School of Computer Science and Engineering

M.Tech (Integrated) Fall Semester 2024-25

CSI3001 - Cloud Computing Methodologies

Assignment – 3

22MIC0130

Vijay Pavan A

SLOT: B1+TB1 / B2+TB2 **Due Date: 25/09/2024**

Q. No	Questions
1.	Implement the following scheduling algorithm for VM allocation in cloud environment i. First Come First Serve (FCFS) ii. Shortest Job First (SJF) iii. MAX_MIN iv. MIN_MIN

Faculty Incharge

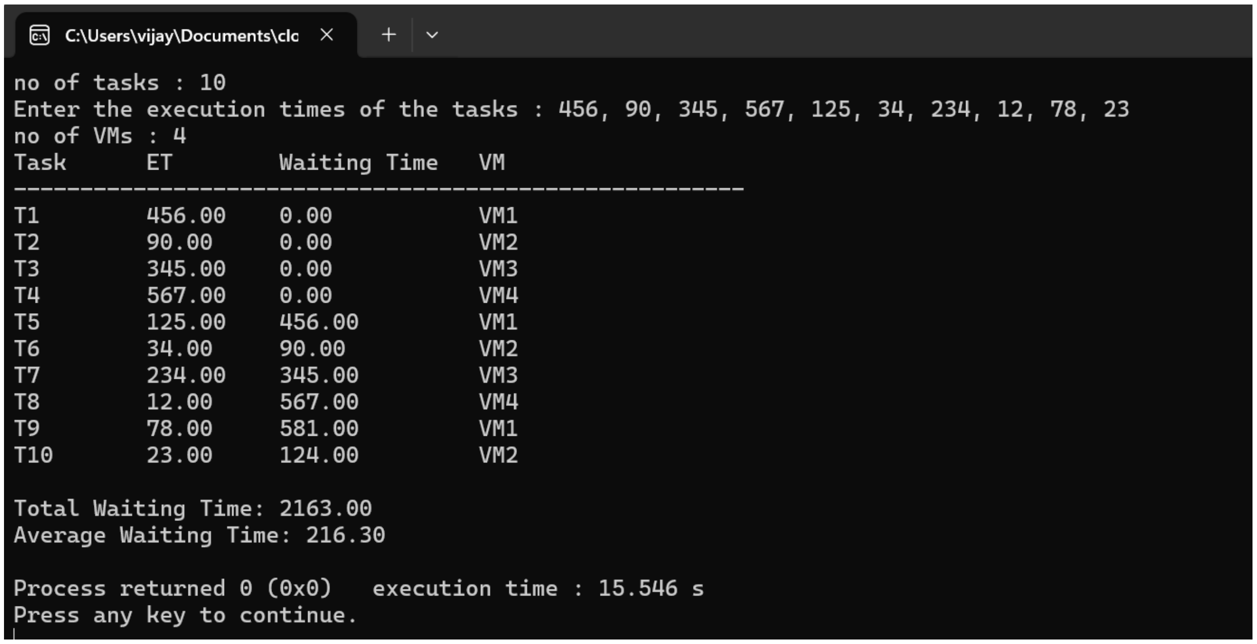
Dr. S. U. Muthunagai

1) i) Code :

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Task {
   char name[10];
   double executionTime;
   double waitingTime;
```

