OPERATING SYSTEM LAB FILE

1. FIRST COME FIRST SERVE :

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

void findWaitingTime(int processes[], int n,
        int bt[], int wt[])
{       wt[0] = 0;

        for (int i = 1; i < n ; i++ )
                wt[i] = bt[i-1] + wt[i-1] ;
}
void findTurnAroundTime( int processes[], int n,
         int bt[], int wt[], int tat[])
{
        for (int i = 0; i < n ; i++)
                tat[i] = bt[i] + wt[i];
}
void findavgTime( int processes[], int n, int bt[])
{
        int wt[n], tat[n], total_wt = 0, total_tat = 0;

        findWaitingTime(processes, n, bt, wt);

        findTurnAroundTime(processes, n, bt, wt, tat);
        cout << "Processes "<< " Burst time "
                << " Waiting time " << " Turn around time\n";
```

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

OUTPUT :

```
Processes  Burst time  Waiting time  Turn around time
 1        10     0         10
 2        5     10         15
 3        8     15         23
Average waiting time = 8.33333
Average turn around time = 16
```

1. SHORTEST JOB FIRST :

```
#include <bits/stdc++.h>
using namespace std;

struct Process {
	int pid; // Process ID
	int bt; // Burst Time
```

```
    int art; // Arrival Time
};

void findWaitingTime(Process proc[], int n,
                                                        int wt[])
{
    int rt[n];

    for (int i = 0; i < n; i++)
        rt[i] = proc[i].bt;

    int complete = 0, t = 0, minm = INT_MAX;
    int shortest = 0, finish_time;
    bool check = false;

    while (complete != n) {
        for (int j = 0; j < n; j++) {
            if ((proc[j].art <= t) &&
            (rt[j] < minm) && rt[j] > 0) {
                minm = rt[j];
                shortest = j;
                check = true;
            }
        }

        if (check == false) {
            t++;
            continue;
        }
        rt[shortest]--;
```

```
            minm = rt[shortest];
            if (minm == 0)
                    minm = INT_MAX;
            if (rt[shortest] == 0) {
                    complete++;
                    check = false;

                    finish_time = t + 1;

                    wt[shortest] = finish_time -
                                            proc[shortest].bt -
                                            proc[shortest].art;

                    if (wt[shortest] < 0)
                            wt[shortest] = 0;
            }
            t++;
    }
}
void findTurnAroundTime(Process proc[], int n,
                                            int wt[], int tat[])
{
    for (int i = 0; i < n; i++)
            tat[i] = proc[i].bt + wt[i];
}
void findavgTime(Process proc[], int n)
{
    int wt[n], tat[n], total_wt = 0,
                                    total_tat = 0;
    findWaitingTime(proc, n, wt);

    findTurnAroundTime(proc, n, wt, tat);
    cout << " P\t\t"
        << "BT\t\t"
        << "WT\t\t"
        << "TAT\t\t\n";

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

OUTPUT :

```
P        BT        WT        TAT
 1        6         7         13
 2        2         0         2
 3        8         14         22
 4        3         0         3
 5        4         2         6
```

```
Average waiting time = 4.6
Average turn around time = 9.2
```

2. SHORTEST REMAINING TIME FIRST :

```
#include <bits/stdc++.h>
using namespace std;

struct Process {
    int pid; // Process ID
    int bt; // Burst Time
    int art; // Arrival Time
};

void findWaitingTime(Process proc[], int n,
                                                int wt[])
{
    int rt[n];
    for (int i = 0; i < n; i++)
            rt[i] = proc[i].bt;

    int complete = 0, t = 0, minm = INT_MAX;
    int shortest = 0, finish_time;
    bool check = false;

    while (complete != n) {
            for (int j = 0; j < n; j++) {
                    if ((proc[j].art <= t) &&
                    (rt[j] < minm) && rt[j] > 0) {
                            minm = rt[j];
                            shortest = j;
```

```
        check = true;
    }
}

if (check == false) {
    t++;
    continue;
}

rt[shortest]--;

minm = rt[shortest];
if (minm == 0)
    minm = INT_MAX;

if (rt[shortest] == 0) {

    complete++;
    check = false;


    finish_time = t + 1;


    wt[shortest] = finish_time -
                proc[shortest].bt -
                proc[shortest].art;

    if (wt[shortest] < 0)
        wt[shortest] = 0;
        }
        t++;
    }
}
void findTurnAroundTime(Process proc[], int n,
                                            int wt[], int tat[])
{
    for (int i = 0; i < n; i++)
        tat[i] = proc[i].bt + wt[i];
}
void findavgTime(Process proc[], int n)
{
    int wt[n], tat[n], total_wt = 0,
                                    total_tat = 0;


    findWaitingTime(proc, n, wt);
    findTurnAroundTime(proc, n, wt, tat);
    cout << " P\t\t"
        << "BT\t\t"
        << "WT\t\t"
        << "TAT\t\t\n";

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

OUTPUT :

