

OPERATING SYSTEMS

(CECSC09 -I)



Submitted By:-

Ashish Kumar

2019UCO1518

INDEX

S.No	TOPIC
1	<u>Practice Linux shell commands.</u>
2	<u>Write C programs using fork(), getpid(), getppid() and exec() system calls.</u>
3	<u>Write a C program to represent a family of processes as a tree.</u>
4	<u>CPU Scheduling Algorithms - FCFS</u>
5	<u>CPU Scheduling Algorithms - SJF</u>
6	<u>CPU Scheduling Algorithms – Round Robin</u>
7	<u>CPU Scheduling Algorithms – preemptive priority scheduling</u>
8	<u>Multilevel Feedback Queue scheduling algorithm.</u>
9	<u>Program to stimulate deadlock detection</u>
10	<u>Program to stimulate deadlock avoidance</u>
11	<u>program to simulate best-fit contiguous memory allocation.</u>
12	<u>Program to stimulate FIFO page replacement algorithm</u>
13	<u>Program to stimulate LRU page replacement algorithm</u>
14	<u>program to simulate Second Chance page replacement algorithm.</u>
15	<u>program to simulate Enhanced Second Chance page replacement algorithm.</u>
16	<u>program to simulate LFU page replacement algorithm.</u>
17	<u>program to simulate FCFS disk scheduling algorithm.</u>
18	<u>program to simulate SSTF disk scheduling algorithm.</u>
19	<u>program to simulate C-SCAN disk scheduling algorithm</u>
20	<u>program to simulate LOOK disk scheduling algorithm.</u>

Program 1 : Linux Shell Commands

mkdir: Used to create a directory if not already exist. It accepts the directory name as an input parameter.

rmdir: It is used to delete a directory if it is empty.

touch: Used to create or update a file.

cat: It is generally used to concatenate the files. It gives the output on the standard output.

more: It is a filter for paging through text one screenful at a time.

cd: Used to change the directory.

touch: Used to create or update a file.

ls: To get the list of all the files or folders.

rm: Used to remove files or directories.

wc: Used to count the number of characters, words in a file.

sort: This command is used to sort the contents of files.

grep: This command is used to search for the specified text in a file.

head: Used to print the first N lines of a file. It accepts N as input and the default value of N is 10.

tail: The tail command displays the last part (10 lines by default) of one or more files or piped data. It can be also used to monitor the file changes in real time

**PROGRAM 2 : Write C programs using fork(), getpid(), getppid() and
exec() system calls.**

CODE:

FILE 1 :

```
#include <iostream>
#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>
using namespace std;
int main(int argc, char *argv[])
{
    cout << "Process Creation and Termination\n";
    pid_t id1, id2;
    id1 = fork();
    id2 = fork();
    if (id1 == 0 && id2 > 0)
    {
        cout << "This is First child Process; pid= " << getpid() << "
            Parent's pid=" << getppid() << endl;
        exit(0);
    }
    else if (id1 > 0 && id2 == 0)
    {
        cout << "This is Second child Process; pid= " << getpid() <
            < "Parent's pid=" << getppid() << endl;
        exit(0);
    }
    else if (id1 == 0 && id2 == 0)
    {
        cout << "This is Third child Process; pid= " << getpid() << "
            Parent's pid=" << getppid() << endl;
        exit(0);
    }
    else if (id1 > 0 && id2 > 0)
    {
        cout << "This is Parent Process; pid= " << getpid() << endl;
        pid_t Id = wait(NULL);
        cout << "Parent Process returns from wait after process "
            << Id << " terminates\n";
        cout << "Calling exec() system call\n";
        char *args[] = {"a", "b", NULL};

        execl("./prog", args);
    }
    else
        cout << "Error\n";
    return 0;
}
```

FILE2 :

```
#include <iostream>
#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>
using namespace std;
int main(int argc, char *argv[])
{
    cout << "This is the replaced file\n";
    cout << "Process Id= " << getpid() << endl;
    return 0;
}
```

PROGRAM : 3 - Write a C program to represent a family of processes as a tree.

Code:

```
#include <iostream>
#include <fstream>
#include <thread>
#include <cstdlib>
#include <list>
#include <chrono>
using namespace std;
using namespace std::chrono;
void withoutThreads() {
    cout << "Without Threads:\n";
    auto start = high_resolution_clock::now(); list<string>
s; bool end = false;
    ifstream inFile;
    ofstream outFile;
    inFile.open("input.txt");
    outFile.open("output.txt");
    if (inFile.fail() || outFile.fail()) {
        cout << "error in opening files\n"; exit(1);
    }
    string line;
    while (getline(inFile, line)) {
        s.push_back(line);
    }
    while (!s.empty()) {
        outFile << s.front() << "\n";
        s.pop_front();
    }
    cout << 1;
    inFile.close();
    outFile.close();
    cout << "done transferring\n";
    auto stop = high_resolution_clock::now(); auto
duration =
        duration_cast<microseconds>(stop - start);
    cout << "Time taken: "
        << duration.count() << " microseconds" << endl;
}
void usingThreads() {
    cout << "With Threads:\n";
    auto start = high_resolution_clock::now();
    list<string> s; bool end = false;
    ifstream inFile;
    ofstream outFile;
    inFile.open("input.txt");
    outFile.open("output.txt");
    auto read = [&]() {
        string line;
        while (getline(inFile, line)) { s.push_back(line); }
        cout << "reading success\n"; end = true;
    };
    auto write = [&]() {
        while (!s.empty()) {
            outFile << s.front() << "\n";
            s.pop_front();
        }
        cout << "writing success\n";
    };

    if (inFile.fail() || outFile.fail()) {
        cout << "error in opening files\n";
        exit(1);
    }
    thread t1(read);
    thread t2(write);
    t2.join();
    t1.join();
    inFile.close();
    outFile.close();
    cout << "done transferring\n";
    auto stop = high_resolution_clock::now(); auto
duration =
        duration_cast<microseconds>(stop - start);
    cout << "Time taken: "
        << duration.count() << " microseconds" << endl;
}
int main() {
    withoutThreads();
    usingThreads();
}
```

