1.fcfs

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

int main()
{
    int n;
    float totalWT = 0, totalTAT = 0;

    cout << "Enter the number of processes -- ";
    cin >> n;

    int bt[n], wt[n], tat[n];


    for(int i = 0; i < n; i++)
    {
        cout << "Enter Burst Time for Process " << i << " -- ";
        cin >> bt[i];
    }


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


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


    cout << "\nPROCESS\tBURST TIME\tWAITING TIME\tTURNAROUND TIME\n";

    for(int i = 0; i < n; i++)
    {
        cout << "P" << i << "\t"
             << bt[i] << "\t\t"
             << wt[i] << "\t\t"
             << tat[i] << endl;

        totalWT += wt[i];
        totalTAT += tat[i];
    }


    cout << "\nAverage Waiting Time-- " << totalWT / n;
    cout << "\nAverage Turnaround Time -- " << totalTAT / n;

    return 0;
}
```

2.Fcfs(Arrival)\

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

int main()
{
    int n;
    cout << "Enter the number of processes -- ";
    cin >> n;
    int at[n], bt[n], wt[n], tat[n], pid[n];

    for(int i = 0; i < n; i++)
    {
        pid[i] = i;
        cout << "Enter Arrival Time for Process " << i << " -- ";
        cin >> at[i];

        cout << "Enter Burst Time for Process " << i << " -- ";
        cin >> bt[i];
    }
    for(int i = 0; i < n-1; i++)
    {
        for(int j = i+1; j < n; j++)
        {
            if(at[i] > at[j])
            {
                swap(at[i], at[j]);
                swap(bt[i], bt[j]);
                swap(pid[i], pid[j]);
            }
        }
    }
    wt[0] = 0;
    int currentTime = bt[0];

    for(int i = 1; i < n; i++)
    {
        if(currentTime < at[i])
        {
            currentTime = at[i];
        }

        wt[i] = currentTime - at[i];
        currentTime += bt[i];
    }


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

    float totalWT = 0, totalTAT = 0;

    cout << "\nPROCESS\tARRIVAL TIME\tBURST TIME\tWAITING TIME\tTURNAROUND TIME\n";

    for(int i = 0; i < n; i++)
    {
        cout << "P" << pid[i] << "\t"
             << at[i] << "\t\t"
             << bt[i] << "\t\t"
             << wt[i] << "\t\t"
            << tat[i] << endl;

        totalWT += wt[i];
        totalTAT += tat[i];
    }

    cout << "\nAverage Waiting Time -- " << totalWT / n;
    cout << "\nAverage Turnaround Time -- " << totalTAT / n;

    return 0;
}
```

3.Sjf

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

int main()
{
    int n;
    cout << "Enter number of processes -- ";
    cin >> n;

    int bt[n], wt[n], tat[n], pid[n];


    for(int i = 0; i < n; i++)
    {
        pid[i] = i;
        cout << "Enter Burst Time for Process " << i << " -- ";
        cin >> bt[i];
    }


    for(int i = 0; i < n-1; i++)
    {
        for(int j = i+1; j < n; j++)
        {
            if(bt[i] > bt[j])
            {
                swap(bt[i], bt[j]);
                swap(pid[i], pid[j]);
            }
        }
    }


    wt[0] = 0;

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


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

    float totalWT = 0, totalTAT = 0;
    cout << "\nPROCESS\tBURST TIME\tWAITING TIME\tTURNAROUND TIME\n";

    for(int i = 0; i < n; i++)
    {
        cout << "P" << pid[i] << "\t"
             << bt[i] << "\t\t"
             << wt[i] << "\t\t"
             << tat[i] << endl;

        totalWT += wt[i];
        totalTAT += tat[i];
    }

    cout << "\nAverage Waiting Time -- " << totalWT / n;
    cout << "\nAverage Turnaround Time -- " << totalTAT / n;

    return 0;
}
```

4.sjf(arrival)

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

int main()
{
    int n;
    cout << "Enter number of processes -- ";
    cin >> n;

    int at[n], bt[n], wt[n], tat[n], pid[n];
    bool completed[n];

    for(int i = 0; i < n; i++)
    {
        pid[i] = i;
        completed[i] = false;

        cout << "Enter Arrival Time for Process " << i << " -- ";
        cin >> at[i];

        cout << "Enter Burst Time for Process " << i << " -- ";
        cin >> bt[i];
    }

    int time = 0, completedCount = 0;

    while(completedCount < n)
    {
        int idx = -1;
        int minBT = 9999;


        for(int i = 0; i < n; i++)
        {
            if(at[i] <= time && !completed[i])
            {
                if(bt[i] < minBT)
                {
                    minBT = bt[i];
                    idx = i;
                }
            }
        }


        if(idx == -1)
        {
            time++;
        }
        else
        {
            wt[idx] = time - at[idx];
            time += bt[idx];
            tat[idx] = wt[idx] + bt[idx];

            completed[idx] = true;
            completedCount++;
        }
    }

    float totalWT = 0, totalTAT = 0;


    cout << "\nPROCESS\tARRIVAL TIME\tBURST TIME\tWAITING TIME\tTURNAROUND TIME\n";

    for(int i = 0; i < n; i++)
    {
        cout << "P" << i << "\t"
             << at[i] << "\t\t"
             << bt[i] << "\t\t"
             << wt[i] << "\t\t"
             << tat[i] << endl;

        totalWT += wt[i];
        totalTAT += tat[i];
    }

    cout << "\nAverage Waiting Time -- " << totalWT / n;
    cout << "\nAverage Turnaround Time -- " << totalTAT / n;

    return 0;
}
```

