Program for shortest Job first (or SJF)

```
#include <iostream>
using namespace std;

int main() {

    int A[100][4];
    int i, j, n, total = 0, index, temp;
    float avg_wt, avg_tat;

    cout << "Enter number of process: ";
    cin >> n;

    cout << "Enter Burst Time and allotting
    Process Id.
    // User input Burst Time and allotting
    Process Id
    for (i = 0; i < n; i++) {
        cout << "P " << i + 1 << ": ";
        cin >> A[i][1];
        A[i][0] = i + 1;
    }

    // sorting process according to their
    Burst Time
    for (i = 0; i < n; i++) {
```

Scanned with ACE Scanner

Program for shortest Job first (or SJF)

```
#include <iostream>
using namespace std;
int main() {
    int A[100][4];
    int i, j, n, total = 0, index, temp;
    float avg_wt, avg_tat;
    cout << "Enter number of process: ";
    cin >> n;

    cout << "Enter Burst Time and allotting Process Id.";
    // User input Burst Time and allotting Process Id
    for (i = 0; i < n; i++) {
        cout << "P " << i + 1 << ": ";
        cin >> A[i][1];
        A[i][0] = i + 1;
    }

    // sorting process according to their Burst Time
    for (i = 0; i < n; i++) {
```

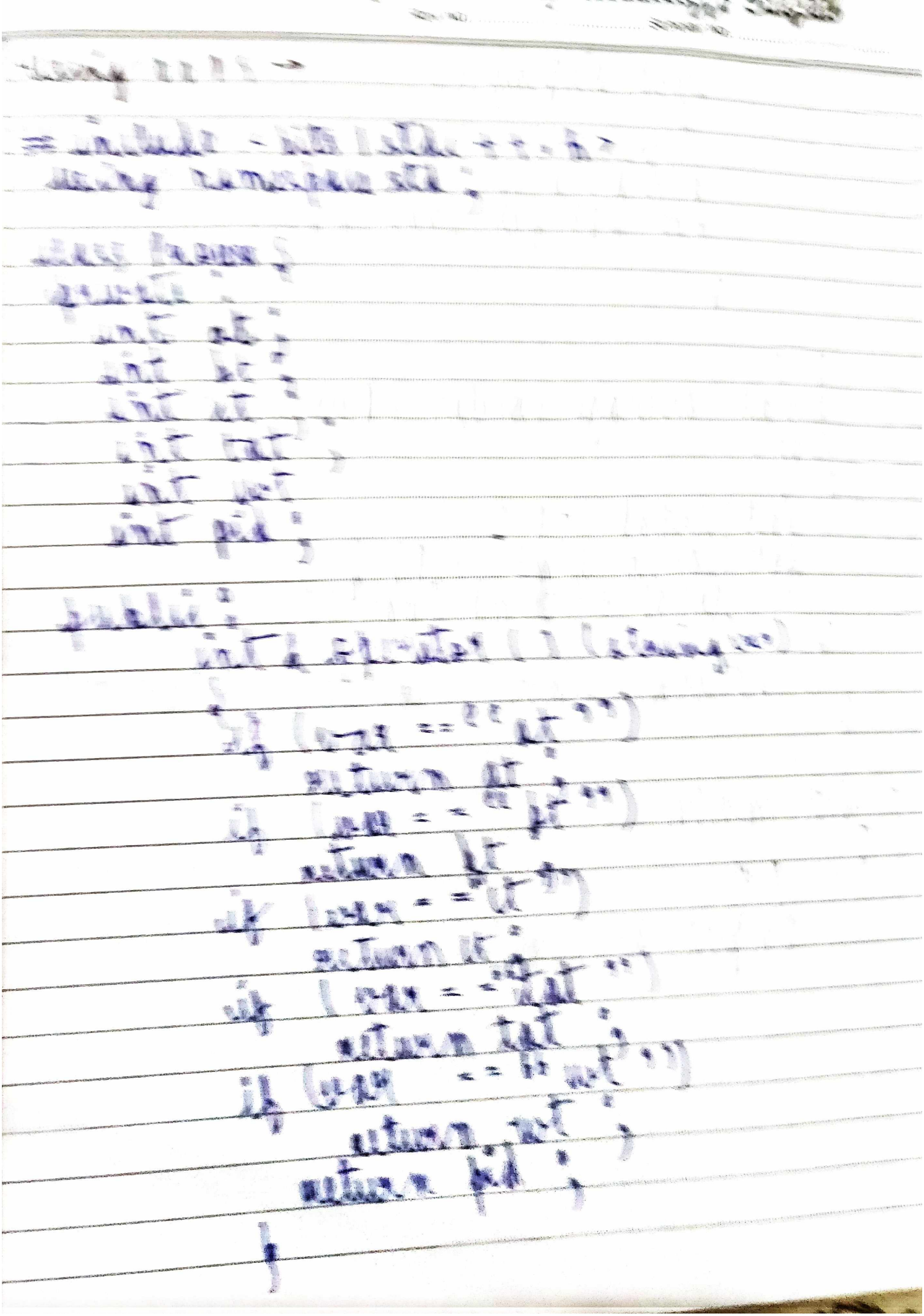
Scanned with ACE Scanner

```
index = i;
for j = (i+1); j < n; j++)
  if A[j][1] < A[index][1])
      index = j;
temp = A[i][1];
A[i][1] = A[index][1];
A[index][1] = temp

temp = A[i][0];
  A[i][0] = A[index][0];
  A[index][0] = temp;
}

A[0][2] = 0;
// calculation of waiting Time
for(i=1; i<n; i++){
  A[i][2] = 0;
  for(j=0; j<i; j++)
    A[i][2] += A[j][1];
  total += A[i][2];
}
avg_wt = (float)total/n;
total = 0;
cout << "\nP  BT  WT  TAT\n";

// data
for(i=0; i<n; i++){
    A[i][3] = A[i][1] + A[i]
```

