[high]);

    return i + 1; // Return the pivot index
}

// Function to swap two MobilePhone structures
void swap(struct MobilePhone *a, struct MobilePhone *b) {
    struct MobilePhone temp = *a;
    *a = *b;
    *b = temp;
}
```

**CODE SCREENSHOT:**

main.c

```c
//21BCE0169

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

// Structure to represent mobile phone details
struct MobilePhone {
    char name[50];
    int ram;
    float price;
};

// Function declarations
void quickSort(struct MobilePhone phones[], int low, int high);
int partition(struct MobilePhone phones[], int low, int high);
void swap(struct MobilePhone *a, struct MobilePhone *b);

int main() {
    int n;

    // Input the number of mobile phones
    printf("Enter the number of mobile phones: ");
    scanf("%d", &n);

    // Input mobile phone details
    struct MobilePhone phones[n];
    printf("Enter the details of each mobile phone (name, RAM, price):\n");
```

Output

```
/tmp/qxNWIXKHGC.o
Enter the number of mobile phones: 3
Enter the details of each mobile phone (name, RAM, price):
iPhone 6 999.99
Samsung 8 899.99
OnePlus 12 849.99
iPhone 6 999.99

Samsung 8 899.99

OnePlus 12 849.99

Mobile phones sorted based on RAM capacity:
Name                RAM (GB)    Price
iPhone              6           999.99
Samsung             8           899.99
OnePlus             12          849.99
```

**OUTPUT:**

Enter the number of mobile phones: 3
Enter the details of each mobile phone (name, RAM, price):
iPhone 6 999.99
Samsung 8 899.99
OnePlus 12 849.99

iPhone 6 999.99

Samsung 8 899.99

OnePlus 12 849.99

Mobile phones sorted based on RAM capacity:

| Name | RAM (GB) | Price |
|---|---|---|
| iPhone | 6 | 999.99 |
| Samsung | 8 | 899.99 |
| OnePlus | 12 | 849.99 |

**OUTPUT SCREENSHOT:**

Output Clear

```
/tmp/qxNWIXKHGC.o
Enter the number of mobile phones: 3
Enter the details of each mobile phone (name, RAM, price):
iPhone 6 999.99
Samsung 8 899.99
OnePlus 12 849.99
iPhone 6 999.99

Samsung 8 899.99

OnePlus 12 849.99

Mobile phones sorted based on RAM capacity:
Name                RAM (GB)    Price
iPhone              6           999.99
Samsung             8           899.99
OnePlus             12          849.99
```

**Merge Sort:**

**CODE:**

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

// Structure to represent a mobile phone
struct MobilePhone {
    char name[50];
    int ram;
    float price;
};

// Function declarations
void merge(struct MobilePhone arr[], int l, int m, int r);
void mergeSort(struct MobilePhone arr[], int l, int r);
```

```
int main() {
    int n;

    // Input the number of mobile phones
    printf("Enter the number of mobile phones: ");
    scanf("%d", &n);

    // Input details of mobile phones
    struct MobilePhone phones[n];
    printf("Enter the details of mobile phones (name, RAM in GB, price):\n");
    for (int i = 0; i < n; i++) {
        scanf("%s %d %f", phones[i].name, &phones[i].ram, &phones[i].price);
    }

    // Sort mobile phones based on RAM using Merge Sort
    mergeSort(phones, 0, n - 1);

    // Print the sorted details
    printf("\nSorted details of mobile phones based on RAM:\n");
    printf("%-20s %-10s %-10s\n", "Name", "RAM (GB)", "Price");
    for (int i = 0; i < n; i++) {
        printf("%-20s %-10d %-10.2f\n", phones[i].name, phones[i].ram, phones[i].price);
    }

    return 0;
}

// Merge function for Merge Sort
void merge(struct MobilePhone arr[], int l, int m, int r) {
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    // Create temporary arrays
    struct MobilePhone L[n1], R[n2];

    // Copy data to temporary arrays L[] and R[]
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    // Merge the temporary arrays back into arr[l..r]
```

```
    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2) {
        if (L[i].ram <= R[j].ram) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    // Copy the remaining elements of L[], if there are any
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    // Copy the remaining elements of R[], if there are any
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

// Merge Sort function
void mergeSort(struct MobilePhone arr[], int l, int r) {
    if (l < r) {
        // Same as (l+r)/2, but avoids overflow for large l and r
        int m = l + (r - l) / 2;

        // Sort first and second halves
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);

        // Merge the sorted halves
        merge(arr, l, m, r);
    }
}
```

