# SJF (Shortest Job first)

```
#include<stdio.h>
#include<stdbool.h>
#include<limits.h>
struct process_struct
{
  int pid;
  int at;
  int bt;
  int ct,wt,tat,rt,start_time;
}ps[100];
int findmax(int a, int b)
{
   return a>b?a:b;
}
int findmin(int a, int b)
{
   return a<b?a:b;
}
int main()
{
   int n;
   bool is_completed[100]={false},is_first_process=true;
   int current_time = 0;
   int completed = 0;;
   printf("Enter total number of processes: ");
   scanf("%d",&n);
   int sum_tat=0,sum_wt=0,sum_rt=0,total_idle_time=0,prev=0,length_cycle;
```

```
  float cpu_utilization;
  int max_completion_time,min_arrival_time;
for(int i=0;i<n;i++)
  {
    printf("\nEnter Process %d Arrival Time: ",i);
    scanf("%d",&ps[i].at);
    ps[i].pid = i ;
  }
  for(int i=0;i<n;i++)
  {
    printf("\nEnter Process %d Burst Time: ",i);
    scanf("%d",&ps[i].bt);
  }
  while(completed!=n)
  {
    //find process with min. burst time in ready queue at current time
    int min_index = -1;
    int minimum = INT_MAX;
    for(int i = 0; i < n; i++) {
      if(ps[i].at <= current_time && is_completed[i] == false) {
        if(ps[i].bt < minimum) {
          minimum = ps[i].bt;
          min_index = i;
        }
        if(ps[i].bt== minimum) {
          if(ps[i].at < ps[min_index].at) {
            minimum= ps[i].bt;
            min_index = i;
          }
```

```
        }
      }
    }

    if(min_index==-1)
    {
       current_time++;
    }
    else
    {
    ps[min_index].start_time = current_time;
    ps[min_index].ct = ps[min_index].start_time + ps[min_index].bt;
    ps[min_index].tat = ps[min_index].ct - ps[min_index].at;
    ps[min_index].wt = ps[min_index].tat - ps[min_index].bt;
    ps[min_index].rt = ps[min_index].wt;
    // ps[min_index].rt = ps[min_index].start_time - ps[min_index].at;

    sum_tat +=ps[min_index].tat;
    sum_wt += ps[min_index].wt;
    sum_rt += ps[min_index].rt;
    total_idle_time += (is_first_process==true) ? 0 : (ps[min_index].start_time -  prev);

    completed++;
    is_completed[min_index]=true;
    current_time = ps[min_index].ct;
    prev= current_time;
    is_first_process = false;
    }
}
```

```
    //Calculate Length of Process completion cycle
    max_completion_time = INT_MIN;
    min_arrival_time = INT_MAX;
    for(int i=0;i<n;i++)
    {
       max_completion_time = findmax(max_completion_time,ps[i].ct);
       min_arrival_time = findmin(min_arrival_time,ps[i].at);
    }
    length_cycle = max_completion_time - min_arrival_time;

    //Output
    printf("\nProcess No.\tAT\tCPU Burst Time\tCT\tTAT\tWT\tRT\n");
    for(int i=0;i<n;i++)

printf("%d\t\t%d\t%d\t\t%d\t%d\t%d\t%d\n",ps[i].pid,ps[i].at,ps[i].bt,ps[i].ct,ps[i].tat,ps[i].wt
,ps[i].rt);

    printf("\n");

    cpu_utilization = (float)(length_cycle - total_idle_time)/ length_cycle;

    printf("\nAverage Turn Around time= %f ",(float)sum_tat/n);
    printf("\nAverage Waiting Time= %f ",(float)sum_wt/n);
    printf("\nAverage Response Time= %f ",(float)sum_rt/n);
    printf("\nThroughput= %f",n/(float)length_cycle);
    printf("\nCPU Utilization(Percentage)= %f",cpu_utilization*100);
    return 0;
}
```