Scanned with ACE Scanner

```
A[i][3] = A[i][1] + A[i][2];
total += A[i][1] + A[i][2];
cout << "P" << A[i][0] << "\t\t"
A[i][1] << "\t\t" << A[i][2]
<< "\t\t" << A[i][3] << endl;
}

avg_tat = (float) total / n;
cout << "Average waiting Time = "
avg_wt << endl;

for (i = 0; i < n; i++) {
    A[i][3] = A[i][1] + A[i][2];
    total += A[i][3];
    cout << "P" << A[i][0] << "\t\t"
    << A[i][1] << "\t\t" << A[i][2] << "\n";
}

avg_tat = (float) total / n;
cout << "Average waiting Time =
" << avg_wt << endl;
cout << "Average Turnaround Time =
" << avg_tat << endl;

}
```

Scanned with ACE Scanner



Scanned with ACE Scanner

```
void display ()
{
    printf ("%d \t %d \t %d \t %d \t %d \t %d \n", pid, at, bt, it, tat, wt);
}
};
float average (vector <Process> p, string var)
{
    int total = 0;
    for (auto temp : p) {
        total += temp [var];
    }
    return (float) total / p.size ();
}

int main ()
{
    /*
    Input description.
```

Scanned with ACE Scanner

```
int n;
cin >> n;
int counter = 0;
vector <Process> p(n);
for (Process &temp : p) {
    temp["id"] = counter++;
    cin >> temp["at"] >> temp["bt"];
}
sort (p.begin(), p.end(),
    [](Process first, Process second) {
        return first["at"] < second["at"];
    });
printf("\nPid\tat\tbt\tct\ttat\twt
        \n");
p[0]["ct"] = p[0]["at"] + second["at"];
p[0].update_after_it();
p[0].display();
for (int i = 1; i < p.size(); i++) {
    if (p[i]["at"] < p[i-1]["bt"]) {
        p[i]["ct"] = p[i-1]["ct"] +
        p[i]["bt"];
    }
    p[i].update_after_it();
    p[i].display();
}
printf("Average waiting time : %f\n",
```

Scanned with ACE Scanner

```
    average(p, "wt");
    return 0;
}
```

PID	AT	BT	CT	TAT	WT
0	0	10	10	10	0
1	10	5	15	5	0
2	15	8	23	8	0

Avg - turn around : 7.666666

Avg - Waiting time = 0.0

Scanned with ACE Scanner

Roll No. .....

