ARTIFICIAL INTELLIGENCE PROJECT ON IMPLEMENTING UCS ALGORITHM FOR PROCESS SCHEDULING PROBLEMS

AIM:

To minimize the run time of Round Robin Algorithm in Operating Systems for Process Scheduling with Uniform Cost Search Algorithm.

Theory:

Process Scheduling involves determining the order in which the processes are executed on the Central Processing Unit (CPU) of a computer system. The main aim of doing process scheduling is to minimize the waiting time, speeding up the CPU execution time.In Process scheduling,we are ensuring that the CPU time and resources are utilized efficiently.

To schedule the processes, there are many algorithms such as **First-come-First-Served(FCFS), Shortest Job First Scheduler (SJF), Round Robin, Priority Scheduling, Multilevel Queue Scheduling,etc.** In this project, we are going to concentrate on comparing the Round Robin Algorithm with the Uniform Cost Search Algorithm,
My proposal is that we can make use of Uniform Cost Search Algorithm (UCS) to do Process Scheduling, to calculate the average waiting time in an Efficient Manner than Round Robin Algorithm in case of single CPU Burst time.

Why it is Important for Process Scheduling:

We are talking about process scheduling throughout this assignment, but we should first understand the importance of scheduling the processes. For every millisecond, the operating system is handling several IO interrupts, handles many processes, threads, etc. So, there must be no wastage of time in handling these processes or shifting from one process to another processes. If even a single second is wasted, it may waste major resources of the CPU. We should try our level best to make the best use of the CPU. So, for this we can reduce the waiting time of processes handled by CPU and we can make best arrangements to reduce the run time of handling these processes.

UNIFORM COST SEARCH:

This is one of the optimization Algorithms used in Artificial Intelligence to find the optimal path in graphs. We can use this algorithm to find the optimal scheduling path in Operating Systems. In UCS, we will use the priority Queue.

- First the initial or source code is inserted into the Queue, after that based on the cost of the nodes, its children are inserted into the Queue.
- Before inserting the child nodes into the Queue, we will check whether the nodes are already in the Queue or not.

- If the child nodes are not in the queue, then the nodes are added into the Queue.
- If the node is already in the queue, then the cost of the node inside the queue is compared with the node which get in the iteration.
- If the cost of the child node is lesser than the cost of the node in the Queue, the node in the queue can be replaced with this node with the lower cost.
- If it is not, then the node inside the node will be maintained as such with the same cost.
- This process will be continued till the Queue gets empty.
- At last we will get the optimal path from the source node to the destination or goal node.

ALGORITHM:

```
function uniform_cost_search(Problem) returns a solution or failure
        node ←a node with STATE.problem.Initial.Pathcost=0
        frontier ← a prior queue ordered by path_cost with node as the only element
        explored ← an empty set()
loop do
        If Empty ?(frontier) then return failure
        node ← pop(frontier)
        If Problem..GOALTEST(node.STATE) then return solution(node)
        Add node.STATE to explored
        For each action in PROBLEM.ACTIONS(node.STATE)
                child ← child_NODE (Problem,noe,action)
                If child.STATE is not in explored and frontier
                        frontier ← INSERT(child.frontier)
                Elif child.STATE is in frontier with higher Pathcost then
                        Replace that frontier node with child
```

As Input for the algorithms we are considering a dataset "**process_schedling-Sheet1"** which consists of three columns **Process_id, Arrival_time, Burst_time** with 60 Rows.

ROUND ROBIN CODE FOR PROCESS SCHEDULING:

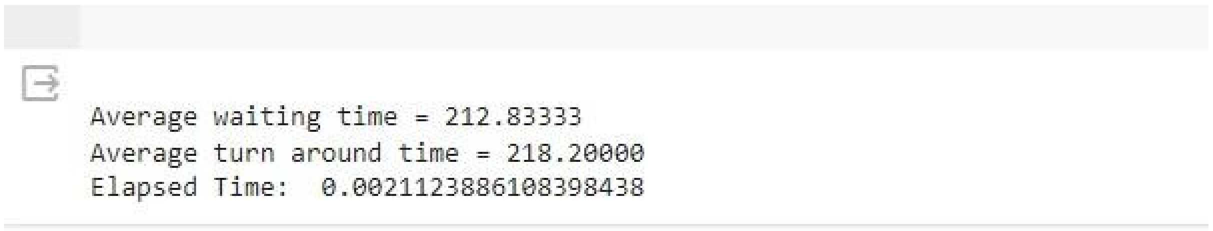
```python
import csv
import time

class Process:
    def __init__(self, process_id, arrival_time, burst_time):
        self.process_id = process_id
        self.arrival_time = arrival_time
        self.burst_time = burst_time
```

```python
def findWaitingTime(processes, n, bt, wt, quantum):
    rem_bt = [0] * n
    for i in range(n):
        rem_bt[i] = bt[i]

    t = 0  # Current time
    queue = []  # Ready queue

    while True:
        done = True

        # Enqueue processes with arrival time <= current time
        for i in range(n):
            if processes[i].arrival_time <= t and rem_bt[i] > 0:
                done = False
                queue.append(i)

        # Dequeue and process each process in the ready queue
        while queue:
            i = queue.pop(0)

            if rem_bt[i] > quantum:
                t += quantum
                rem_bt[i] -= quantum
            else:
                t = t + rem_bt[i]
                wt[i] = t - processes[i].burst_time -
processes[i].arrival_time
                rem_bt[i] = 0

        if done:
            break

def findTurnAroundTime(processes, n, bt, wt, tat):
    for i in range(n):
        tat[i] = bt[i] + wt[i]

def findavgTime(processes, n, bt, quantum):
    wt = [0] * n
    tat = [0] * n
    findWaitingTime(processes, n, bt, wt, quantum)
    findTurnAroundTime(processes, n, bt, wt, tat)


    total_wt = 0
    total_tat = 0
    for i in range(n):
        total_wt = total_wt + wt[i]
        total_tat = total_tat + tat[i]

    print("\nAverage waiting time = %.5f " % (total_wt / n))
    print("Average turn around time = %.5f " % (total_tat / n))

if __name__ == "__main__":
    # Provide the absolute file path
    filename = "process_scheduling - Sheet1.csv"

    processes = []
    burst_time = []

    # Read inputs from CSV file
    with open(filename, "r") as file:
        reader = csv.reader(file)
        next(reader)  # Skip header row
        for row in reader:
            process_id, arrival_time, burst = map(int, row)
            processes.append(Process(process_id, arrival_time, burst))
            burst_time.append(burst)

    n = len(processes)
    quantum = 2
    start = time.time()

    # Sort processes based on arrival time
    processes.sort(key=lambda x: x.arrival_time)

    findavgTime(processes, n, burst_time, quantum)

    end = time.time()
    elapsed_time = end - start
    print("Elapsed Time: ", elapsed_time)
```

Code Explanation:

- At the start of the program inside the class, with the help of the constructor the process_id, arrival_time, burst_time are initialized.
- In the findWaiting function, the waiting time for each process is calculated based on the arrival time and the time quantum
- Based on the time quantum, the processes are preempted and the final waiting time for each process is calculated.
- In the function, findTurnAroundTime, the turnaround time for each process is calculated using the formula **TurnAround Time= Burst time- waiting time.**
- Then with the help of findAvgTime function, the average waiting time is calculated
- Then the overall execution time is also calculated for the entire process.

OUTPUT:



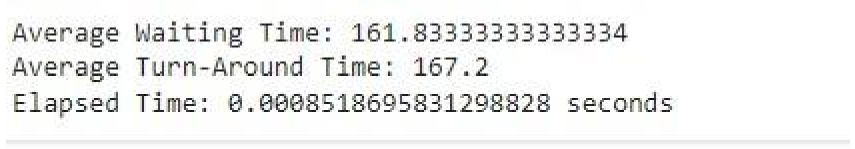
```
Average waiting time = 212.83333
Average turn around time = 218.20000
Elapsed Time:  0.0021123886108398438
```