5.srtf

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

int main()
{
    int n;
    cout << "Enter number of processes -- ";
    cin >> n;

    int bt[n], rt[n], wt[n], tat[n];

    // Input Burst Time
    for(int i = 0; i < n; i++)
    {
```

```
    cout << "Enter Burst Time for Process " << i << " -- ";
    cin >> bt[i];
    rt[i] = bt[i];   // Remaining Time initialize
}

int completed = 0, time = 0;
int shortest;
int minRT;
bool found;

while(completed != n)
{
    shortest = -1;
    minRT = 9999;
    found = false;

    // Find process with shortest remaining time
    for(int i = 0; i < n; i++)
    {
        if(rt[i] > 0 && rt[i] < minRT)
        {
            minRT = rt[i];
            shortest = i;
            found = true;
        }
    }

    // If no process found
    if(!found)
    {
        time++;
        continue;
    }

    // Execute shortest process for 1 unit time
    rt[shortest]--;
    time++;

    // If process completed
    if(rt[shortest] == 0)
    {
        completed++;
        wt[shortest] = time - bt[shortest];
        tat[shortest] = wt[shortest] + bt[shortest];
    }
}

float totalWT = 0, totalTAT = 0;

// Output
cout << "\nPROCESS\tBURST TIME\tWAITING TIME\tTURNAROUND TIME\n";

for(int i = 0; i < n; i++)
{
    cout << "P" << i << "\t"
         << bt[i] << "\t\t"
         << wt[i] << "\t\t"
         << tat[i] << endl;

    totalWT += wt[i];
    totalTAT += tat[i];
    }

    cout << "\nAverage Waiting Time -- " << totalWT / n;
    cout << "\nAverage Turnaround Time -- " << totalTAT / n;

    return 0;
}
```

6.Priority(lower number=high priority)

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

int main()
{
    int n;
    cout << "Enter number of processes -- ";
    cin >> n;

    int bt[n], pr[n], wt[n], tat[n], pid[n];

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

        cout << "Enter Burst Time for Process " << i << " -- ";
        cin >> bt[i];

        cout << "Enter Priority for Process " << i << " -- ";
        cin >> pr[i];
    }

    for(int i = 0; i < n-1; i++)
    {
        for(int j = i+1; j < n; j++)
        {
            if(pr[i] > pr[j])
            {
                swap(pr[i], pr[j]);
                swap(bt[i], bt[j]);
                swap(pid[i], pid[j]);
            }
        }
    }

    wt[0] = 0;

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

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

    float totalWT = 0, totalTAT = 0;
    cout << "\nPROCESS\tPRIORITY\tBURST TIME\tWAITING TIME\tTURNAROUND TIME\n";

    for(int i = 0; i < n; i++)
    {
        cout << "P" << pid[i] << "\t"
             << pr[i] << "\t\t"
             << bt[i] << "\t\t"
             << wt[i] << "\t\t"
             << tat[i] << endl;

        totalWT += wt[i];
        totalTAT += tat[i];
    }
    cout << "\nAverage Waiting Time -- " << totalWT / n;
    cout << "\nAverage Turnaround Time -- " << totalTAT / n;

    return 0;
}
```

8.priority(with arrival)

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

int main()
{
    int n;
    cout << "Enter number of processes -- ";
    cin >> n;

    int at[n], bt[n], pr[n], wt[n], tat[n];
    bool completed[n];


    for(int i = 0; i < n; i++)
    {
        completed[i] = false;

        cout << "Enter Arrival Time for Process " << i << " -- ";
        cin >> at[i];

        cout << "Enter Burst Time for Process " << i << " -- ";
        cin >> bt[i];

        cout << "Enter Priority for Process " << i << " -- ";
        cin >> pr[i];
    }

    int time = 0, done = 0;

    while(done < n)
    {
        int idx = -1;
        int highestPriority = 9999;


        for(int i = 0; i < n; i++)
        {
            if(at[i] <= time && !completed[i])
            {
                if(pr[i] < highestPriority)
                {
                    highestPriority = pr[i];
                    idx = i;
                }
            }
        }

        if(idx == -1)
        {
            time++;
        }
        else
        {
            wt[idx] = time - at[idx];
            time += bt[idx];
            tat[idx] = wt[idx] + bt[idx];

            completed[idx] = true;
            done++;
        }
    }

    float totalWT = 0, totalTAT = 0;

    // Output
    cout << "\nPROCESS\tARRIVAL TIME\tPRIORITY\tBURST TIME\tWAITING TIME\tTURNAROUND TIME\n";

    for(int i = 0; i < n; i++)
    {
        cout << "P" << i << "\t"
             << at[i] << "\t\t"
             << pr[i] << "\t\t"
             << bt[i] << "\t\t"
             << wt[i] << "\t\t"
             << tat[i] << endl;

        totalWT += wt[i];
        totalTAT += tat[i];
    }

    cout << "\nAverage Waiting Time -- " << totalWT / n;
    cout << "\nAverage Turnaround Time -- " << totalTAT / n;