**CODE SCREENSHOT:**

main.c

```c
//21BCE0169

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

// Structure to represent a mobile phone
struct MobilePhone {
    char name[50];
    int ram;
    float price;
};

// Function declarations
void merge(struct MobilePhone arr[], int l, int m, int r);
void mergeSort(struct MobilePhone arr[], int l, int r);

int main() {
    int n;

    // Input the number of mobile phones
    printf("Enter the number of mobile phones: ");
    scanf("%d", &n);

    // Input details of mobile phones
    struct MobilePhone phones[n];
    printf("Enter the details of mobile phones (name, RAM in GB, price
        ):\n");
```

Output

```
/tmp/qxNWIXKHGC.o
Enter the number of mobile phones: 3
Enter the details of mobile phones (name, RAM in GB, price):
iPhone 8  859.99
Samsung 12  769.99
OnePlus 6  549.99
iPhone 8  859.99

Samsung 12  769.99

OnePlus 6  549.99

Sorted details of mobile phones based on RAM:
Name                RAM (GB)   Price
OnePlus             6          549.99
iPhone              8          859.99
Samsung             12         769.99
```

**OUTPUT:**

Enter the number of mobile phones: 3
Enter the details of mobile phones (name, RAM in GB, price):
iPhone 8  859.99
Samsung 12  769.99
OnePlus 6  549.99