```
} Task;
typedef struct VirtualMachine {
   char name[10];
   double executionTime;
} VirtualMachine;
void scheduleTasks(Task* tasks, int taskCount, VirtualMachine* vms, int vmCount) {
   for (int i = 0; i < taskCount; i++) {
      tasks[i].waitingTime = 0;
      if (i < vmCount) {
         vms[i].executionTime = tasks[i].executionTime;
      } else {
         VirtualMachine* vm = &vms[i % vmCount];
         tasks[i].waitingTime = vm->executionTime;
         vm->executionTime += tasks[i].executionTime;
      }
   }
}
void printSchedule(Task* tasks, int taskCount, VirtualMachine* vms, int vmCount) {
   double totalWaitingTime = 0;
   printf("%-10s%-10s%-15s%-10s\n", "Task", "ET", "Waiting Time", "VM");
   printf("------------------------------------------------------\n");
   for (int i = 0; i < taskCount; i++) {
      printf("%-10s%-10.2f%-15.2f%-10s\n", tasks[i].name, tasks[i].executionTime,
tasks[i].waitingTime, vms[i % vmCount].name);
      totalWaitingTime += tasks[i].waitingTime;
   }
   printf("\nTotal Waiting Time: %.2f\n", totalWaitingTime);
   printf("Average Waiting Time: %.2f\n", (vmCount > 0) ? totalWaitingTime / taskCount : 0);
}
Task* getTasks(int taskCount) {
   Task* tasks = (Task*) malloc(taskCount * sizeof(Task));
   printf("Enter the execution times of the tasks : ");
   scanf("%lf, %lf, %lf, %lf, %lf, %lf, %lf, %lf, %lf, %lf", &tasks[0].executionTime,
&tasks[1].executionTime, &tasks[2].executionTime,
&tasks[3].executionTime,&tasks[4].executionTime,
&tasks[5].executionTime,&tasks[6].executionTime,
&tasks[7].executionTime,&tasks[8].executionTime, &tasks[9].executionTime);
   for (int i = 1; i <= taskCount; i++)
   {
      sprintf(tasks[i-1].name, "T%d", i);
   }
   return tasks;
}
VirtualMachine* getVirtualMachines(int vmCount) {
   VirtualMachine* vms = (VirtualMachine*) malloc(vmCount * sizeof(VirtualMachine));
```

```
    for (int i = 1; i <= vmCount; i++) {
        sprintf(vms[i-1].name, "VM%d", i);
        vms[i-1].executionTime = 0;
    }
    return vms;
}
int main() {
    int taskCount, vmCount;
    printf("no of tasks : ");
    scanf("%d", &taskCount);
    Task* tasks = getTasks(taskCount);
    printf("no of VMs : ");
    scanf("%d", &vmCount);
    VirtualMachine* vms = getVirtualMachines(vmCount);
    scheduleTasks(tasks, taskCount, vms, vmCount);
    printSchedule(tasks, taskCount, vms, vmCount);
    free(tasks);
    free(vms);
    return 0;
}
```

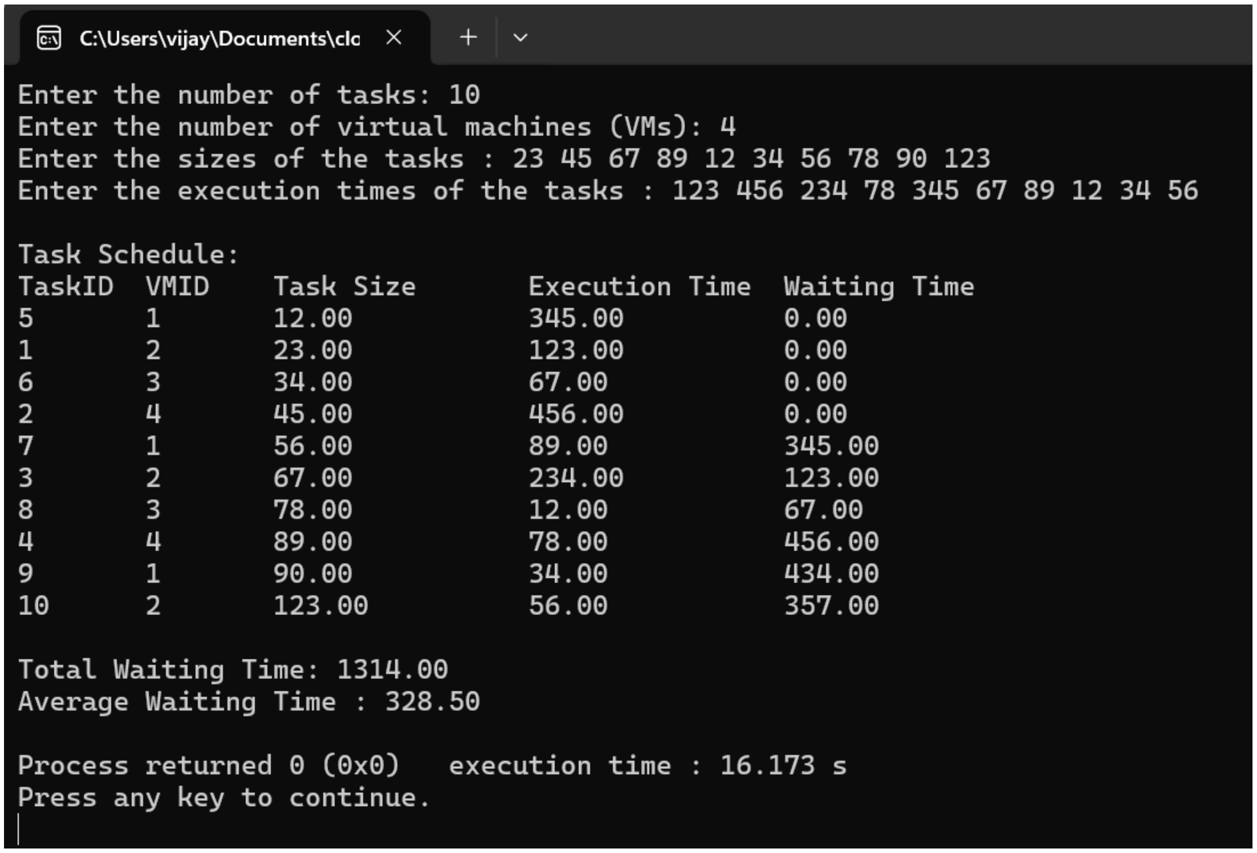