CODE FOR UNIFORM COST SEARCH FOR PROCESS SCHEDULING:

```
import heapq
import csv
import time

start = time.time()

class Process:
    def __init__(self, process_id, arrival_time, burst_time):
        self.process_id = process_id
        self.arrival_time = arrival_time
        self.burst_time = burst_time
    def __lt__(self, other):
        return self.burst_time < other.burst_time

def ucs_scheduling(processes):
    completion_time = 0
    waiting_time = 0
    turn_around_time = 0  # Added to track turn-around time

    priority_queue = []
    heapq.heapify(priority_queue)

    for process in processes:
        if process.arrival_time > completion_time:
            completion_time = process.arrival_time

        waiting_time += completion_time - process.arrival_time
        completion_time += process.burst_time
        turn_around_time += completion_time - process.arrival_time  #
Update turn-around time


        heapq.heappush(priority_queue, process)

    average_waiting_time = waiting_time / len(processes)
    average_turn_around_time = turn_around_time / len(processes)

    return average_waiting_time, average_turn_around_time  # Return both
average waiting time and turn-around time

def read_processes_from_csv(filename):
    processes = []

    with open(filename, "r") as file:
        reader = csv.reader(file)
        next(reader)  # Skip header row
        for row in reader:
            process_id, arrival_time, burst_time = map(int, row)
            processes.append(Process(process_id, arrival_time,
burst_time))
```

```
    return processes

# Example usage:
if __name__ == "__main__":
    # Provide the absolute file path
    filename = "process_scheduling - Sheet1.csv"

    # Read processes from CSV file
    processes = read_processes_from_csv(filename)

    # Calculate both average waiting time and turn-around time
    average_waiting_time, average_turn_around_time =
ucs_scheduling(processes)

    end = time.time()
    elapsed_time = end - start

    print(f"Average Waiting Time: {average_waiting_time}")
    print(f"Average Turn-Around Time: {average_turn_around_time}")
    print(f"Elapsed Time: {elapsed_time} seconds")
```

OUTPUT:



```
Average Waiting Time: 161.83333333333334
Average Turn-Around Time: 167.2
Elapsed Time: 0.0008518695831298828 seconds
```

CODE EXPLANATION FOR UNIFORM COST SEARCH ALGORITHM:

- In the starting of the program, inside the class with the help of the constructor, the process id, arrival time, burst time are initialized.
- With the predefined __lt__ method, the processes are compared based on their burst time and passed to the below functions.
- In the **ucs_scheduling** function, the completion_time, turn_around_time, waiting_time are calculated.
- The function takes a list of Process objects each representing a process.
- The function maintains a Priority Queue to schedule the process based on their burst times.
- Then the **'completion_time'** is calculated by considering arrival time for each process and added together.