```
P         BT         WT         TAT
 1         6          7          13
 2         2          0          2
 3         8          14          22
 4         3          0          3
 5         4          2          6

Average waiting time = 4.6
Average turn around time = 9.2
```

3. ROUND ROBIN :

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


void findWaitingTime(int processes[], int n,
                        int bt[], int wt[], int quantum)
{



        int rem_bt[n]
        for (int i = 0 ; i < n ; i++)
```

```
            rem_bt[i] = bt[i];

        int t = 0; // Current time


        while (1)
        {
            bool done = true;


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

                if (rem_bt[i] > 0)
                {
                    done = false; // There is a pending process

                    if (rem_bt[i] > quantum)
                    {

                        t += quantum;


                        rem_bt[i] -= quantum;
                    }


                    else
                    {

                        t = t + rem_bt[i];


                        wt[i] = t - bt[i];


                        rem_bt[i] = 0;
                    }
                }
            }


            if (done == true)
            break;
        }
    }
```

```
void findTurnAroundTime(int processes[], int n,
                                        int bt[], int wt[], int tat[])
{

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


void findavgTime(int processes[], int n, int bt[],
                                                    int quantum)
{
        int wt[n], tat[n], total_wt = 0, total_tat = 0;


        findWaitingTime(processes, n, bt, wt, quantum);


        findTurnAroundTime(processes, n, bt, wt, tat);


        cout << "PN\t "<< " \tBT "
                << " WT " << " \tTAT\n";


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

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 = 19.6667
```

4. PRIORITY SCHEDULING:

```
#include<bits/stdc++.h>
using namespace std;

struct Process
{
        int pid; // Process ID
        int bt; // CPU Burst time required
        int priority; // Priority of this process
};

bool comparison(Process a, Process b)
{
        return (a.priority > b.priority);
}

void findWaitingTime(Process proc[], int n,
                                        int wt[])
{
    wt[0] = 0;


    for (int i = 1; i < n ; i++ )
        wt[i] = proc[i-1].bt + wt[i-1] ;
}

void findTurnAroundTime( Process proc[], int n,
                        int wt[], int tat[])
{

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

void findavgTime(Process proc[], int n)
{
    int wt[n], tat[n], total_wt = 0, total_tat = 0;


    findWaitingTime(proc, n, wt);


    findTurnAroundTime(proc, n, wt, tat);


    cout << "\nProcesses "<< " Burst time "
        << " Waiting time " << " Turn around time\n";
    for (int i=0; i<n; i++)
    {
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];
        cout << " " << proc[i].pid << "\t\t"
            << proc[i].bt << "\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;
}

void priorityScheduling(Process proc[], int n)
{
    // Sort processes by priority
    sort(proc, proc + n, comparison);

    cout<< "Order in which processes gets executed \n";
    for (int i = 0 ; i < n; i++)
        cout << proc[i].pid <<" " ;

    findavgTime(proc, n);
}
```

OUTPUT :

```
Order in which processes gets executed
1 3 2
Processes  Burst time  Waiting time  Turn around time
 1        10      0          10
 3        8      10          18
 2        5      18          23

Average waiting time = 9.33333
Average turn around time = 17
```

5. BANKERS ALGORITM:

```
#include <stdio.h>
int main()
{
        int n, m, i, j, k;
        n = 5;
        m = 3;
        int alloc[5][3] = { { 0, 1, 0 },
                                          { 2, 0, 0 },
                                          { 3, 0, 2 },
                                          { 2, 1, 1 },
                                          { 0, 0, 2 } };

        int max[5][3] = { { 7, 5, 3 },
                                     { 3, 2, 2 },
                                     { 9, 0, 2 },
                                     { 2, 2, 2 },
                                     { 4, 3, 3 } };
```

```
int avail[3] = { 3, 3, 2 };


int f[n], ans[n], ind = 0;
for (k = 0; k < n; k++) {
    f[k] = 0;
}
int need[n][m];
for (i = 0; i < n; i++) {
    for (j = 0; j < m; j++)
        need[i][j] = max[i][j] - alloc[i][j];
}
int y = 0;
for (k = 0; k < 5; k++) {
    for (i = 0; i < n; i++) {
        if (f[i] == 0) {


            int flag = 0;
            for (j = 0; j < m; j++) {
                if (need[i][j] > avail[j]){
                    flag = 1;
                    break;
                }
            }


            if (flag == 0) {
                ans[ind++] = i;
                for (y = 0; y < m; y++)
                    avail[y] += alloc[i][y];
                f[i] = 1;
                }
            }
        }
    }

    int flag = 1;

    for(int i=0;i<n;i++)
    {
    if(f[i]==0)
    {
        flag=0;
        printf("The following system is not safe");
        break;
    }
    }

    if(flag==1)
    {
    printf("Following is the SAFE Sequence\n");
    for (i = 0; i < n - 1; i++)
        printf(" P%d ->", ans[i]);
    printf(" P%d", ans[n - 1]);
    }



    return (0);



}
```

```
                }
```