```
no of tasks : 10
Enter the execution times of the tasks : 456, 90, 345, 567, 125, 34, 234, 12, 78, 23
no of VMs : 4
Task      ET        Waiting Time   VM
-------------------------------------------------------
T1        456.00    0.00           VM1
T2        90.00     0.00           VM2
T3        345.00    0.00           VM3
T4        567.00    0.00           VM4
T5        125.00    456.00         VM1
T6        34.00     90.00          VM2
T7        234.00    345.00         VM3
T8        12.00     567.00         VM4
T9        78.00     581.00         VM1
T10       23.00     124.00         VM2

Total Waiting Time: 2163.00
Average Waiting Time: 216.30

Process returned 0 (0x0)   execution time : 15.546 s
Press any key to continue.
```

ii) Code :

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Task {
    int taskId;
    float size;
    float executionTime;
    int vmId;
```

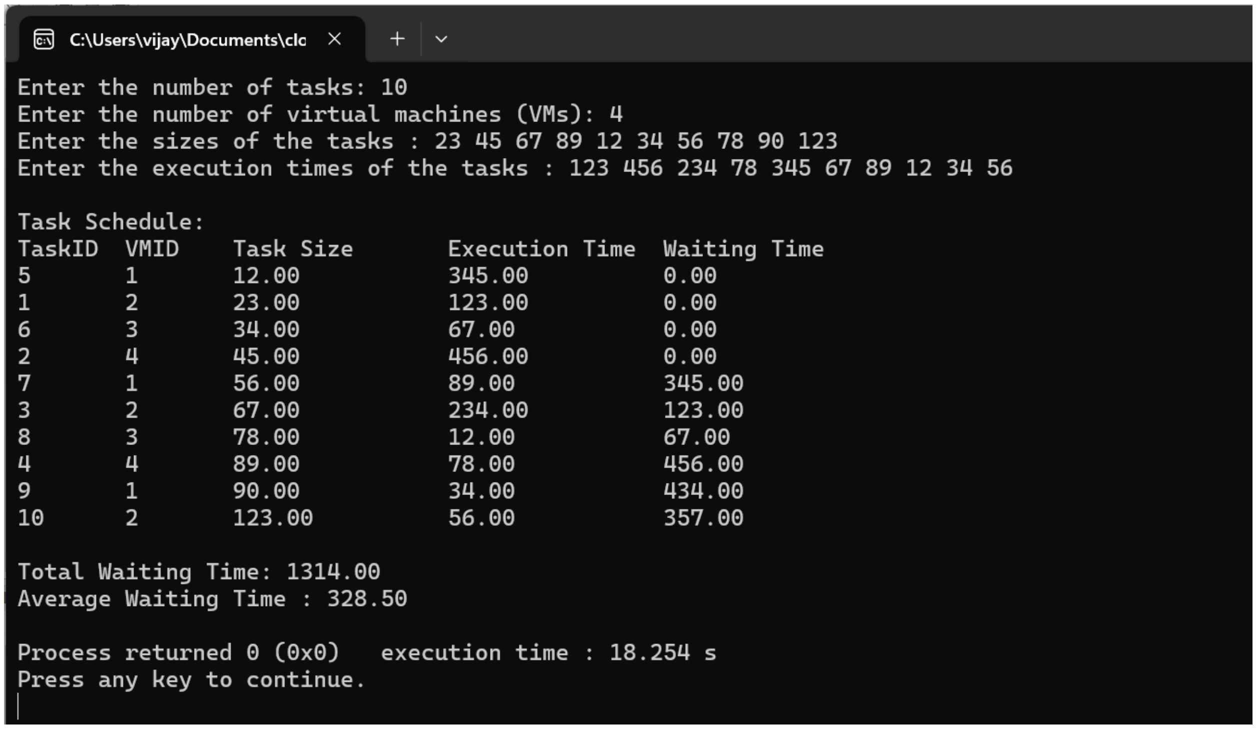
```
} Task;
void sortBySize(Task* tasks, int taskCount) {
    for (int i = 0; i < taskCount - 1; i++) {
        for (int j = 0; j < taskCount - 1 - i; j++) {
            if (tasks[j].size > tasks[j + 1].size) {
                Task temp = tasks[j];
                tasks[j] = tasks[j + 1];
                tasks[j + 1] = temp;
            }
        }
    }
}
int main() {
    int taskCount, vmCount;
    float* vmWaitingTime;
    float* previousExecutionTime;
    printf("Enter the number of tasks: ");
    scanf("%d", &taskCount);
    printf("Enter the number of virtual machines (VMs): ");
    scanf("%d", &vmCount);
    Task* tasks = (Task*)malloc(taskCount * sizeof(Task));
    vmWaitingTime = (float*)malloc(vmCount * sizeof(float));
    previousExecutionTime = (float*)malloc(vmCount * sizeof(float));
    for (int i = 0; i < vmCount; i++) {
        vmWaitingTime[i] = 0.0f;
        previousExecutionTime[i] = 0.0f;
    }
    printf("Enter the sizes of the tasks : ");
    for (int i = 0; i < taskCount; i++) {
        scanf("%f", &tasks[i].size);
    }
    printf("Enter the execution times of the tasks : ");
    for (int i = 0; i < taskCount; i++) {
        scanf("%f", &tasks[i].executionTime);
    }