OUTPUT:

The screenshot shows a C++ program in Sublime Text. The program reads lines from 'input.txt' and writes them to 'output.txt'. It compares the time taken to do this sequentially (without threads) versus using parallel threads (with threads). The output shows that using threads significantly reduces the time taken, from 2500 microseconds to 0 microseconds.

```
23 while (!s.empty()) {
24     outFile << s.front() << '\n';
25     s.pop_front();
26 }
27 cout << 1;
28 inFile.close();
29 outFile.close();
30 cout << "done transferring\n";
31 auto stop = high_resolution_clock::now(); auto duration =
32     duration_cast<microseconds>(stop - start);
33 cout << "Time taken: "
34     << duration.count() << " microseconds" << endl;
35 }
36 void usingThreads() {
37     cout << "With Threads:\n";
38     auto start = high_resolution_clock::now();
39     list<string> s; bool end = false;
40     ifstream inFile;
41     ofstream outFile;
42     inFile.open("input.txt");
43     outFile.open("output.txt");
44     auto read = [&]() {
45         string line;
46         while (getline(inFile, line)) { s.push_back(line); }
47         cout << "reading success\n"; end = true;
48     };
49     auto write = [&]() {
50         while (!s.empty()) {
51             outFile << s.front() << '\n'; s.pop_front();
52         }
53         cout << "writing success\n";
54     };
55 }
```

Without Threads:
1done transferring
Time taken: 2500 microseconds
With Threads:
reading success
writing success
done transferring
Time taken: 0 microseconds
[Finished in 1.1s]

PROGRAM 4 - CPU SCHEDULING ALGORITHMS – FCFS

Code:

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

struct process {
    int pid;
    int arrival_time;
    int burst_time;
    int start_time;
    int completion_time;
    int turnaround_time;
    int waiting_time;
    int response_time;
};

bool compareArrival(process p1, process p2)
{
    return p1.arrival_time < p2.arrival_time;
}

bool compareID(process p1, process p2)
{
    return p1.pid < p2.pid;
}

int main() {
    int n;
    struct process p[100];
    float avg_turnaround_time;
    float avg_waiting_time;
    float avg_response_time;
    float cpu_utilisation;
    int total_turnaround_time = 0;
    int total_waiting_time = 0;
    int total_response_time = 0;
    int total_idle_time = 0;
    float throughput;
    // fcfs
    cout << setprecision(2) << fixed;
    cin >> n;

    for (int i = 0; i < n; i++) {
        cin >> p[i].arrival_time;
        cin >> p[i].burst_time;
        p[i].pid = i + 1;
        cout << endl;
    }

    sort(p, p + n, compareArrival);
    for (int i = 0; i < n; i++) {
        p[i].start_time = (i == 0) ? p[i].arrival_time : max(p[i - 1].completion_time, p[i].arrival_time);
        p[i].completion_time = p[i].start_time + p[i].burst_time;
        p[i].turnaround_time = p[i].completion_time - p[i].arrival_time;
        p[i].waiting_time = p[i].turnaround_time - p[i].burst_time;
```



```

        p[i].response_time = p[i].start_time - p[i].arrival_time;

        total_turnaround_time += p[i].turnaround_time;
        total_waiting_time += p[i].waiting_time;
        total_response_time += p[i].response_time;
        total_idle_time += (i == 0) ? (p[i].arrival_time) : (p[i].start_time - p[i - 1].completion_time);
    }

    avg_turnaround_time = (float) total_turnaround_time / n;
    avg_waiting_time = (float) total_waiting_time / n;
    avg_response_time = (float) total_response_time / n;
    cpu_utilisation = ((p[n - 1].completion_time - total_idle_time) / (float) p[n - 1].completion_time) * 100;
    throughput = float(n) / (p[n - 1].completion_time - p[0].arrival_time);

    sort(p, p + n, compareID);
    cout << "#P\t" << "AT\t" << "BT\t" << "ST\t" << "CT\t" << "TAT\t" << "WT\t" << "RT\t" << endl;