Dining Philosopher —

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

int state[N];
int phil[N] = {0,1,2,3,4};

sem_t mutex;
sem_t S[N];

void test (int phnum)
{
    if (state[phnum] == HUNGRY
    && state[LEFT] != EATING
    && state[RIGHT] != EATING) {

    state[phnum] = EATING;

    sleep(2);

    printf("Philosopher %d takes fork
and %d\n",
        phnum+1, LEFT+1, phnum+1)

    sem_post(&S[phnum]);
                    )
    }
```

Scanned with ACE Scanner

Deadlock Detection →

```
#include <stdio.h>
static int mark[20];
int i, j, np, nr;

int main()
{
int alloc[10][10], request[10], r[10],
w[10];

printf("\n enter the no of the proces
");
scanf("%d", &np);
printf("\n enter the no of resources:
scanf("%d", &nr);
for(i=0; i<nr; i++)
{

printf("\n Total Amount of the
Resource R%d: ", i+1);
scanf("%d", &r[i]);
for(i=0; i<nr; i++)

{
printf("\n Total Amount of the Res
R%d = ", i+1);
scanf("%d", &r[i]);
}
```

Scanned with ACE Scanner

```
for(j = 0 ; j < nr ; j++)
avail[j] = r[j];
for(i=0 ; i < np ; i++)
{
avail[j] = alloc[i][j]
}
}
for(i=0 ; i < np ; i++)
{
int count = 0
for( j=0 , j < nr ; j++)
    if alloc[i][j] == 0)
    count++ ;
    else
      break ;
}
if ( count == nr)
mark[i] = 1 ;
}
for(j=0 ; j < nr ; j++)
  w[j] = avail[j] ;
```

Scanned with ACE Scanner

Roll No.

Priority & Prevention & Non premitive scheduling

```
#include<bits/stdc++.h>
using namespace std;
struct process
{
int at, bt, pr, pno;
};

process proc[50];
{
if(a.at==b.at)
{
return a.pr<b.pr;
}
else
{
return a.at<b.at;

cout<<"\n Process"<<"Burst time"
<<"Waiting time"<<"Turnaround

for(int i=0; i<n; i++)
{
total_wt = total_wt + wt[i];
total_tat = total_tat + tat[i];
cout<<"\t"<<proc[i].pid<<"
}
```

Scanned with ACE Scanner

Scholar No. .....

```
cout << "\n Average waiting time = "
     << (float) total_wt / (float) n;
cout << "\n Average turnaround time = "
     << (float) total_tat / (float) n;
}
void priority_scheduling (process proc[],
int n)
{
  sort (proc, proc+n, sort_processes);
  cout << "\n Order in which process gets
executed \n";
  cout << proc[i].pid << "\n";
  findavgTime (proc, n);
}
// Driver mode
int main()
{
process proc[] = { {1, 10, 2}, {2, 5, 0} };
int n = size of proc / size of proc[0];
priority_scheduling (proc, n);
return 0;
}
```

Scanned with ACE Scanner

Robin Round →
C++ program for implementation of RR scheduling →

```
#include <iostream>
using namespace std;

// function to find the waiting time for all
// processes
void findWaitingTime(int processes[], int n,
    int bt[], int wt[], int quantum)
{
    // Make a copy of burst times bt[] to store remaining
    // burst time
    int rem_bt[n];
    for (int i = 0; i < n; i++)
        rem_bt[i] = bt[i];

    int t = 0; // current time

// Keep traversing processes in round robin manner
// until all of them are not done.
while (1)
{
    bool done = true;
```

Scanned with ACE Scanner

```
// Traverse all processes one by one repeatedly.
for (int i = 0; i < n; i++)

// If burst time of a processes one by one repeatedly.
for (int i = 0; i < n; i++)
{
    bool done = true;
    // Traverse all processes one by one repeatedly.
    for (int i = 0; i < n; i++)

        // 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 pro

            if (rem_bt[i] > quantum)
            {
            // Increase the value of t, i.e. s
            // how much time a processes has been processed.
            t = t + rem_bt[i];
```

Scanned with ACE Scanner