    for (int i = 0; i < taskCount; i++) {
        tasks[i].taskId = i + 1;
        tasks[i].vmId = 0;
    }
    sortBySize(tasks, taskCount);
    printf("\nTask Schedule:\n");
    printf("TaskID\tVMID\tTask Size\tExecution Time\tWaiting Time\n");
    for (int i = 0; i < taskCount; i++) {
        tasks[i].vmId = i % vmCount + 1;
        float waitingTime = (previousExecutionTime[i % vmCount] > 0) ? vmWaitingTime[i %
vmCount] + previousExecutionTime[i % vmCount] : 0;

        printf("%d\t%d\t%.2f\t\t%.2f\t\t%.2f\n", tasks[i].taskId, tasks[i].vmId, tasks[i].size,
tasks[i].executionTime, waitingTime);

        vmWaitingTime[i % vmCount] += previousExecutionTime[i % vmCount];
        previousExecutionTime[i % vmCount] = tasks[i].executionTime;
    }
    float totalWaitingTime = 0.0f;
    for (int i = 0; i < vmCount; i++) {
        totalWaitingTime += vmWaitingTime[i];
    }
    float averageWaitingTime = totalWaitingTime / vmCount;
    printf("\nTotal Waiting Time: %.2f\n", totalWaitingTime);
    printf("Average Waiting Time : %.2f\n", averageWaitingTime);

    free(tasks);
    free(vmWaitingTime);
    free(previousExecutionTime);
    return 0;
}
```



```
Enter the number of tasks: 10
Enter the number of virtual machines (VMs): 4
Enter the sizes of the tasks : 23 45 67 89 12 34 56 78 90 123
Enter the execution times of the tasks : 123 456 234 78 345 67 89 12 34 56

Task Schedule:
TaskID  VMID    Task Size       Execution Time  Waiting Time
5       1       12.00           345.00          0.00
1       2       23.00           123.00          0.00
6       3       34.00           67.00           0.00
2       4       45.00           456.00          0.00
7       1       56.00           89.00           345.00
3       2       67.00           234.00          123.00
8       3       78.00           12.00           67.00
4       4       89.00           78.00           456.00
9       1       90.00           34.00           434.00
10      2       123.00          56.00           357.00

Total Waiting Time: 1314.00
Average Waiting Time : 328.50

Process returned 0 (0x0)   execution time : 16.173 s
Press any key to continue.
```

