Name :- Pallvi
Course :- Bsc IT (II)
Section :- B
Subject :- Operating system

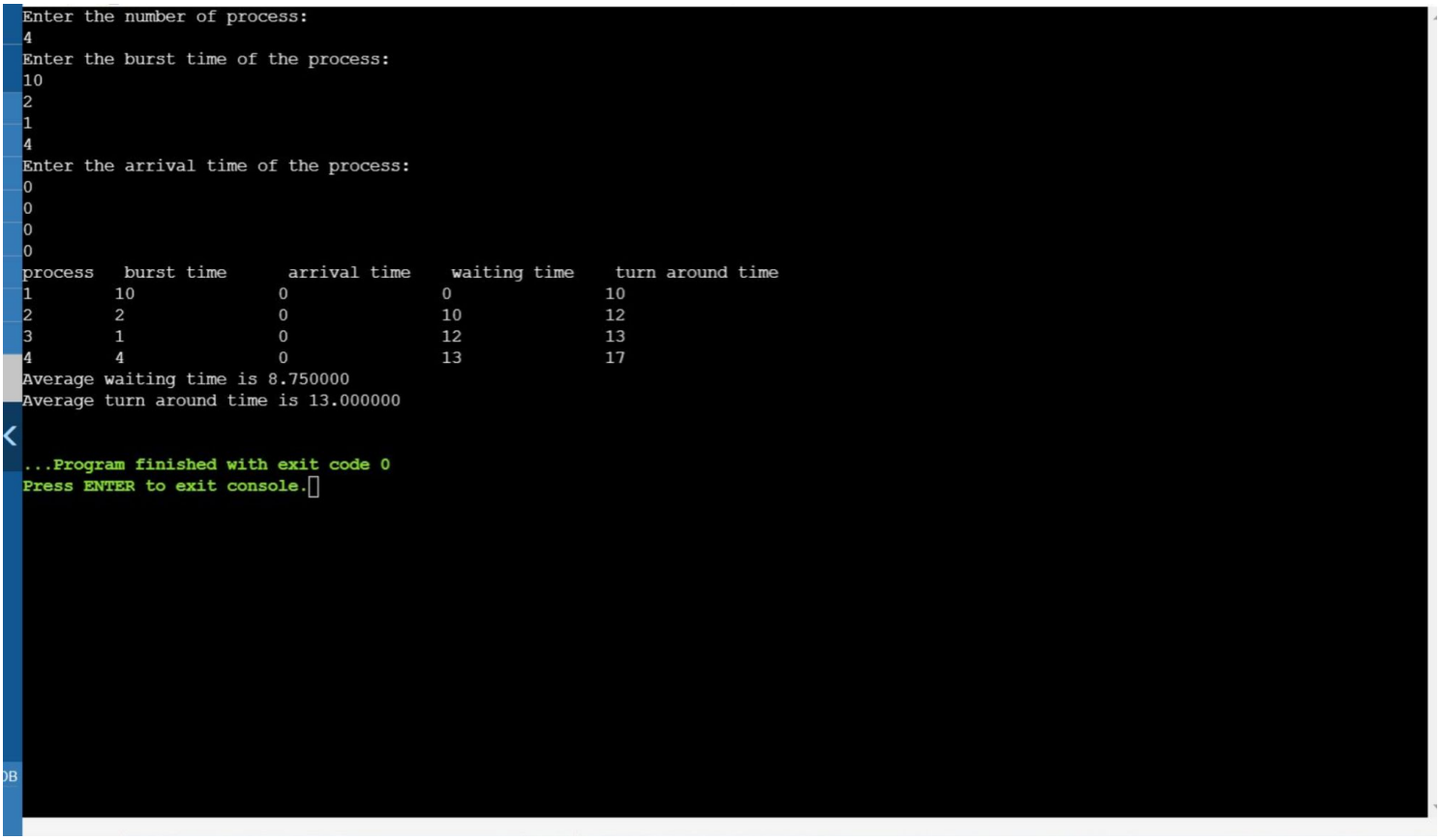
Problem statement 2 :-

C program :-

```
# include < stdio . h >
# include < conio . h >
# define max 30
void main()
{
    int i, j, n, t, p[max], bt[max],
    wt[max], tat[max];
    float awt = 0, atat = 0;
    clrscr();
    printf("Enter the number of process");
    scanf("%d", &n);
    printf("Enter the process number");
    for(i=0; i<n; i++
    {
      scanf("%d", &p[i]);
    }
    printf("Enter the burst time of the
    process");
    for(i=0; i<n; i++)
    {
      scanf("%d", &bt[i]);
    }
```

```
{ for (i=0 ; i < n ; i++)
  for (j=0; j < n-i-1 ; j++)
  {
    if (bt [j] > bt [j+1])
    {
      t = bt [j];
      bt [j] = bt [j+1];
      bt [j+1] = t ;

      t = p [j];
      p [j] = p [j+1];
      p [j+1] = t;
    }
  }
}
printf (" process \t burst time \t waiting
 time \t turnaround time \n");
for (i=0 ; i <n; i++)
{
  wt [i] = 0;
  tat [i] = 0;
  for (j=0; j<i; j++)
  {
    wt [i] = wt [i] + bt [j] ;
  }
  tat [i] = wt [i] + bt [i];
  awt = awt + wt [i];
  atat = atat + tat [i];
  printf (" %d \t %d \t\t %d \t \t
    %d \n", p [i] , bt [i] , wt [i] ,
      tat [i]);
```


Enter the number of process:
4
Enter the burst time of the process:
10
2
1
4
Enter the arrival time of the process:
0
0
0
0
process burst time arrival time waiting time turn around time
1 10 0 0 10
2 2 0 10 12
3 1 0 12 13
4 4 0 13 17
Average waiting time is 8.750000
Average turn around time is 13.000000
...Program finished with exit code 0
Press ENTER to exit console.

```
    getch();
}
```

Problem Statement 2 :-

```
#include <assert.h>
#include <ctype.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>

char* readline();
char* ltrim(char*);
char* rtrim(char*);
char** split_string(char*);

int parse_int(char*);
/*
* Complete the 'minimumAverage' function