iPhone 8  859.99

Samsung 12  769.99

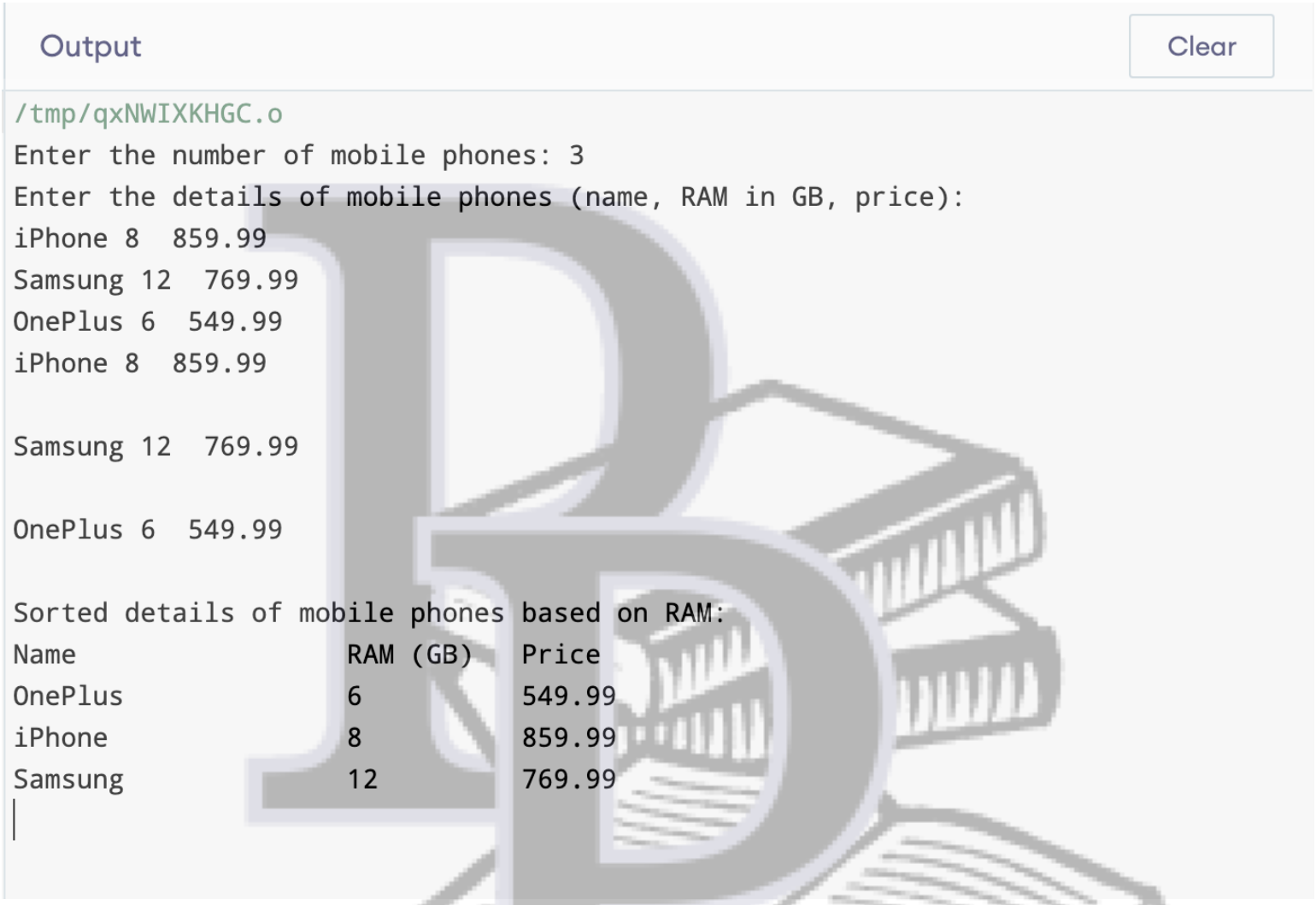
OnePlus 6  549.99



Sorted details of mobile phones based on RAM:

| Name | RAM (GB) | Price |
|---|---|---|
| OnePlus | 6 | 549.99 |
| iPhone | 8 | 859.99 |
| Samsung | 12 | 769.99 |

