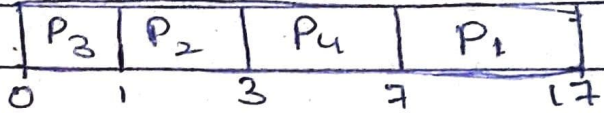


Qns 2:

Process	Arrival time	Burst time
P <sub>1</sub>	0	10
P <sub>2</sub>	0	2
P <sub>3</sub>	0	1
P <sub>4</sub>	0	4

Turn around time = ?, waiting time = ?, avg T.T = ?,  
avg W.T = ?

Turn around time = CT - AT (CT = Completion time)  
WT = TAT - BT



Process	AT	BT	CT	TAT	WT
P <sub>1</sub>	0	10	17	17	7
P <sub>2</sub>	0	2	3	3	1
P <sub>3</sub>	0	1	1	1	0
P <sub>4</sub>	0	4	7	7	3

avg TAT =  $\frac{28}{4}$  7 // ans

avg WT =  $\frac{11}{4}$  2.75 // ans.

```
1 #include<stdio.h>
2 int main()
3 {
4     int
5     bt[20],p[20],wt[20],tat[20],i,j,n=4,tot
6     tal=0,pos,temp;
7     float avg_wt,avg_tat;
8     printf("\nEnter Burst Time:\n");
9     for(i=0;i<n;i++)
10     {
11         printf("p%d:",i+1);
12         scanf("%d",&bt[i]);
13         p[i]=i+1;
14     }
15     for(i=0;i<n;i++)
16     {
17         pos=i;
18         for(j=i+1;j<n;j++)
19         {
20             if(bt[j]<bt[pos])
21                 pos=j;
22         }
23         temp=bt[i];
24         bt[i]=bt[pos];
25         bt[pos]=temp;
26         temp=p[i];
27         p[i]=p[pos];
28         p[pos]=temp;
29     }
30     wt[0]=0;
31     for(i=1;i<n;i++)
32     {
33         wt[i]=0;
34         for(j=0;j<i;j++)
35             wt[i]+=bt[j];
```



```

19         if(bt[j]<bt[pos])
20             pos=j;
21     }
22     temp=bt[i];
23     bt[i]=bt[pos];
24     bt[pos]=temp;
25     temp=p[i];
26     p[i]=p[pos];
27     p[pos]=temp;
28 }
29 wt[0]=0;
30 for(i=1;i<n;i++)
31 {
32     wt[i]=0;
33     for(j=0;j<i;j++)
34         wt[i]+=bt[j];
35     total+=wt[i];
36 }
37 avg_wt=(float)total/n;
38 total=0;
39 printf("\nPro \t BT \tWT
40 \tTAT");
41 for(i=0;i<n;i++)
42 {
43     tat[i]=bt[i]+wt[i];
44     total+=tat[i];
45     printf("\np%d \t %d \t %d
46 \t %d",p[i],bt[i],wt[i],tat[i]);
47 }
48 avg_tat=(float)total/n;
49 printf("\n\nAverage Waiting
50 Time=%f",avg_wt);
51 printf("\nAverage Turnaround
52 Time=%f\n",avg_tat);
53 }

```

Enter Burst Time:

p1:10

p2:2

p3:1

p4:4

Pro	BT	WT	TAT
p3	1	0	1
p2	2	1	3
p4	4	3	7
p1	10	7	17

Average Waiting Time=2.750000

Average Turnaround Time=7.000000

Processes completed process Entered

algorithm

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

```
int main()
```

```
{
```

```
    int bt[20], p[20], wt[20], tat[20], i, j, n, total=0,  
    pos, temp;
```

```
    float avg_wt, avg_tat;
```

```
    printf("Enter number of process:");
```

```
    scanf("%d", &n);
```

```
    printf("\nEnter Burst Time:\n");
```

```
    for (i=0; i<n; i++)
```

```
    {
```

```
        printf("p%d:", i+1);
```

```
        scanf("%d", &bt[i]);
```

```
        p[i] = i+1;
```

```
    }
```

```
// sorting of burst times
```

```
for (i=0; i<n; i++)
```

```
{
```

```
    pos = i;
```

```
    for (j=i+1; j<n; j++)
```

```
    {
```

```
        if (bt[j] < bt[pos])
```

```
            pos = j;
```

```
    }
```

```
    temp = bt[i];
```

```
    bt[i] = bt[pos];
```

```
    bt[pos] = temp;
```

```
    temp = p[i];
```

```
    p[i] = p[pos];
```

```
    p[pos] = temp;
```

```
}
```

```
wt[0] = 0;
```

```
for (i=1; i<n; i++)
```

```
{
```