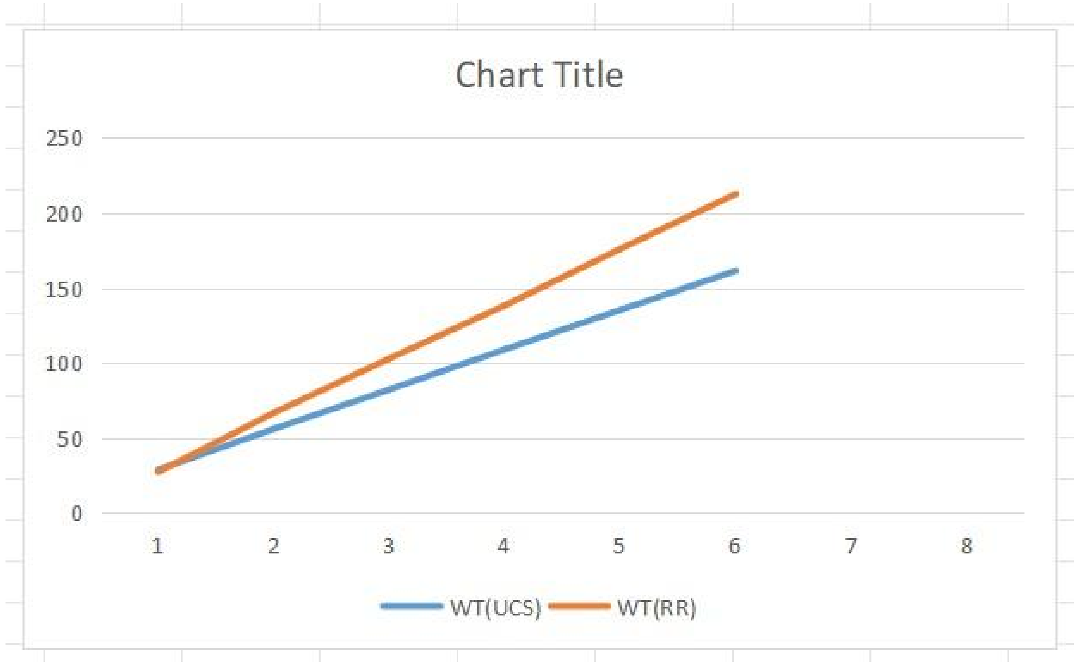
- Then the **‘waiting_time’** is calculated by considering the total time the processes are waiting in the ready queue.
- The **‘turn_around_time’** is calculated by the **‘completion_time’ and ‘arrival_time’.**
- The **‘average_waiting_time’** and the **‘average_turn_around_time’** is calculated.
- Then the csv file is readed and the passed to the function and the output has been recorded.

COMPARISON BETWEEN ROUND ROBIN AND UNIFORM COST SEARCH:

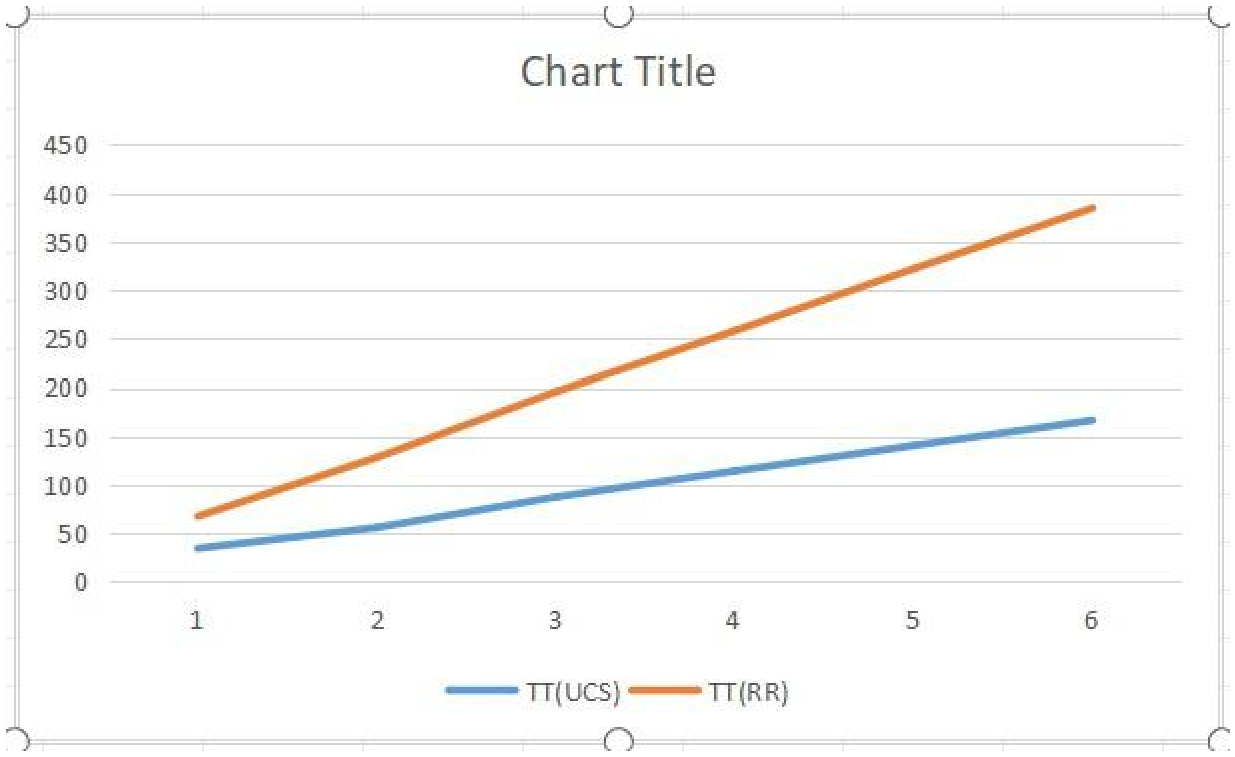
ALGORITHMS	**AVG WAITING TIME**	**AVG TT TIME**	**EXECUTION TIME (s)**
ROUND ROBIN	**212.61667**	**212.00**	**0.002980**
UCS	**161.45**	**166.78**	**0.0011930**