iii) Code :

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Task {
    int taskId;
    float size;
    float executionTime;
    int vmId;
} Task;
void sortBySize(Task* tasks, int taskCount) {
    for (int i = 0; i < taskCount - 1; i++) {
        for (int j = 0; j < taskCount - 1 - i; j++) {
            if (tasks[j].size > tasks[j + 1].size) {
                Task temp = tasks[j];
                tasks[j] = tasks[j + 1];
                tasks[j + 1] = temp;
            }
        }
    }
}
int main() {
    int taskCount, vmCount;
    float* vmWaitingTime;
    float* previousExecutionTime;
    printf("Enter the number of tasks: ");
    scanf("%d", &taskCount);
    printf("Enter the number of virtual machines (VMs): ");
    scanf("%d", &vmCount);
    Task* tasks = (Task*)malloc(taskCount * sizeof(Task));
    vmWaitingTime = (float*)malloc(vmCount * sizeof(float));
    previousExecutionTime = (float*)malloc(vmCount * sizeof(float));
    for (int i = 0; i < vmCount; i++) {
        vmWaitingTime[i] = 0.0f;
        previousExecutionTime[i] = 0.0f;
    }
    printf("Enter the sizes of the tasks : ");
    for (int i = 0; i < taskCount; i++) {
        scanf("%f", &tasks[i].size);
    }
    printf("Enter the execution times of the tasks : ");
    for (int i = 0; i < taskCount; i++) {
        scanf("%f", &tasks[i].executionTime);
    }
    for (int i = 0; i < taskCount; i++) {
        tasks[i].taskId = i + 1;
        tasks[i].vmId = 0;
    }
    sortBySize(tasks, taskCount);
    printf("\nTask Schedule:\n");
    printf("TaskID\tVMID\tTask Size\tExecution Time\tWaiting Time\n");
    for (int i = 0; i < taskCount; i++) {
        tasks[i].vmId = i % vmCount + 1;
        float waitingTime = (previousExecutionTime[i % vmCount] > 0) ? vmWaitingTime[i %
vmCount] + previousExecutionTime[i % vmCount] : 0;

        printf("%d\t%d\t%.2f\t\t%.2f\t\t%.2f\n", tasks[i].taskId, tasks[i].vmId, tasks[i].size,
tasks[i].executionTime, waitingTime);