    return 0;
}
```

9.priority(high=high priority)(without at)

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

int main()
{
    int n;
    cout << "Enter number of processes -- ";
    cin >> n;

    int bt[n], pr[n], wt[n], tat[n], pid[n];


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

        cout << "Enter Burst Time for Process " << i << " -- ";
        cin >> bt[i];

        cout << "Enter Priority for Process " << i << " -- ";
        cin >> pr[i];
    }


    for(int i = 0; i < n-1; i++)
    {
        for(int j = i+1; j < n; j++)
        {
            if(pr[i] < pr[j])
            {
                swap(pr[i], pr[j]);
                swap(bt[i], bt[j]);
                swap(pid[i], pid[j]);
            }
        }
    }


    wt[0] = 0;

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

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

    float totalWT = 0, totalTAT = 0;

    cout << "\nPROCESS\tPRIORITY\tBURST TIME\tWAITING TIME\tTURNAROUND TIME\n";

    for(int i = 0; i < n; i++)
    {
        cout << "P" << pid[i] << "\t"
             << pr[i] << "\t\t"
             << bt[i] << "\t\t"
             << wt[i] << "\t\t"
             << tat[i] << endl;

        totalWT += wt[i];
        totalTAT += tat[i];
    }

    cout << "\nAverage Waiting Time -- " << totalWT / n;
    cout << "\nAverage Turnaround Time -- " << totalTAT / n;

    return 0;
}
```

10.RR

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

int main()
{
    int n, tq;
    cout << "Enter number of processes -- ";
    cin >> n;

    int bt[n], rt[n], wt[n], tat[n];

    cout << "Enter Time Quantum -- ";
    cin >> tq;


    for(int i = 0; i < n; i++)
    {
        cout << "Enter Burst Time for Process " << i << " -- ";
        cin >> bt[i];
        rt[i] = bt[i];
        wt[i] = 0;
    }

    int time = 0;
    bool done;

    while(true)
    {
        done = true;

        for(int i = 0; i < n; i++)
        {
            if(rt[i] > 0)
            {
                done = false;

                if(rt[i] > tq)
                {
                    time += tq;
                    rt[i] -= tq;
                }
                else
                {
                    time += rt[i];
                    wt[i] = time - bt[i];
                    rt[i] = 0;
                }
            }
        }

        if(done == true)
            break;
    }

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

    float totalWT = 0, totalTAT = 0;

    cout << "\nPROCESS\tBURST TIME\tWAITING TIME\tTURNAROUND TIME\n";

    for(int i = 0; i < n; i++)
    {
        cout << "P" << i << "\t"
             << bt[i] << "\t\t"
             << wt[i] << "\t\t"
             << tat[i] << endl;

        totalWT += wt[i];
        totalTAT += tat[i];
    }

    cout << "\nAverage Waiting Time -- " << totalWT / n;
    cout << "\nAverage Turnaround Time -- " << totalTAT / n;

    return 0;
}
```

12.RR(with AT)

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

int main()
{
    int n, tq;
    cout << "Enter number of processes -- ";
    cin >> n;

    int at[n], bt[n], rt[n], wt[n], tat[n];
    bool visited[n];

    cout << "Enter Time Quantum -- ";
    cin >> tq;


    for(int i = 0; i < n; i++)
    {
        cout << "Enter Arrival Time for Process " << i << " -- ";
        cin >> at[i];

        cout << "Enter Burst Time for Process " << i << " -- ";
        cin >> bt[i];

        rt[i] = bt[i];
        visited[i] = false;
    }

    queue<int> q;
    int time = 0, completed = 0;


    q.push(0);
    visited[0] = true;
```

```
while(completed < n)
{
    int curr = q.front();
    q.pop();

    if(rt[curr] > tq)
    {
        time += tq;
        rt[curr] -= tq;
    }
    else
    {
        time += rt[curr];
        wt[curr] = time - at[curr] - bt[curr];
        rt[curr] = 0;
        completed++;
    }

    for(int i = 0; i < n; i++)
    {
        if(at[i] <= time && !visited[i])
        {
            q.push(i);
            visited[i] = true;
        }
    }


    if(rt[curr] > 0)
    {
        q.push(curr);
    }

    if(q.empty())
    {
        for(int i = 0; i < n; i++)
        {
            if(!visited[i])
            {
                q.push(i);
                visited[i] = true;
                break;
            }
        }
    }
}

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

float totalWT = 0, totalTAT = 0;


cout << "\nPROCESS\tARRIVAL TIME\tBURST TIME\tWAITING TIME\tTURNAROUND TIME\n";

for(int i = 0; i < n; i++)
{
    cout << "P" << i << "\t"
         << at[i] << "\t\t"
```

```
             << bt[i] << "\t\t"
             << wt[i] << "\t\t"
             << tat[i] << endl;

        totalWT += wt[i];
        totalTAT += tat[i];
    }

    cout << "\nAverage Waiting Time -- " << totalWT / n;
    cout << "\nAverage Turnaround Time -- " << totalTAT / n;

    return 0;
}
```