# SRTF (Preemptive Algorithm)

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

struct process_struct
{
  int pid;
  int at;
  int bt;
  int ct,wt,tat,rt,start_time;
}ps[100];

int findmax(int a, int b)
{
   return a>b?a:b;
}

int findmin(int a, int b)
{
   return a<b?a:b;
}

int main()
{

   int n;
   float bt_remaining[100];
   bool is_completed[100]={false},is_first_process=true;
```

```
int current_time = 0;
int completed = 0;;
float sum_tat=0,sum_wt=0,sum_rt=0,total_idle_time=0,length_cycle,prev=0;
float cpu_utilization;

int max_completion_time,min_arrival_time;

printf("Enter total number of processes: ");
scanf("%d",&n);
for(int i=0;i<n;i++)
{
   printf("\nEnter Process %d Arrival Time: ",i);
   scanf("%d",&ps[i].at);
   ps[i].pid = i ;
}

for(int i=0;i<n;i++)
{
   printf("\nEnter Process %d Burst Time: ",i);
   scanf("%d",&ps[i].bt);
   bt_remaining[i]= ps[i].bt;
}

while(completed!=n)
{
   //find process with min. burst time in ready queue at current time
   int min_index = -1;
   int minimum = INT_MAX;
   for(int i = 0; i < n; i++) {
```

```
        if(ps[i].at <= current_time && is_completed[i] == false) {
            if(bt_remaining[i] < minimum) {
                minimum = bt_remaining[i];
                min_index = i;
            }
            if(bt_remaining[i]== minimum) {
                if(ps[i].at < ps[min_index].at) {
                    minimum= bt_remaining[i];
                    min_index = i;
                }
            }
        }
    }


    if(min_index==-1)
    {
        current_time++;
    }
    else
    {
        if(bt_remaining[min_index] == ps[min_index].bt)
        {
                ps[min_index].start_time = current_time;
                total_idle_time += (is_first_process==true) ? 0 : (ps[min_index].start_time -
prev);
                is_first_process=false;
        }
        bt_remaining[min_index] -= 1;
```

```
            current_time++;
            prev=current_time;
            if(bt_remaining[min_index] == 0)
            {
                ps[min_index].ct = current_time;
                ps[min_index].tat = ps[min_index].ct - ps[min_index].at;
                ps[min_index].wt= ps[min_index].tat - ps[min_index].bt;
                ps[min_index].rt = ps[min_index].start_time - ps[min_index].at;



                sum_tat +=ps[min_index].tat;
                sum_wt += ps[min_index].wt;
                sum_rt += ps[min_index].rt;
                completed++;
                is_completed[min_index]=true;
                //total_idle_time += (is_first_process==true) ? 0 : (ps[min_index].start_time -
prev);
               // prev= ps[min_index].ct; // or current_time;
            }
        }
    }
    //Calculate Length of Process completion cycle
    max_completion_time = INT_MIN;
    min_arrival_time = INT_MAX;
    for(int i=0;i<n;i++)
    {
        max_completion_time = findmax(max_completion_time,ps[i].ct);
        min_arrival_time = findmin(min_arrival_time,ps[i].at);
    }
```

```
    length_cycle = max_completion_time - min_arrival_time;

    //Output
    printf("\nProcess No.\tAT\tCPU Burst Time\tCT\tTAT\tWT\tRT\n");
    for(int i=0;i<n;i++)

printf("%d\t\t%d\t%d\t\t%d\t%d\t%d\t%d\n",ps[i].pid,ps[i].at,ps[i].bt,ps[i].ct,ps[i].tat,ps[i].wt
,ps[i].rt);

    printf("\n");

    cpu_utilization = (float)(length_cycle - total_idle_time)/ length_cycle;

    printf("\nAverage Turn Around time= %f ",(float)sum_tat/n);
    printf("\nAverage Waiting Time= %f ",(float)sum_wt/n);
    printf("\nAverage Response Time= %f ",(float)sum_rt/n);
    printf("\nThroughput= %f",n/(float)length_cycle);
    printf("\nCPU Utilization(Percentage)= %f",cpu_utilization*100);
    return 0;
}
```