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



```

    p[i] = p[pos];
    p[pos] = temp;
}

wt[i]

for (i = 1; i < n; i++)
{
    wt[i] = 0;
    for (j = 0; j < i; j++)
        wt[i] += bt[j];
    total += wt[i];
}

```

```

avg-wt = (float) total/n;
total = 0;
printf("Process\t Burst Time \t Waiting Time\t\n\t\t\t Turnaround Time");

for (i = 0; i < n; i++)
{
    tat[i] = bt[i] + wt[i];
    total += tat[i];
    printf("\n p \t\t bt \t\t wt \t\t tat",
           p[i], bt[i], wt[i], tat[i]);
}

avg-tat = (float) total/n;
printf("Average Waiting Time = %.f", avg-wt);
printf("Average Turnaround Time = %.f",
       avg-tat);

return 0;
}

```

```

Ans. #include <stdio.h>
#include <ctype.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdin.h>
#include <stdlib.h>
#include <string.h>

char* headline();
char* ltrim(char*);
char* rtrim(char*);
int parse_int(char*);

int main()
{
    FILE* fptr = fopen(getenv("Output-Path"), "w");
    int n = parse_int(ltrim(rtrim(headline())));
    int** customers = malloc(n * size_of(int*));
    for (int i = 0; i < n; i++)
    {
        *(customers + i) = malloc(2 * (size_of(int)));
        char** customers_item_temp = split_string(rtrim(headline()));
        for (int j = 0; j < 2; j++)
        {
            int customers_item = parse_int(*(customers_item_temp + j));

```

Scanned by TapScanner

```

        (*(customers + i) + j) = customers_item;
    }
    int result = minimum_Average(n, 2, customers);
    fprintf(fptr, "%d\n", result);
    fclose(fptr);
    return 0;
}

char* headline()
{
    size_t alloc_length = 1024;
    size_t data_length = 0;
    char* data = malloc(alloc_length);
    while (true)
    {
        char* cursor = data + data_length;
        char* line = fgets(cursor, alloc_length - data_length, stdin);
        if (!line)

```

```

return 0;
}

char * headline() {
    size_t alloc_length = 1024;
    size_t data_length = 0;
    char * data = malloc(alloc_length);
    while (true) {
        char * cursor = data + data_length;
        char * line = fgets(cursor, alloc_length - data_length, stdin);
        if (!line)
            break;
        if (data_length < alloc_length - 1 || data[data_length - 1] == '\n')
            break;
        alloc_length <<= 1;
        data = realloc(data, alloc_length);
        if (!data) {
            data = '\0';
            break;
        }
        if (data[data_length - 1] == '\n') {
            data[data_length - 1] = '\0';
        }
    }
}

```

```

else {
    data = realloc(data, data_length + 1);
    if (!data) {
        data = '\0';
    } else {
        data[data_length] = '\0';
    }
    return data;
}

char * ltrim(char * str) {
    if (!str)
        return '\0';
    if (!*str)
        return str;
    while (*str == '\0' || isspace(*str))
        str++;
    return str;
}

char * rtrim(char * str) {

```



```

* (end + 1) = '\0';
return str;
}

char ** split_string (char * str)
{
    char ** splits = NULL;
    char * token = strtok (str, " ");

    int spaces = 0;
    while (token) {
        splits = realloc (splits, size of (char *) * ++ spaces);
        if (! splits) {
            return splits;
        }
        splits [spaces - 1] = token;
        token = strtok (NULL, " ");
    }
    return splits;
}

int parse - int (char * str) {
    char * end_ptr;
    int value = strtol (str, & end_ptr, 10);
    if (end_ptr == str || end_ptr != '\0')
        exit (EXIT_FAILURE);
    return value;
}

```