12.Dineing Philosoper

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

int main() {
    int totalPhilosophers, hungryCount;
    cout << "Enter the total no. of philosophers: ";
    cin >> totalPhilosophers;
    vector<int> hungryPositions;
    cout << "How many are hungry: ";
    cin >> hungryCount;
    for (int i = 0; i < hungryCount; i++) {
        int pos;
        cout << "Enter philosopher " << i + 1 << " position: ";
        cin >> pos;
        hungryPositions.push_back(pos - 1);
    }

    int choice;
    do {
        cout << "1.One can eat at a time 2.Two can eat at a time 3.Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        if (choice == 1) {
            cout << "Allow one philosopher to eat at any time\n";
            for (int i = 0; i < hungryCount; i++) {
                cout << "P " << hungryPositions[i] << " is granted to eat\n";
                for (int j = 0; j < hungryCount; j++) {
                    if (j != i) cout << "P " << hungryPositions[j] << " is waiting\n";
                }
            }
        }
        else if (choice == 2) {
            cout << "Allow two philosophers to eat at same time\n";
            for (int i = 0; i < hungryCount; i++) {
                for (int j = i + 1; j < hungryCount; j++) {
                    cout << "combination " << i + 1 << "\n";
                    cout << "P " << hungryPositions[i] << " and P " << hungryPositions[j] << " are granted to eat\n";
                    for (int k = 0; k < hungryCount; k++) {
                        if (k != i && k != j) cout << "P " << hungryPositions[k] << " is waiting\n";
                    }
                }
            }
        }
    } while (choice != 3);

    return 0;
}
```

12.MFT

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

int main() {
    int totalMemory, blockSize, numProcesses;
    cout << "Enter the total memory available (in Bytes) -- ";
    cin >> totalMemory;
    cout << "Enter the block size (in Bytes)-- ";
    cin >> blockSize;
    cout << "Enter the number of processes – ";
    cin >> numProcesses;
    int processes[numProcesses];
    for (int i = 0; i < numProcesses; i++) {
        cout << "Enter memory required for process " << i + 1 << " (in Bytes) -- ";
        cin >> processes[i];
    }
    int numBlocks = totalMemory / blockSize;
    cout << "No. of Blocks available in memory -- " << numBlocks << "\n\n";

    int internalFrag = 0, externalFrag = 0, allocatedBlocks = 0;
    cout << "PROCESS\tALLOCATED\tINTERNAL FRAGMENTATION\n";
    for (int i = 0; i < numProcesses; i++) {
        if (allocatedBlocks < numBlocks) {
            if (processes[i] <= blockSize) {
                int frag = blockSize - processes[i];
                cout << i + 1 << "\tYES\t\t" << frag << "\n";
                internalFrag += frag;
                allocatedBlocks++;
            } else {
                cout << i + 1 << "\tNO\t\t-----\n";
            }
        } else {
            cout << "Memory is Full, Remaining Processes cannot be accommodated\n";
            externalFrag += processes[i];
        }
    }
    cout << "Total Internal Fragmentation is " << internalFrag << "\n";
    cout << "Total External Fragmentation is " << totalMemory - allocatedBlocks * blockSize << "\n";
    return 0;
}
```

13.MVT:

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

int main() {
    int totalMemory, processSize, allocatedMemory = 0;
    char choice;
    cout << "Enter the total memory available (in Bytes)  ";
    cin >> totalMemory;

    int processNum = 1;
    cout << "\n";
    int memory[100], procCount = 0;
    while (true) {
        cout << "Enter memory required for process " << processNum << " (in Bytes) – ";
        cin >> processSize;
        if (allocatedMemory + processSize <= totalMemory) {
            memory[procCount++] = processSize;
            allocatedMemory += processSize;
            cout << "Memory is allocated for Process " << processNum << "\n";
        } else {
            cout << "Memory is Full\n";
            break;
        }
        cout << "Do you want to continue(y/n) – ";
        cin >> choice;
        if (choice == 'n' || choice == 'N') break;
        processNum++;
    }

    cout << "Total Memory Available – " << totalMemory << "\n\n";
    cout << "PROCESS\tMEMORY ALLOCATED\n";
    for (int i = 0; i < procCount; i++) {
        cout << i + 1 << "\t" << memory[i] << "\n";
    }
    cout << "Total Memory Allocated is " << allocatedMemory << "\n";
    cout << "Total External Fragmentation is " << totalMemory - allocatedMemory << "\n";
    return 0;
}
```

14.Worst Fit

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

int main() {
    int nBlocks, nFiles;
    cout << "Enter the number of blocks: ";
    cin >> nBlocks;
    cout << "Enter the number of files: ";
    cin >> nFiles;

    int blockSize[nBlocks], origBlockSize[nBlocks], fileSize[nFiles], blockAlloc[nFiles];
    bool allocated[nBlocks] = {false};

    for (int i = 0; i < nBlocks; i++) {
        cout << "Block " << i + 1 << ": ";
        cin >> blockSize[i];
        origBlockSize[i] = blockSize[i];
    }

    for (int i = 0; i < nFiles; i++) {
        cout << "File " << i + 1 << ": ";
        cin >> fileSize[i];
        blockAlloc[i] = -1;
        int worstIdx = -1;
        for (int j = 0; j < nBlocks; j++) {
            if (!allocated[j] && blockSize[j] >= fileSize[i]) {
                if (worstIdx == -1 || blockSize[j] > blockSize[worstIdx]) {
                    worstIdx = j;
                }
            }
        }
        if (worstIdx != -1) {
            blockAlloc[i] = worstIdx;
            allocated[worstIdx] = true;
        }
    }

    cout << "\nFile No\tFile Size\tBlock No\tBlock Size\tFragment\n";
    for (int i = 0; i < nFiles; i++) {
        if (blockAlloc[i] != -1) {
            int b = blockAlloc[i];
            cout << i + 1 << "\t" << fileSize[i] << "\t\t"
                 << b + 1 << "\t\t"
                 << origBlockSize[b] << "\t\t"
                 << origBlockSize[b] - fileSize[i] << "\n";
        } else {
            cout << i + 1 << "\t" << fileSize[i] << "\t\tNot Allocated\n";
        }
    }

    return 0;
}
```

