Name : Sourabh Laxman Rananaware.

Reg_No : 11800322

Email address : sr.sourabh98@gmail.com

GitHub Link : https://github.com/srsourabh/Oprating-System.git

---

Question :

Consider a scheduling approach which is non pre-emptive similar to shortest job next in nature. The priority of each job is dependent on its estimated run time, and also the amount of time it has spent waiting. Jobs gain higher priority the longer they wait, which prevents indefinite postponement. The jobs that have spent a long time waiting compete against those estimated to have short run times. The priority can be computed as :

Priority = 1+ Waiting time / Estimated run time

Using the data given below compute the waiting time and turnaround time for each process and average waiting time and average turnaround time.

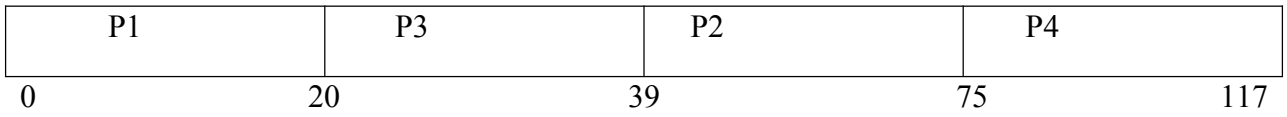
Process	Arrival time	Burst time
P_1	0	20
P_2	5	36
P_3	13	19
P_4	17	42

Solution :

Description :

In this problem we have given number of processes, Arrival time and Burst time. This problem contains the processes which are non pre-emptive. Non pre-emptive means the process cannot terminate until it get completed or no other process can run intrupting this process. We have to solve this question by using shortest job next method. We have to consider burst time for solve this. But in this question there are certain conditions like 1. Priority of each job is dependent on

its estimated run time and also the amount of time it has spent waiting. Means we have to calculate priority by estimated run time.2. Jobs gain higher priority longer they wait. Means higher priority process will run at last.



Processes	Arriving Time	Burst Time	Completion Time	Turn-Around Time	Waiting Time	Priority
P1	0	20	20	20	0	1
P2	5	36	75	70	34	1.94
P3	13	19	39	26	7	1.36
P4	17	42	117	100	58	2.38

Average Waiting Time = 0+34+7+58/4

=24.75

Average Turn-Around Time = 20+70+26+100/54

= 54

Algorithm :

1. Start
2. Create structure
3. declare variables
4. compare schedule1 and schedule2
5. read value of number of processes
6. read values of process id, arriving time and burst time.
7. Initialize for loop and compare ->sort(pro+i,pro+n,compare2);
   a. if(pro[i-1].ct<pro[i].at) then return : pro[i].ct;

else return : pro[i].ct=pro[i-1].ct+pro[i].bt;

8.then print avg turnaround time and avg waiting time.

9.end.

Complexity :

- In min heap, each process will be added and deleted exactly once.
- Adding an element takes log(n) time and deleting an element takes log(n) time.
- Thus, for n processes, time complexity = n x 2log(n) = nlog(n)