OUTPUT :

```
Following is the SAFE Sequence
 P1 -> P3 -> P4 -> P0 -> P2
```

6. FIRST FIT ALGORITM:

```
#include<bits/stdc++.h>
using namespace std;


void firstFit(int blockSize[], int m,
                        int processSize[], int n)
{

        int allocation[n];


        memset(allocation, -1, sizeof(allocation));


        for (int i = 0; i < n; i++)
        {
                for (int j = 0; j < m; j++)
                {
                        if (blockSize[j] >= processSize[i])
                        {

                                allocation[i] = j;


                                blockSize[j] -= processSize[i];

                                break;
                        }
                }
        }
        cout << "\nProcess No.\tProcess Size\tBlock no.\n";
        for (int i = 0; i < n; i++)
        {
            cout << " " << i+1 << "\t\t"
                << processSize[i] << "\t\t";
            if (allocation[i] != -1)
                cout << allocation[i] + 1;
            else
                cout << "Not Allocated";
            cout << endl;
        }
    }
```

OUTPUT:

```
Process No.     Process Size    Block no.
1               212             2
2               417             5
3               112             2
4               426             Not Allocated
```

7. WORST FIT ALGORITM:

```
#include<bits/stdc++.h>
using namespace std;



void worstFit(int blockSize[], int m, int processSize[],
                                                        int
n)
{
```

```
int allocation[n];

memset(allocation, -1, sizeof(allocation));

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

        int wstIdx = -1;
        for (int j=0; j<m; j++)
        {
                if (blockSize[j] >= processSize[i])
                {
                        if (wstIdx == -1)
                                wstIdx = j;
                        else if (blockSize[wstIdx] < blockSize[j])
                                wstIdx = j;
                }
        }


        if (wstIdx != -1)
        {

                allocation[i] = wstIdx;
                blockSize[wstIdx] -= processSize[i];
        }
}

cout << "\nProcess No.\tProcess Size\tBlock no.\n";
for (int i = 0; i < n; i++)
```

```
        {
                cout << " " << i+1 << "\t\t" << processSize[i] << "\t\t";
                if (allocation[i] != -1)
                        cout << allocation[i] + 1;
                else
                        cout << "Not Allocated";
                cout << endl;
        }
}
```

OUTPUT :

```
Process No.    Process Size    Block no.
   1         212         5
   2         417         2
   3         112         5
   4         426         Not Allocated
```

8. BEST FIT :

```
public class GFG
{
        static void bestFit(int blockSize[], int m, int processSize[],

        int n)
        {

                int allocation[] = new int[n];

                // Initially no block is assigned to any process
```

```
for (int i = 0; i < allocation.length; i++)
    allocation[i] = -1;
for (int i=0; i<n; i++)
{

    int bestIdx = -1;
    for (int j=0; j<m; j++)
    {
        if (blockSize[j] >= processSize[i])
        {
            if (bestIdx == -1)
                bestIdx = j;
            else if (blockSize[bestIdx] > blockSize[j])
                bestIdx = j;
        }
    }


    if (bestIdx != -1)
    {
        allocation[i] = bestIdx;


        blockSize[bestIdx] -= processSize[i];
    }
}

System.out.println("\nProcess No.\tProcess Size\tBlock no.");
for (int i = 0; i < n; i++)
{
```

```
                        System.out.print(" " + (i+1) + "\t\t" + processSize[i] + "\t\t");
                        if (allocation[i] != -1)
                                System.out.print(allocation[i] + 1);
                        else
                                System.out.print("Not Allocated");
                        System.out.println();
                }
        }
```

OUTPUT :

```
Process No.    Process Size    Block no.
 1        212        4
 2        417        2
 3        112        3
 4        426        5
```