15.Best Fit

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

int main() {
    int nBlocks, nFiles;
    cout << "Enter the number of blocks: ";
    cin >> nBlocks;
    cout << "Enter the number of files: ";
    cin >> nFiles;

    int blockSize[nBlocks], origBlockSize[nBlocks], fileSize[nFiles], blockAlloc[nFiles];
    bool allocated[nBlocks] = {false};

    for (int i = 0; i < nBlocks; i++) {
        cout << "Block " << i + 1 << ": ";
        cin >> blockSize[i];
        origBlockSize[i] = blockSize[i];
    }

    for (int i = 0; i < nFiles; i++) {
        cout << "File " << i + 1 << ": ";
        cin >> fileSize[i];
        blockAlloc[i] = -1;
        int bestIdx = -1;
        for (int j = 0; j < nBlocks; j++) {
            if (!allocated[j] && blockSize[j] >= fileSize[i]) {
                if (bestIdx == -1 || blockSize[j] < blockSize[bestIdx]) {
                    bestIdx = j;
                }
            }
        }
        if (bestIdx != -1) {
            blockAlloc[i] = bestIdx;
            allocated[bestIdx] = true;
        }
    }
    cout << "\nFile No\tFile Size\tBlock No\tBlock Size\tFragment\n";
    for (int i = 0; i < nFiles; i++) {
        if (blockAlloc[i] != -1) {
            int b = blockAlloc[i];
            cout << i + 1 << "\t" << fileSize[i] << "\t\t"
                 << b + 1 << "\t\t"
                 << origBlockSize[b] << "\t\t"
                 << origBlockSize[b] - fileSize[i] << "\n";
        } else {
            cout << i + 1 << "\t" << fileSize[i] << "\t\tNot Allocated\n";
        }
    }

    return 0;
}
```

16.First Fit

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

int main() {
    int nBlocks, nFiles;
    cout << "Enter the number of blocks: ";
    cin >> nBlocks;
    cout << "Enter the number of files: ";
    cin >> nFiles;

    int blockSize[nBlocks], origBlockSize[nBlocks], fileSize[nFiles], blockAlloc[nFiles];
    bool used[nBlocks] = {false};

    for (int i = 0; i < nBlocks; i++) {
        cout << "Block " << i + 1 << ": ";
        cin >> blockSize[i];
        origBlockSize[i] = blockSize[i];
    }

    for (int i = 0; i < nFiles; i++) {
        cout << "File " << i + 1 << ": ";
        cin >> fileSize[i];
        blockAlloc[i] = -1;

        for (int j = nBlocks - 1; j >= 0; j--) {
            if (!used[j] && blockSize[j] >= fileSize[i]) {
                blockAlloc[i] = j;
                used[j] = true;
                break;
            }
        }
    }

    cout << "\nFile No\tFile Size\tBlock No\tBlock Size\tFragment\n";
    for (int i = 0; i < nFiles; i++) {
        if (blockAlloc[i] != -1) {
            int b = blockAlloc[i];
            cout << i + 1 << "\t" << fileSize[i] << "\t\t"
                 << b + 1 << "\t\t"
                 << origBlockSize[b] << "\t\t"
                 << origBlockSize[b] - fileSize[i] << "\n";
        }
    }
    return 0;
}
```

17.FIFO

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

int main() {
    int pages[12] = {2,3,2,1,5,2,4,5,3,2,5,2};
    int frames[3];
    int pf = 0, pos = 0;

    for(int i = 0; i < 3; i++)
        frames[i] = -1;

    for(int i = 0; i < 12; i++) {
        int found = 0;

        for(int j = 0; j < 3; j++) {
            if(frames[j] == pages[i]) {
                found = 1;
                break;
            }
        }

        if(found == 0) {
            frames[pos] = pages[i];
            pos = (pos + 1) % 3;
            pf++;
        }

        for(int j = 0; j < 3; j++)
            cout << frames[j] << " ";
        cout << endl;
    }

    cout << "Number of page faults: " << pf << endl;

    return 0;
}
```

18.LRU

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

int main() {
    int pages[12] = {2,3,2,1,5,2,4,5,3,2,5,2};
    int frames[3], time[3];
    int pf = 0, counter = 0;

    for(int i = 0; i < 3; i++) {
        frames[i] = -1;
        time[i] = 0;
    }

    for(int i = 0; i < 12; i++) {
        int found = 0;
        for(int j = 0; j < 3; j++) {
            if(frames[j] == pages[i]) {
                counter++;
                time[j] = counter;
                found = 1;
                break;
            }
        }

        if(found == 0) {
            int min = time[0], pos = 0;

            for(int j = 1; j < 3; j++) {
                if(time[j] < min) {
                    min = time[j];
                    pos = j;
                }
            }

            counter++;
            frames[pos] = pages[i];
            time[pos] = counter;
            pf++;
        }

        for(int j = 0; j < 3; j++)
            cout << frames[j] << " ";
        cout << endl;
    }

    cout << "Number of page faults: " << pf << endl;

    return 0;
}
```