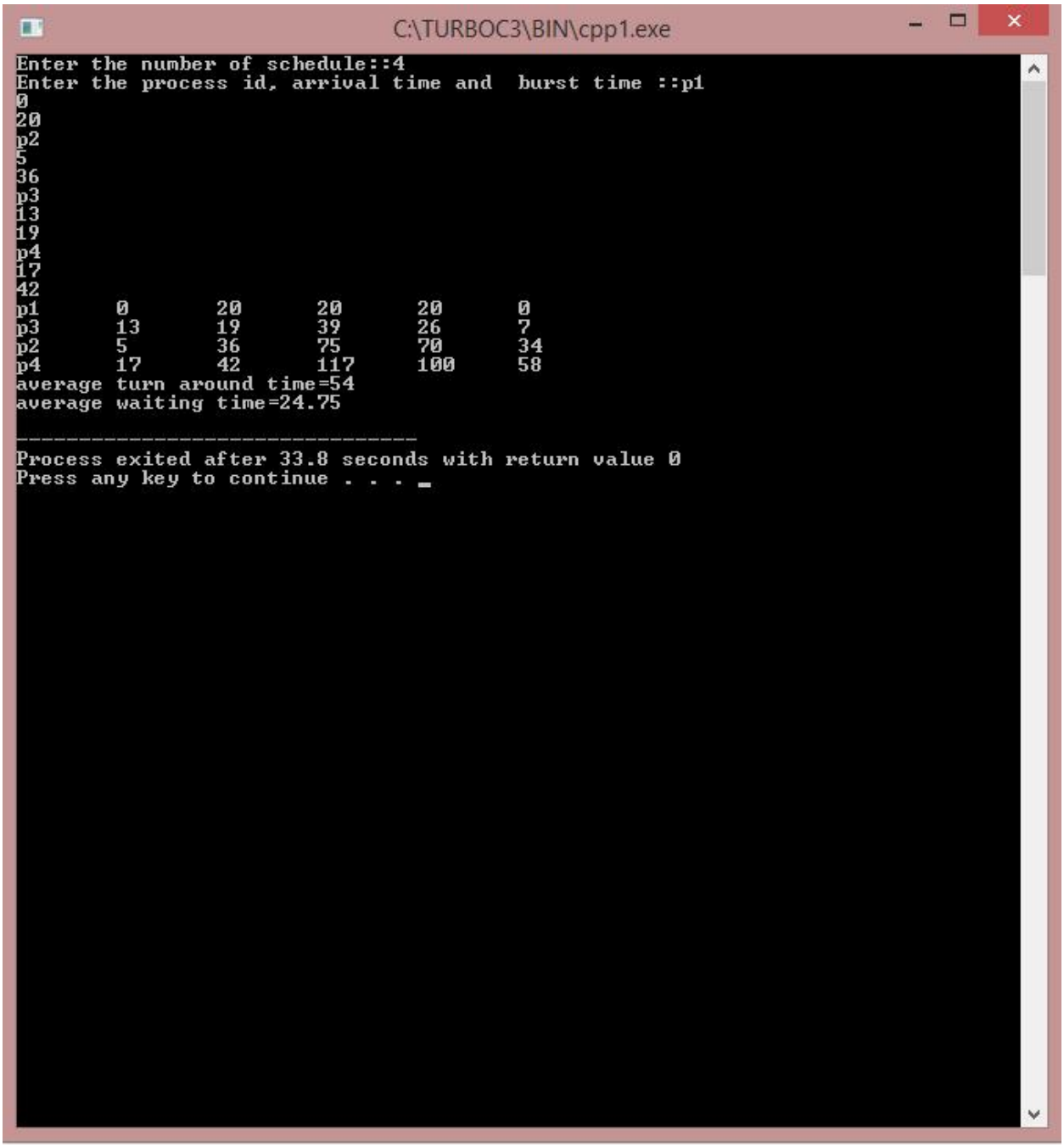
Code Snipet :

```
#include <iostream>
#include<bits/stdc++.h>
#include <algorithm.>
using namespace std;
int ab;
typedef struct schedule
{
  string pro_id;
  int arr_t,burst_t,comp_t,tat_a,wait_t;

};schedule;

bool compare(schedule a,schedule b)
{
  return a.arr_t < b.arr_t;
}

bool compare2(schedule a,schedule b)
```

```
{
  return a.burst_t < b.burst_t && a.arr_t <= ab;
}
int main()
{
  schedule pro[10];
  int n,i,j;
  cout<<"Enter the number of schedule::";
  cin>>n;
  cout<<"Enter the process id, arrival time and  burst time ::";

  for(i=0;i<n;i++)
  {
            cin>>pro[i].pro_id;
            cin>>pro[i].arr_t;
            cin>>pro[i].burst_t;
  }
  sort(pro,pro+n,compare);
  pro[0].comp_t=pro[0].burst_t+pro[0].arr_t;
  pro[0].tat_a=pro[0].comp_t-pro[0].arr_t;
  pro[0].wait_t=pro[0].tat_a-pro[0].burst_t;

  for(i=1;i<n;i++)
  {
            ab=pro[i-1].comp_t;
            sort(pro+i,pro+n,compare2);
            if(pro[i-1].comp_t<pro[i].arr_t)
            {
                        pro[i].comp_t=pro[i-1].comp_t+pro[i].burst_t+(pro[i].arr_t-pro[i-1].comp_t);
            }
            else
            {

                        pro[i].comp_t=pro[i-1].comp_t+pro[i].burst_t;
```

```
            }
            pro[i].tat_a=pro[i].comp_t-pro[i].arr_t;
            pro[i].wait_t=pro[i].tat_a-pro[i].burst_t;
}
   float sum=0;
   float sum1=0;
for(i=0;i<n;i++)
{

            cout<<pro[i].pro_id<<"\t"<<pro[i].arr_t<<"\t"<<pro[i].burst_t
            <<"\t"<<pro[i].comp_t<<"\t"<<pro[i].tat_a<<"\t"<<pro[i].wait_t;
            sum=sum+pro[i].tat_a;
            sum1=sum1+pro[i].wait_t;

            cout<<endl;
}
float     avgta=sum/n;
cout<<"average turn around time="<<avgta<<endl;
float avgwt=sum1/n;
cout<<"average waiting time="<<avgwt<<endl;
return 0;
}
```

Test Cases :


C:\TURBOC3\BIN\cpp1.exe
Enter the number of schedule::4
Enter the process id, arrival time and burst time ::p1
0
20
p2
5
36
p3
13
19
p4
17
42
p1 0 20 20 20 0
p3 13 19 39 26 7
p2 5 36 75 70 34
p4 17 42 117 100 58
average turn around time=54
average waiting time=24.75
Process exited after 33.8 seconds with return value 0
Press any key to continue . . .