**OUTPUT SCREENSHOT:**



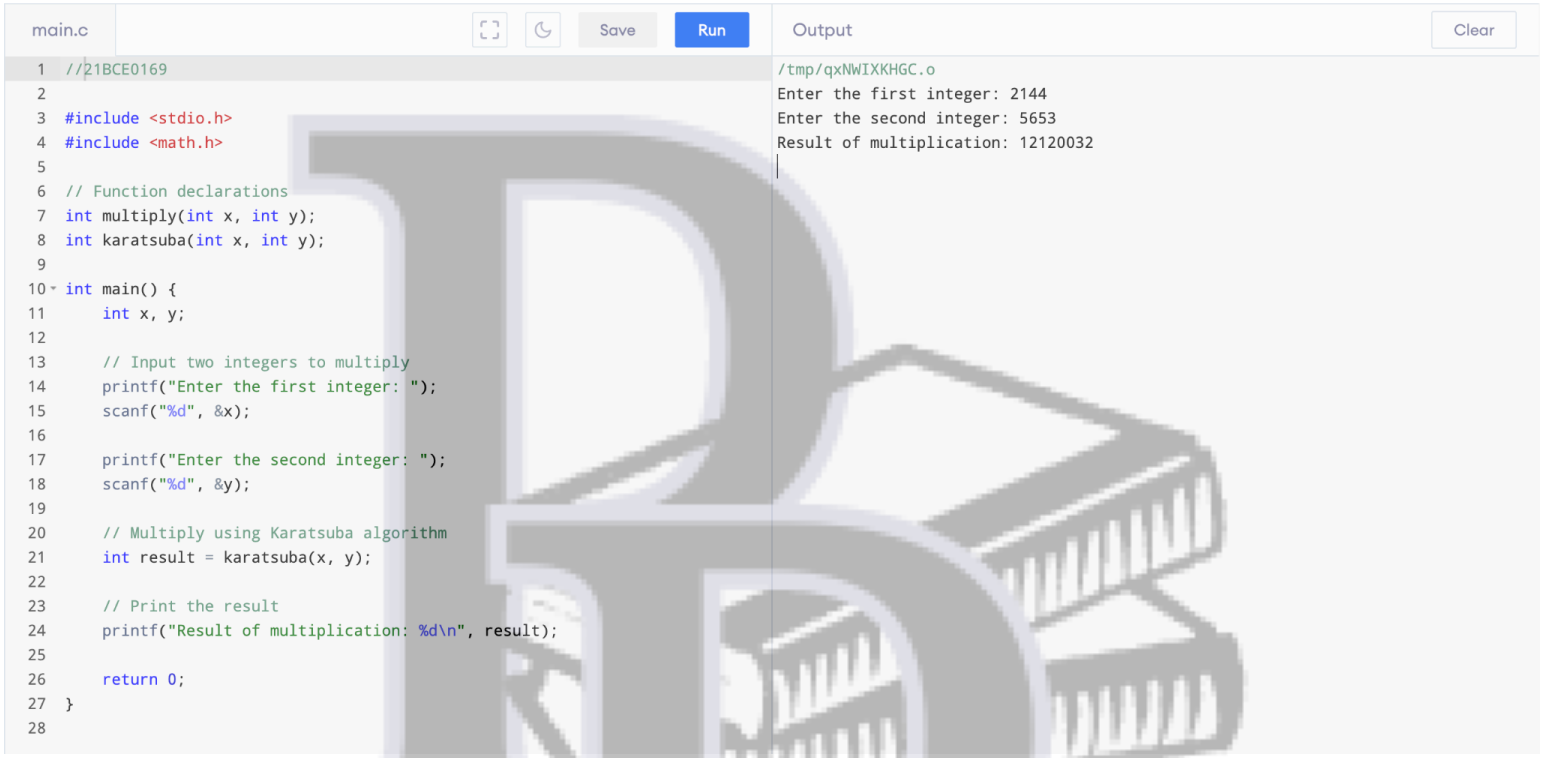

**ii) Karatsuba Faster Integer Multiplication Problem.**

**CODE:**

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

// Function declarations
int multiply(int x, int y);
int karatsuba(int x, int y);

int main() {
    int x, y;
```

```
    // Input two integers to multiply
    printf("Enter the first integer: ");
    scanf("%d", &x);

    printf("Enter the second integer: ");
    scanf("%d", &y);

    // Multiply using Karatsuba algorithm
    int result = karatsuba(x, y);

    // Print the result
    printf("Result of multiplication: %d\n", result);

    return 0;
}

// Function to calculate the product of two integers using Karatsuba algorithm
int karatsuba(int x, int y) {
    if (x < 10 || y < 10) {
        // Base case: Use simple multiplication for small integers
        return x * y;
    }

    // Calculate the number of digits in each integer
    int n = fmax(log10(x) + 1, log10(y) + 1);
    int halfN = n / 2;

    // Split the integers into two halves
    int a = x / (int)pow(10, halfN);
    int b = x % (int)pow(10, halfN);
    int c = y / (int)pow(10, halfN);
    int d = y % (int)pow(10, halfN);

    // Recursive steps of Karatsuba algorithm
    int ac = karatsuba(a, c);
    int bd = karatsuba(b, d);
    int adbc = karatsuba(a + b, c + d) - ac - bd;

    // Combine the results to get the final product
    return ac * (int)pow(10, 2 * halfN) + adbc * (int)pow(10, halfN) + bd;
}
```

**CODE SCREENSHOT:**




**OUTPUT:**

Enter the first integer: 2144
Enter the second integer: 5653
Result of multiplication: 12120032

**OUTPUT SCREENSHOT:**