def FCFO(proc,nomber,burst):

# Function to find the waiting

# time for all processes

def findWaitingTime(processes, n,bt, wt):

# waiting time for first process is 0

wt[0] = 0

# calculating waiting time

for i in range(1, n ):

wt[i] = bt[i - 1] + wt[i - 1]

# Function to calculate turnaround time

def findTurnAroundTime(processes, n,bt, wt, tat):

# calculating turnaround

# time by adding bt[i] + wt[i]

for i in range(n):

tat[i] = bt[i] + wt[i]

# Function to calculate average time

def findavgTime( processes, n, bt):

wt = [0] \* n

tat = [0] \* n

total\_wt = 0

total\_tat = 0

# Function to find waiting time of all processes

findWaitingTime(processes, n, bt, wt)

# Function to find turn around

# time for all processes

findTurnAroundTime(processes, n,

bt, wt, tat)

# Display processes along

# with all details

print( "Processes Burst time " +

" Waiting time " +

" Turn around time")

# Calculate total waiting time

# and total turn around time

for i in range(n):

total\_wt = total\_wt + wt[i]

total\_tat = total\_tat + tat[i]

print(" " + str(i + 1) + "\t\t" +

str(bt[i]) + "\t " +

str(wt[i]) + "\t\t " +

str(tat[i]))

print( "Average waiting time = "+

str(total\_wt / n))

print("Average turn around time = "+

str(total\_tat / n))

findavgTime(proc,nomber,burst)

def Round\_Robin(process,n,bt,quantum):

# Function to find the waiting time for all processes

def findWaitingTime(processes, n, bt,

wt, quantum):

rem\_bt = [0] \* n

# Copy the burst time into rt[]

rem\_bt = bt.copy()

t = 0 # Current time

# Keep traversing processes in round

# robin manner until all of them are

# not done.

while(1):

done = True

# Traverse all processes one by

# one repeatedly

for i in range(n):

# If burst time of a process is greater

# than 0 then only need to process further

if (rem\_bt[i] > 0) :

done = False # There is a pending process

if (rem\_bt[i] > quantum) :

# Increase the value of t i.e. shows

# how much time a process has been processed

t += quantum

# Decrease the burst\_time of current

# process by quantum

rem\_bt[i] -= quantum

# If burst time is smaller than or equal

# to quantum. Last cycle for this process

else:

# Increase the value of t i.e. shows

# how much time a process has been processed

t = t + rem\_bt[i]

# Waiting time is current time minus

# time used by this process

wt[i] = t - bt[i]

# As the process gets fully executed

# make its remaining burst time = 0

rem\_bt[i] = 0

# If all processes are done

if (done == True):

break

# Function to calculate turn around time

def findTurnAroundTime(processes, n, bt, wt, tat):

# Calculating turnaround time

for i in range(n):

tat[i] = bt[i] + wt[i]

# Function to calculate average waiting

# and turn-around times.

def findavgTime(processes, n, bt, quantum):

wt = [0] \* n

tat = [0] \* n

# Function to find waiting time

# of all processes

findWaitingTime(processes, n, bt, wt, quantum)

# Function to find turn around time

# for all processes

findTurnAroundTime(processes, n, bt,wt, tat)

# Display processes along with all details

print("Processes Burst Time Waiting",

"Time Turn-Around Time")

total\_wt = 0

total\_tat = 0

for i in range(n):

total\_wt = total\_wt + wt[i]

total\_tat = total\_tat + tat[i]

print(" ", i + 1, "\t\t", bt[i],

"\t\t", wt[i], "\t\t", tat[i])

print("\nAverage waiting time = %.15f "%(total\_wt /n) )

print("Average turn around time = %.15f "% (total\_tat / n))

findavgTime(process,n,bt,quantum)

def SJF(processes,n,burst\_time,choice = 1):

arrival\_tme = []

time\_slice = burst\_time.copy()