        vmWaitingTime[i % vmCount] += previousExecutionTime[i % vmCount];
        previousExecutionTime[i % vmCount] = tasks[i].executionTime;
    }
    float totalWaitingTime = 0.0f;
    for (int i = 0; i < vmCount; i++) {
        totalWaitingTime += vmWaitingTime[i];
    }
    float averageWaitingTime = totalWaitingTime / vmCount;
    printf("\nTotal Waiting Time: %.2f\n", totalWaitingTime);
    printf("Average Waiting Time : %.2f\n", averageWaitingTime);
    free(tasks);
    free(vmWaitingTime);
    free(previousExecutionTime);
    return 0;
}
```



C:\Users\vijay\Documents\clo

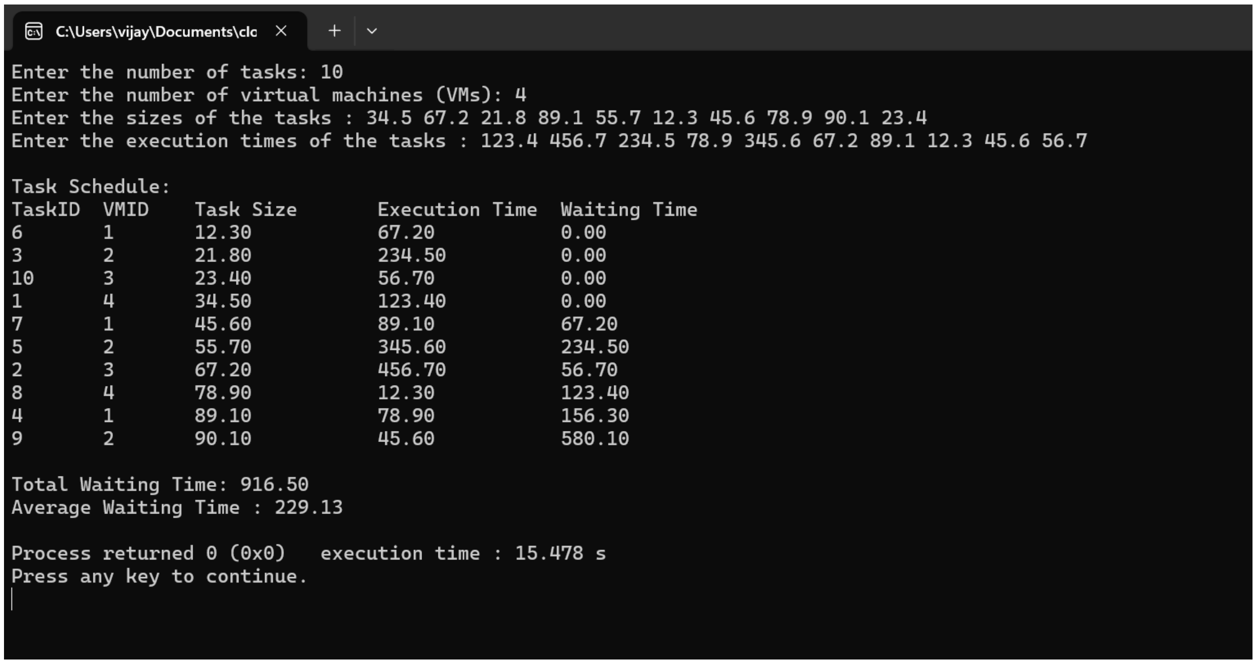
```
Enter the number of tasks: 10
Enter the number of virtual machines (VMs): 4
Enter the sizes of the tasks : 23 45 67 89 12 34 56 78 90 123
Enter the execution times of the tasks : 123 456 234 78 345 67 89 12 34 56

Task Schedule:
TaskID  VMID    Task Size       Execution Time  Waiting Time
5       1       12.00           345.00          0.00
1       2       23.00           123.00          0.00
6       3       34.00           67.00           0.00
2       4       45.00           456.00          0.00
7       1       56.00           89.00           345.00
3       2       67.00           234.00          123.00
8       3       78.00           12.00           67.00
4       4       89.00           78.00           456.00
9       1       90.00           34.00           434.00
10      2       123.00          56.00           357.00