19.Optimal

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

int main() {
    int n, f;
    cout << "Enter length of the reference string: ";
    cin >> n;

    vector<int> pages(n);
    cout << "Enter the reference string: ";
    for(int i = 0; i < n; i++)
        cin >> pages[i];

    cout << "Enter no of frames: ";
    cin >> f;

    vector<int> frames(f, -1);
    int pf = 0;

    for(int i = 0; i < n; i++) {
        int found = 0;
        for(int j = 0; j < f; j++) {
            if(frames[j] == pages[i]) {
                found = 1;
                break;
            }
        }

        if(found == 0) {
            int pos = -1;

            for(int j = 0; j < f; j++) {
                if(frames[j] == -1) {
                    pos = j;
                    break;
                }
            }

            if(pos == -1) {
                int farthest = i + 1;
                int index = -1;

                for(int j = 0; j < f; j++) {
                    int k;
                    for(k = i + 1; k < n; k++) {
                        if(frames[j] == pages[k])
                            break;
                    }

                    if(k == n) {
                        index = j;
                        break;
                    }

                    if(k > farthest) {
                        farthest = k;
                        index = j;
                    }
                }

                pos = index;
            }

            frames[pos] = pages[i];
            pf++;
        }

        for(int j = 0; j < f; j++)
            cout << frames[j] << " ";
        cout << endl;
    }

    float rate = (float)pf / n * 100;
    cout << "Number of page faults : " << pf << " Page fault rate = " << rate << endl;

    return 0;
}
```

20.FiFO user input

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

int main() {
    int n, f;
    cout << "Enter length of the reference string: ";
    cin >> n;

    vector<int> pages(n);
    cout << "Enter the reference string: ";
    for(int i = 0; i < n; i++)
        cin >> pages[i];

    cout << "Enter no of frames: ";
    cin >> f;

    vector<int> frames(f, -1);
    int pf = 0, index = 0;

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

        for(int j = 0; j < f; j++) {
            if(frames[j] == pages[i]) {
                hit = 1;
                break;
            }
        }

        if(hit == 0) {
            frames[index] = pages[i];
            index = (index + 1) % f;
            pf++;
        }

        for(int j = 0; j < f; j++)
            cout << frames[j] << " ";
        cout << endl;
    }

    float rate = (float)pf / n * 100;

    cout << "Number of page faults : " << pf;
    cout << " Page fault rate = " << rate << endl;

    return 0;
}
```

21.LRU user input

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

int main() {
    int n, f;
    cout << "Enter length of the reference string: ";
    cin >> n;
    vector<int> pages(n);
    cout << "Enter the reference string: ";
    for(int i = 0; i < n; i++)
        cin >> pages[i];

    cout << "Enter no of frames: ";
    cin >> f;

    vector<int> frames(f, -1), recent(f, 0);
    int pf = 0, time = 0;

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

        for(int j = 0; j < f; j++) {
            if(frames[j] == pages[i]) {
                time++;
                recent[j] = time;
                hit = 1;
                break;
            }
        }

        if(hit == 0) {
            int pos = 0, min = recent[0];

            for(int j = 1; j < f; j++) {
                if(recent[j] < min) {
                    min = recent[j];
                    pos = j;
                }
            }

            time++;
            frames[pos] = pages[i];
            recent[pos] = time;
            pf++;
        }

        for(int j = 0; j < f; j++)
            cout << frames[j] << " ";
        cout << endl;
    }

    float rate = (float)pf / n * 100;

    cout << "Number of page faults : " << pf;
    cout << " Page fault rate = " << rate << endl;

    return 0;
}
```

21.Producer Consumer:

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

int main() {
    int buffer = -1;
    int choice, value;
    while(true) {
        cout << "1. Produce 2. Consume 3. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        if(choice == 1) {
            if(buffer != -1) {
                cout << "Buffer is Full\n";
            } else {
                cout << "Enter the value: ";
                cin >> value;
                buffer = value;
            }
        } else if(choice == 2) {
            if(buffer == -1) {
                cout << "Buffer is Empty\n";
            } else {
                cout << "The consumed value is " << buffer << endl;
                buffer = -1;
            }
        } else if(choice == 3) {
            break;
        } else {
            cout << "Invalid choice\n";
        }
    }

    return 0;
}
```

24. Indexed File Allocation (Chained Simulation)

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

int main() {
    int indexBlock;
    cout << "Enter index block: ";
    cin >> indexBlock;

    int n;
    cout << "Enter number of files on index: ";
    cin >> n;

    vector<int> fileSize(n);
    cout << "Enter memory requirement for each file:\n";
    for(int i = 0; i < n; i++)
        cin >> fileSize[i];

    vector<int> allocated(100, 0);
    cout << "Allocated\nFile indexed\n";

    for(int i = 0; i < n; i++) {
        int allocatedBlocks = 0;
        cout << indexBlock << "->" << i + 1 << ":";
        while(allocatedBlocks < fileSize[i]) {
            int loc = rand() % 100;
            if(allocated[loc] == 0) {
                allocated[loc] = 1;
                allocatedBlocks++;
                cout << loc;
                if(allocatedBlocks < fileSize[i])
                    cout << ",";
            }
        }
        cout << endl;
    }

    int more;
    cout << "Enter 1 to enter more files and 0 to exit: ";
    cin >> more;