Avg TT Time- Average Turnaround time.

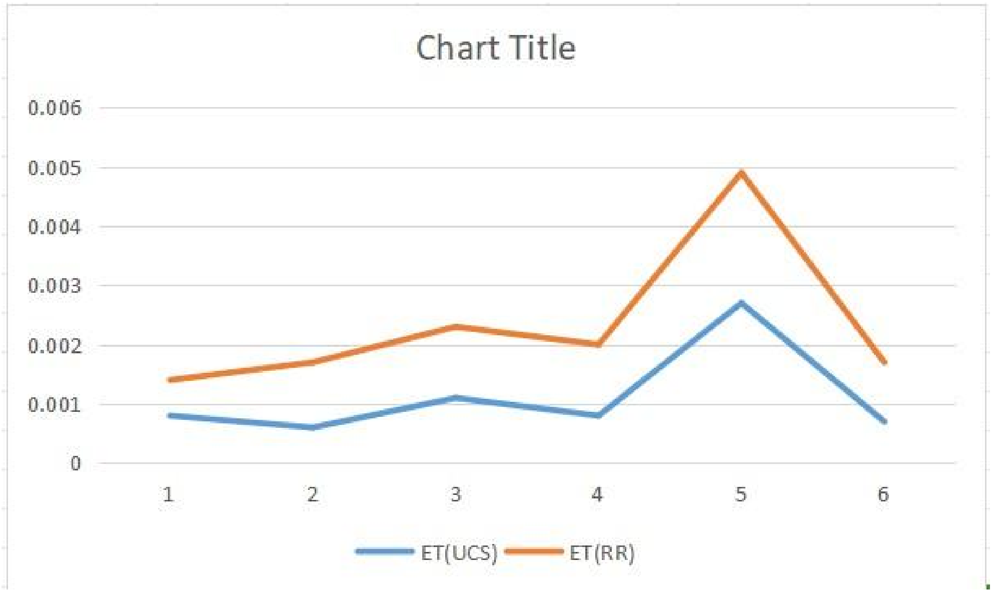
COMPARISON GRAPH FOR AVERAGE WAITING BETWEEN RR AND UCS:



COMPARISON GRAPH FOR AVERAGE TURNAROUND TIME BETWEEN UCS AND RR:



COMPARISON GRAPH FOR EXECUTION TIME BETWEEN UCS AND RR:



INTERPRETATION FROM THE TABLE AND GRAPHS:

- If we analyze the output of the above two algorithms, we can infer that the **average waiting time given by Round Robin is 212.8333 and average waiting time given by UCS is 161.83333.**
- **It is clear that the average waiting time given by UCS is lesser than Round Robin.**
- If we analyze the **average turnaround time value for Round Robin it is 212.00** and for **UCS it is 167.20**
- **So, It is clear that the average turnaround time value for UCS is lesser than Round Robin.**
- If we analyze the **execution time for Round Robin it is 0.00211** and for **UCS it is 0.00085.**
- **It is clear that the execution time for UCS is lesser than Round Robin**

CONCLUSION:

Since the **Average waiting time and Execution time of Uniform Cost Search (UCS) is LESSER than Round Robin,** we can use **UCS Algorithm for Process Scheduling.**

Thank You!