Ttime = [0] \* n

time = 0

waiting = [0] \* n

for i in range (n):

arrival\_tme.append(int(input("enter the arrival time of the job")))

b = sorted(time\_slice)

c = sorted(arrival\_tme)

print("0:")

def min\_time\_slice(time\_slice,arr\_time,c):

a = []

for i in range (len(time\_slice)):

if arr\_time[i] < c:

a.append(arr\_time[i])

return min(a)

if choice == 1:

for i in range (n):

index = arrival\_tme.index(c[0])

min\_time = min\_time\_slice(time\_slice,arrival\_tme,c[0])

c.pop(0)

b.pop(b.index(time\_slice[index])) #pop out the corresponding element out of sorted time slice array

time += min\_time

time\_slice.pop(index)

arrival\_tme.pop(index)

if i > 1:

waiting[i] = time

Ttime = waiting[i] + time

m = min\_time\_slice(time\_slice,arrival\_tme,time)

index = c.index(arrival\_tme[time\_slice.index(m)])

c.sort()

c[0],c[index] = c[index],c[0]

# index = time\_slice.index(b[0])

# c.remove(b[0])

# time += b.pop(0)

# time\_slice.pop(index)

# arrival\_tme.pop(index)

# l = [time\_slice[arrival\_tme.index(x)] for x in c[0:time+1] ]

# b = sorted(l)

# print(time,":",end= " ")

for i in range(n):

total\_wt = total\_wt + waiting[i]

total\_tat = total\_tat + Ttime[i]

print(" ", processes[i][0], "\t\t",

processes[i][1], "\t\t",

waiting[i], "\t\t", Ttime[i])

print("\nAverage waiting time = %.5f " % (total\_wt / n))

print("Average turn around time = %.7f", total\_tat / n)

# elif choice == 2:

# # Function to find the waiting time

# # for all processes

# def findWaitingTime(processes, n, wt):

# rt = [0] \* n

#

# # Copy the burst time into rt[]

# for i in range(n):

# rt[i] = processes[i][1]

# complete = 0

# t = 0

# minm = 999999999

# short = 0

# check = False

#

# # Process until all processes gets

# # completed

# while (complete != n):

#

# # Find process with minimum remaining

# # time among the processes that

# # arrives till the current time`

# for j in range(n):

# if ((processes[j][2] <= t) and

# (rt[j] < minm) and rt[j] > 0):

# minm = rt[j]

# short = j

# check = True

# if (check == False):

# t += 1

# continue

#

# # Reduce remaining time by one

# rt[short] -= 1

#

# # Update minimum

# minm = rt[short]

# if (minm == 0):

# minm = 999999999

#

# # If a process gets completely

# # executed

# if (rt[short] == 0):

#

# # Increment complete

# complete += 1

# check = False

#

# # Find finish time of current

# # process

# fint = t + 1

#

# # Calculate waiting time

# wt[short] = (fint - proc[short][1] -

# proc[short][2])

#

# if (wt[short] < 0):

# wt[short] = 0

#

# # Increment time

# t += 1

#

# # Function to calculate turn around time

# def findTurnAroundTime(processes, n, wt, tat):

#

# # Calculating turnaround time

# for i in range(n):

# tat[i] = processes[i][1] + wt[i]

#

# # Function to calculate average waiting

#

# # and turn-around times.

# def findavgTime(processes, n):

# wt = [0] \* n

# tat = [0] \* n

#

# # Function to find waiting time

# # of all processes

# findWaitingTime(processes, n, wt)

#

# # Function to find turn around time

# # for all processes

# findTurnAroundTime(processes, n, wt, tat)

#

# # Display processes along with all details

# print("Processes Burst Time Waiting",

# "Time Turn-Around Time")

# total\_wt = 0

# total\_tat = 0

# for i in range(n):

# total\_wt = total\_wt + wt[i]

# total\_tat = total\_tat + tat[i]

# print(" ", processes[i][0], "\t\t",

# processes[i][1], "\t\t",

# wt[i], "\t\t", tat[i])

#

# print("\nAverage waiting time = %.5f " % (total\_wt / n))

# print("Average turn around time = ", total\_tat / n)

# findavgTime(processes,n)

else:

print("Wrong input")

if \_\_name\_\_ == "\_\_main\_\_":

n = int(input("Enter the number of processes"))

processes = []

burst\_time = []

for item in range (n):

burst\_time.append(int(input("Enter the burst time of process")))

processes.append(item)

print("enter 1 for FCFS \n enter 3 for SJF \n enter 2 for Round Robin ")

k = int(input())

if k==1:

FCFO(processes,n,burst\_time)

elif(k==2):

qua = int(input("Enter the time slice"))

Round\_Robin(processes,n,burst\_time,qua)

elif(k==3):

# choice = int(input("enter 2 if you want preemptive SJF"))

SJF(processes,n,burst\_time)

else:

print("wrong input")