    if(more == 1) {
        main();
    }

    return 0;
}
```

25.Banker’s Algorithm (Deadlock Avoidance Simulation)

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

int main() {
    int n, m;
    cout << "Enter the no. of processes and resources: ";
    cin >> n >> m;

    vector<vector<int>> maxClaim(n, vector<int>(m));
    vector<vector<int>> allocation(n, vector<int>(m));
    vector<vector<int>> need(n, vector<int>(m));
    vector<int> available(m);

    cout << "Enter the claim matrix:\n";
    for(int i = 0; i < n; i++)
        for(int j = 0; j < m; j++)
            cin >> maxClaim[i][j];

    cout << "Enter the allocation matrix:\n";
    for(int i = 0; i < n; i++)
        for(int j = 0; j < m; j++)
            cin >> allocation[i][j];

    cout << "Resource vector: ";
    for(int j = 0; j < m; j++)
        cin >> available[j];

    for(int i = 0; i < n; i++)
        for(int j = 0; j < m; j++)
            need[i][j] = maxClaim[i][j] - allocation[i][j];

    vector<bool> finish(n, false);
    vector<int> work = available;

    while(true) {
        bool done = true;
        for(int i = 0; i < n; i++) {
            if(!finish[i]) {
                bool canAllocate = true;
                for(int j = 0; j < m; j++) {
                    if(need[i][j] > work[j]) {
                        canAllocate = false;
                        break;
                    }
                }
                if(canAllocate) {
                    cout << "All the resources can be allocated to Process " << i+1 << endl;
                    cout << "Available resources are: ";
                    for(int j = 0; j < m; j++)
                        work[j] += allocation[i][j];
                    for(int j = 0; j < m; j++)
                        cout << work[j] << " ";
                    cout << endl;
                    cout << "Process " << i+1 << " executed?:y\n";
                    finish[i] = true;
                    done = false;
                }
            }
        }
        if(done) break;
    }

    bool safe = true;
    for(int i = 0; i < n; i++)
        if(!finish[i]) safe = false;

    if(safe) cout << "System is in safe mode\nThe given state is safe state\n";
    else cout << "System is not in safe state\n";

    return 0;
}
```

26.Deadlock Prevention (Safe Sequence Simulation)

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

struct Job {
    string name;
    int time;
    bool executed;
};

int main() {
    int n;
    cout << "Enter no of jobs: ";
    cin >> n;

    vector<Job> jobs(n);
    for(int i = 0; i < n; i++) {
        cout << "Enter name and time: ";
        cin >> jobs[i].name >> jobs[i].time;
        jobs[i].executed = false;
    }

    int available;
    cout << "Enter the available resources: ";
    cin >> available;
    vector<string> safeSeq;

    while(safeSeq.size() < n) {
        bool allocated = false;

        for(int i = 0; i < n; i++) {
            if(!jobs[i].executed && jobs[i].time <= available) {
                available += jobs[i].time;
                jobs[i].executed = true;
                safeSeq.push_back(jobs[i].name + " " + to_string(jobs[i].time));
                allocated = true;
            }
        }

        if(!allocated) {
            cout << "System is unsafe, deadlock might occur\n";
            break;
        }
    }

    if(safeSeq.size() == n) {
        cout << "Safe sequence is: ";
        for(int i = 0; i < n; i++) {
            cout << safeSeq[i];
            if(i != n-1) cout << ", ";
        }
        cout << "." << endl;
    }

    return 0;
}
```

27. FCFS Disk Scheduling (C++)

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

int main() {
    int n;
    cout << "Enter no.of tracks: ";
    cin >> n;

    vector<int> tracks(n);
    cout << "Enter track position: ";
    for(int i = 0; i < n; i++)
        cin >> tracks[i];

    cout << "Tracks traversed\tDifference between tracks" << endl;

    int totalMovement = 0;
    cout << tracks[0] << endl;

    for(int i = 1; i < n; i++) {
        int diff = abs(tracks[i] - tracks[i-1]);
        cout << tracks[i] << "\t\t" << diff << endl;
        totalMovement += diff;
    }
    double avg = (double)totalMovement / (n - 1);
    cout << "Average header movements:" << fixed << setprecision(6) << avg << endl;

    return 0;
}
```

28. SCAN Disk Scheduling (C++)

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath>
#include <iomanip>
using namespace std;

int main() {
    int n;
    cout << "Enter no.of tracks: ";
    cin >> n;

    vector<int> tracks(n);
    cout << "Enter track position: ";
    for(int i = 0; i < n; i++)
        cin >> tracks[i];

    sort(tracks.begin(), tracks.end(), greater<int>()); // Descending for SCAN

    cout << "Tracks traversed\tDifference between tracks" << endl;

    int totalMovement = 0;
    cout << tracks[0] << endl;

    for(int i = 1; i < n; i++) {
        int diff = abs(tracks[i] - tracks[i-1]);
        cout << tracks[i] << "\t\t" << diff << endl;
        totalMovement += diff;
    }

    double avg = (double)totalMovement / (n - 1);
    cout << "Average header movements: " << fixed << setprecision(2) << avg << endl;

    return 0;
}
```

29.C-SCAN Disk Scheduling (C++)

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath>
#include <iomanip>
using namespace std;

int main() {
    vector<int> tracks;
    int n, start;

    cout << "Enter no.of tracks: ";
    cin >> n;

    tracks.resize(n);
    cout << "Enter the track position: ";
    for(int i = 0; i < n; i++)
        cin >> tracks[i];

    cout << "Enter starting position: ";
    cin >> start;

    vector<int> left, right;
    for(int i = 0; i < n; i++) {
        if(tracks[i] >= start)
            right.push_back(tracks[i]);
        else
            left.push_back(tracks[i]);
    }

    sort(right.begin(), right.end());
    sort(left.begin(), left.end());

    vector<int> order;
    order.push_back(start);
    for(int i : right) order.push_back(i);
    for(int i : left) order.push_back(i);

    cout << "Tracks traversed\tDifference Between tracks" << endl;

    int totalMovement = 0;
    for(int i = 1; i < order.size(); i++) {
        int diff = abs(order[i] - order[i-1]);
        cout << order[i] << "\t\t" << diff << endl;
        totalMovement += diff;
    }

    double avg = (double)totalMovement / (order.size() - 1);
    cout << "Average header movements: " << fixed << setprecision(2) << avg << endl;

    return 0;
}
```