Total Waiting Time: 1314.00
Average Waiting Time : 328.50

Process returned 0 (0x0)   execution time : 18.254 s
Press any key to continue.
```

iv) Code

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Task {
  int taskId;
  float size;
  float executionTime;
  int vmId;
} Task;
void sortBySize(Task* tasks, int taskCount) {
  for (int i = 0; i < taskCount - 1; i++) {
    for (int j = 0; j < taskCount - 1 - i; j++) {
      if (tasks[j].size > tasks[j + 1].size) {
        Task temp = tasks[j];
        tasks[j] = tasks[j + 1];
        tasks[j + 1] = temp;
      }
    }
  }
}
int main() {
  int taskCount, vmCount;
  float* vmWaitingTime;
  float* previousExecutionTime;
  printf("Enter the number of tasks: ");
```

```
    scanf("%d", &taskCount);
    printf("Enter the number of virtual machines (VMs): ");
    scanf("%d", &vmCount);
    Task* tasks = (Task*)malloc(taskCount * sizeof(Task));
    vmWaitingTime = (float*)malloc(vmCount * sizeof(float));
    previousExecutionTime = (float*)malloc(vmCount * sizeof(float));
    for (int i = 0; i < vmCount; i++) {
        vmWaitingTime[i] = 0.0f;
        previousExecutionTime[i] = 0.0f;
    }
    printf("Enter the sizes of the tasks : ");
    for (int i = 0; i < taskCount; i++) {
        scanf("%f", &tasks[i].size);
    }
    printf("Enter the execution times of the tasks : ");
    for (int i = 0; i < taskCount; i++) {
        scanf("%f", &tasks[i].executionTime);
    }
    for (int i = 0; i < taskCount; i++) {
        tasks[i].taskId = i + 1;
        tasks[i].vmId = 0;
    }
    sortBySize(tasks, taskCount);
    printf("\nTask Schedule:\n");
    printf("TaskID\tVMID\tTask Size\tExecution Time\tWaiting Time\n");
    for (int i = 0; i < taskCount; i++) {
        tasks[i].vmId = i % vmCount + 1;
        float waitingTime = (previousExecutionTime[i % vmCount] > 0) ? vmWaitingTime[i %
vmCount] + previousExecutionTime[i % vmCount] : 0;
        printf("%d\t%d\t%.2f\t\t%.2f\t\t%.2f\n", tasks[i].taskId, tasks[i].vmId, tasks[i].size,
tasks[i].executionTime, waitingTime);
        vmWaitingTime[i % vmCount] += previousExecutionTime[i % vmCount];
        previousExecutionTime[i % vmCount] = tasks[i].executionTime;
    }
    float totalWaitingTime = 0.0f;
    for (int i = 0; i < vmCount; i++) {
        totalWaitingTime += vmWaitingTime[i];
    }
    float averageWaitingTime = totalWaitingTime / vmCount;
    printf("\nTotal Waiting Time: %.2f\n", totalWaitingTime);
    printf("Average Waiting Time : %.2f\n", averageWaitingTime);
    free(tasks);
    free(vmWaitingTime);
    free(previousExecutionTime);
    return 0;
```

```
}
```



C:\Users\vijay\Documents\clo

```
Enter the number of tasks: 10
Enter the number of virtual machines (VMs): 4
Enter the sizes of the tasks : 34.5 67.2 21.8 89.1 55.7 12.3 45.6 78.9 90.1 23.4
Enter the execution times of the tasks : 123.4 456.7 234.5 78.9 345.6 67.2 89.1 12.3 45.6 56.7

Task Schedule:
TaskID  VMID    Task Size       Execution Time  Waiting Time
6       1       12.30           67.20           0.00
3       2       21.80           234.50          0.00
10      3       23.40           56.70           0.00
1       4       34.50           123.40          0.00
7       1       45.60           89.10           67.20
5       2       55.70           345.60          234.50
2       3       67.20           456.70          56.70
8       4       78.90           12.30           123.40
4       1       89.10           78.90           156.30
9       2       90.10           45.60           580.10

Total Waiting Time: 916.50
Average Waiting Time : 229.13

Process returned 0 (0x0)   execution time : 15.478 s
Press any key to continue.
```