```
// Traverse all processes one by one
repeatedly.
for (int i = 0 ; i < n ; i++)

// If burst time of a processes one
by one repeatedly.
for (int i = 0 ; i < n ; i++)
{
    bool done = true ;
    // Traverse all processes one by
    one repeatedly.
    for (int i = 0 ; i < n ; i++)

        // 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 pro

            if (rem_bt[i] > quantum
            {
            // Increase the value of t , i.e.
            // how much time a process
            has been processed.
            t = t + rem_bt[i] ;
```

Scanned with ACE Scanner

// 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 turned around time
void findTurnAroundTime (int processes[], int n,
int bt[], int bt[], int wt[], int tat[]);

// calculating turnaround time by adding
// bt[i] + wt[i]

Scanned with ACE Scanner

for (int i = 0; i < n; i++)
    tat[i] = bt[i] + wt[i];
}

// function to calculate average time
void findavgTime (int processes[],
int n, int bt[], int quantum);

{
    int wt[n], tat[n], total_wt = 0,
total_tat = 0;

// 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 the details
cout << "PN\t" << "BT" << "\tWT" << "\tTAT\n";

// calculate total waiting time and total turn

```
for (int i = 0; i < n; i++)
{
 total_wt = total_wt + wt[i];
 total_tat = total_tat + tat[i];
 cout << "  " << i+1 << "\t\t"
 << bt[i] << "\t "
   << wt[i] << "\t\t" << tat[i]
 << endl;
}

cout << "Average waiting time = "
     << (float) total_wt / (float) n;
cout << "\n Average turn around time = "
     << (float) total_tat / (float) n;
}
// Driver code
int main()
{
 // process id's
 int processes[] = {1, 2, 3};
 int n = size of processes / size of
processes[0];

// Burst time of all processes of
all processes
int burst_time[] = {10, 5, 8};

// Time quantum
 int quantum = 2
```

Scanned with ACE Scanner

Roll No. ..... Scholar No. .....

findavgTime(processes, n, burst_time, quantum);
return 0;
}

output →

PN	BT	WT	TAT
1	10	13	23
2	5	10	15
3	8	13	21

Average waiting time = 12
Average turn around time = 19.667

Program for Round Robin Scheduling with arrival time as zero & different and same arrival times

```
#include <iostream>
#include <limits>

using namespace std;
struct Process {
    int AT, BT, ST[20], WT, FT, TAT, pos;
};
```

SESSIONAL PAPERS
Visha PRODUCTS BHOPAL

Scanned with ACE Scanner

```
};
int quant;

int main() {
    int n, i, j;

// Taking input →
cout << "Enter the no of processes:";
cin >> n;
Process p[n];

cout << "Enter the quantum:" << endl;
cin >> n;
Process p[n];

cout << "Enter the quantum:" << endl;
cin >> quant;

cout << "Enter the quantum:" <<
endl;
cin >> quant;

cout << "Enter the process numbers:
" << endl;
for (i = 0; i < n; i++)
    cin >> p[i].pid;

cout << "Enter the burst time of
processes:" << endl;
```

Scanned with ACE Scanner

```
for (i = 0; i < n; i++)
    cin >> p[i].BT;
// Declaring variable →
int c = n, s[n][20];
float time = 0, mini = INT_MAX,
b[n], a[n];

// Initializing burst and arrival
time arrays
int index = -1;
for (i = 0; i < n; i++) {
    b[i] = p[i].BT;
    a[i] = p[i].AT;
    for (j = 0; j < 20; j++) {
        s[i][j] = -1;
    }
}

int tot_wt, tot_tat;
tot_wt = 0;
tot_tat = 0;
bool flag = false;

while (c != 0) {
    mini = INT_MAX;
    flag = false;

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

```
total_wt = total_wt + wt[i];
total_tat = total_tat + tat[i];
cout << "\t" << i+1 << "\t" << bt[i] << "\t"
<< wt[i] << "\t\t" << tat[i]
<< endl;
}

cout << "\nAverage waiting time = "
<< (float) total_wt / (float) n;
cout << "\nAverage turn around time = "
<< (float) total_tat / (float) n;
cout << "\nAverage turn around time"
<< (float) total_tat / (float) n;
}

// Driver code
int main()
{
int processes[] = {1, 2, 3};
int processes[] = {1, 2, 3};
int n = size of processes / size of
processes[0];

int processes[] = {10, 5, 8};

findavgTime(processes, n, burst_time);
return 0;
```