30.Sequential File Allocation (C++)

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

int main() {
    int start, length;
    cout << "Enter the starting block & length of file: ";
    cin >> start >> length;

    vector<int> disk(100, 0); // simulate 100 disk blocks, 0 = free

    bool canAllocate = true;
    for(int i = start; i < start + length; i++) {
        if(disk[i] != 0) {
            canAllocate = false;
            break;
        }
    }

    if(canAllocate) {
        for(int i = start; i < start + length; i++) {
            disk[i] = 1;
            cout << i << "->1" << endl;
        }
        cout << "The file is allocated to disk." << endl;
    } else {
        cout << "Cannot allocate file: blocks not free." << endl;
    }

    return 0;
}
```

31.Single Level Directory (C++)

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

int main() {
    string dirName;
    cout << "Enter name of directory -- ";
    cin >> dirName;

    vector<string> files;
    int choice;

    do {
        cout << "1. Create File 2. Delete File 3. Search File\n";
        cout << "4. Display Files 5. Exit Enter your choice – ";
        cin >> choice;

        if(choice == 1) {
            string fname;
            cout << "Enter the name of the file -- ";
            cin >> fname;
            files.push_back(fname);
        }
        else if(choice == 2) {
            string fname;
            cout << "Enter the name of the file – ";
            cin >> fname;
            bool found = false;
            for(auto it = files.begin(); it != files.end(); ++it) {
                if(*it == fname) {
                    files.erase(it);
                    found = true;
                    cout << "File " << fname << " is deleted" << endl;
                    break;
                }
            }
            if(!found) cout << "File " << fname << " not found" << endl;
        }
        else if(choice == 3) {
            string fname;
            cout << "Enter the name of the file – ";
            cin >> fname;
            bool found = false;
            for(string f : files) {
                if(f == fname) {
                    found = true;
                    break;
                }
            }
            if(found) cout << "File " << fname << " found" << endl;
            else cout << "File " << fname << " not found" << endl;
        }
        else if(choice == 4) {
            if(files.empty()) cout << "No files in directory" << endl;
            else {
                cout << "The Files are -- ";
                for(string f : files) cout << f << " ";
                cout << endl;
            }
        }

    } while(choice != 5);

    return 0;
}
```

32.Two-Level Directory (C++)

```
#include <iostream>
#include <map>
#include <vector>
#include <string>
using namespace std;

int main() {
    map<string, vector<string>> directories; // directory name -> files
    int choice;

    do {
        cout << "1. Create Directory 2. Create File 3. Delete File\n";
        cout << "4. Search File 5. Display 6. Exit\n";
        cout << "Enter your choice -- ";
        cin >> choice;

        if(choice == 1) {
            string dirName;
            cout << "Enter name of directory -- ";
            cin >> dirName;
            if(directories.find(dirName) == directories.end()) {
                directories[dirName] = vector<string>();
                cout << "Directory created" << endl;
            } else {
                cout << "Directory already exists" << endl;
            }
        }
        else if(choice == 2) {
            string dirName, fileName;
            cout << "Enter name of the directory – ";
            cin >> dirName;
            if(directories.find(dirName) != directories.end()) {
                cout << "Enter name of the file -- ";
                cin >> fileName;
                directories[dirName].push_back(fileName);
                cout << "File created" << endl;
            } else {
                cout << "Directory does not exist" << endl;
            }
        }
        else if(choice == 3) {
            string dirName, fileName;
            cout << "Enter name of the directory – ";
            cin >> dirName;
            if(directories.find(dirName) != directories.end()) {
                cout << "Enter name of the file -- ";
                cin >> fileName;
                auto &files = directories[dirName];
                bool found = false;
                for(auto it = files.begin(); it != files.end(); ++it) {
                    if(*it == fileName) {
                        files.erase(it);
                        found = true;
                        cout << "File deleted" << endl;
                        break;
                    }
                }
                if(!found) cout << "File not found" << endl;
            } else {
                cout << "Directory does not exist" << endl;
            }
        }
        else if(choice == 4) {
            string dirName, fileName;
            cout << "Enter name of the directory – ";
            cin >> dirName;
            if(directories.find(dirName) != directories.end()) {
                cout << "Enter name of the file -- ";
                cin >> fileName;
                bool found = false;
                for(string f : directories[dirName]) {
                    if(f == fileName) {
                        found = true;
                        break;
                    }
                }
                if(found) cout << "File found" << endl;
                else cout << "File not found" << endl;
            } else {
                cout << "Directory does not exist" << endl;
            }
        }
        else if(choice == 5) {
            for(auto &dir : directories) {
                cout << "Directory: " << dir.first << endl;
                if(dir.second.empty()) cout << "No files" << endl;
                else {
                    for(string f : dir.second) cout << f << " ";
                    cout << endl;
                }
            }
        }

    } while(choice != 6);

    return 0;
}
```