below.
*
* The function is expected to return an
Integer
* The function accepts
2D_Integer_Array customers as parameter.
*/
int minimumAverage(int customers_rows,
int customers_columns, int** customers)
{
}
```

```
int main()
{
    FILE * fptr =
    fopen(getenv("Output - Path"), "w");

    int n = parse_int(ltrim(rtrim(readline
                                  ())));

    int ** customers = malloc(n * sizeof(int*));

    for (int i = 0; i < n; i++) {
        *(customers + i) = malloc(2 * (sizeof(int)));

    char ** customers_item_temp = split_
    string(rtrim(readline()));

    for (int j = 0; j < 2; j++) {
        int customers_item =
    parse_int(*(customers_item_temp + j));
    (*(customers + i) + j) =
        customers_item;
    }
    }
    int result = minimum Average(n, 2,
    customers);
    fprintf(fptr, "%d\n", result);
    fclose(fptr);
    return 0;
}
char * readline() {
    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;
}
data_length += strlen (cursor);
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';
   data = realloc (data, data_length);
   if (!data) {
     data = '\0';
    }
   } else {
     data = realloc (data, data_length + 1);
   if (! data) {
      data = '\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 ) {
    if (! str ) {
      return '\0';
    }
    if (! * str ) {
      return str ;
}
  char * end = str + strlen (str) - 1;
while ( end >= str && isspace [ * end ])
  {
    end--;
  }
```

```
* (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, sizeof (char *) * ++
  + spaces);
  if (! splits) {
   return splits;
   }
   splits [spaces - 1] = token;
    token = strtok ( NULL, " ");
  }
  return splits;
}
int parse_int (char * str){
 char * endptr;
  int value = strol ( str, & endptr, 10);
  if (endptr == str || * endptr != '\0')
    {
      exit (EXIT_FAILURE);
    }
 return value;
}
```