Scanned with ACE Scanner

Output →
Processes Burst time
waiting time Turn
around time || the output is wrong
please correct it

1	10	0
10		
2	5	10
15		
3	8	15
23		

Average waiting time = 8.33333
Average turn around time = 16

Scanned with ACE Scanner

FCFS →

```
#include <iostream>
using namespace std;

// function to find the waiting time for
all
// processes
void findWaitingTime(int processes[], int
n, int bt[], int wt[])
{
    // waiting time for first process is 0
    wt[0] = 0;
    // calculating waiting time
    for (int i = 1; i; i < n; i++)
        wt[i] = bt[i-1] + wt[i-1];
}

// function to calculate turn around time
void findTurnAroundTime(int process
[], int n,
        int bt[], int wt[], int wt[],
        int tat[])
{
    for int i = 0; i < n; i++)
    tat[i] = bt[i] + wt[i];
}
```

Scanned with ACE Scanner

FCFS →

```
#include <iostream>
using namespace std;

// Function to find the waiting time for
   all
// processes
void findWaitingTime(int processes[], int
  n, int bt[], int wt[])
{
  // waiting time for first process is 0
  wt[0] = 0;
  // calculating waiting time
  for (int i = 1; i < n; i++)
      wt[i] = bt[i-1] + wt[i-1];
}

// Function to calculate turn around time
void findTurnAroundTime(int process
[], int n
          int bt[], int wt[], int wt[],
          int tat[])
{
  for int i = 0; i < n; i++)
  tat[i] = bt[i] + wt[i];
}
```

Scanned with ACE Scanner

FCFS →

```
#include <iostream>
using namespace std;

// Function to find the waiting time for
   all
// processes
void findWaitingTime(int processes[], int
  n, int bt[], int wt[])
{
  // waiting time for first process is 0
  wt[0] = 0;
  // calculating waiting time
  for (int i = 1; i; i < n; i++)
      wt[i] = bt[i-1] + wt[i-1];
}

// Function to calculate turn around time
void findTurnAround time( int process
[], int n
      int bt[], int wt[], int wt[],
      int tat[])
{
  for int i = 0; i < n; i++)
  tat[i] = bt[i] + wt[i];
}
```

Scanned with ACE Scanner

```
// function to calculate turn around t
void findTurnAroundTime(int proce
[], int n, int bt[], int wt[], in
tat[])
{
    // calculating turnaround time by
adding
    // bt[i] + wt[i]
    for (int i = 0; i < n; i++)
        tat[i] = bt[i] + wt[i];
}

// function to calculate average time
void findavgTime(int processes[]
n, int bt[])
{
    int wt[n], tat[n], total_wt =
total_tat = 0;

findwaitingTime(processes, n, b
, wt, tat);

// Display processes along with all
cout << "processes " << "
<< "waiting Time " << "Turn
Around time\n";

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

Scanned with ACE Scanner

```
float p time +0.1;
if (a[i] <= p && mini > a[i] && b(i)
>0) {
    index = i;
    mini = a[i];
    flag = true;
}
}
// if at = 1 then loop gets out hence
set flag to false
if at = 1 then loop gets out hence set
flag to false.
if (! flag) {
    time++;
    continue;
}

// calculating start time
j=0;
while (s[index][j] != -1)
{
    j++;
}
if (s index)(j) == -1)
    s[index][j] = time;
    p[index].ST[j] = time;
}
if (b[index] <= quant) {
```

Scanned with ACE Scanner

```
double avg_wt, avg_tat;
avg_wt = tot_wt / static_cast<double>(n);
avg_tat = tot_tat / static_cast<double>(n);

cout << "The average wait time is "
     << avg_wt << endl;
cout << "The average turnaround time is "
     << avg_tat << endl;

return 0;
}
```

Scanned with ACE Scanner