    for (int i = 0; i < n; i++) {
        cout << p[i].pid << "\t" << p[i].arrival_time << "\t" << p[i].burst_time << "\t" << p[i].start_time << "\t" <<
        p[i].completion_time << "\t" << p[i].turnaround_time << "\t" << p[i].waiting_time << "\t" << p[i].response_time << "\t" <<
        endl;
    }
    cout << "Average Turnaround Time = " << avg_turnaround_time << endl;
    cout << "Average Waiting Time = " << avg_waiting_time << endl;
    cout << "Average Response Time = " << avg_response_time << endl;
    cout << "CPU Utilization = " << cpu_utilisation << "%" << endl;
    cout << "Throughput = " << throughput << " process/unit time" << endl;

return 0;
}

/*

AT - Arrival Time of the process
BT - Burst time of the process
ST - Start time of the process
CT - Completion time of the process
TAT - Turnaround time of the process
WT - Waiting time of the process
RT - Response time of the process

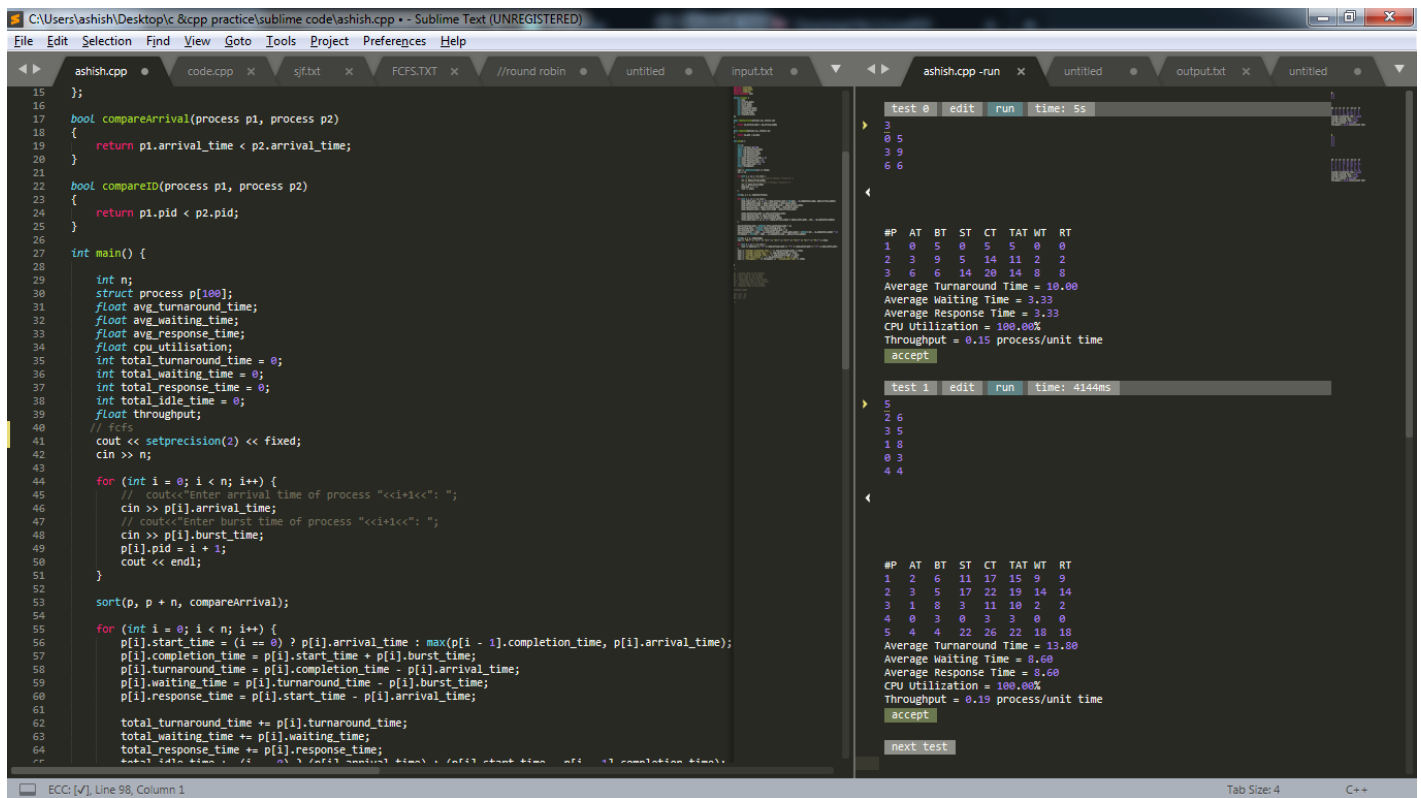
Formulas used:

TAT = CT - AT
WT = TAT - BT
RT = ST - AT

*/

```

Output:



The screenshot shows a C++ program in Sublime Text (UNREGISTERED) running. The code implements a round-robin scheduling algorithm. The output is displayed in two panels on the right.

Test 0 Output:

```
test 0  edit  run  time: 5s
3
0 5
3 9
6 6
```

#P	AT	BT	ST	CT	TAT	WT	RT
1	0	5	0	5	5	0	0
2	3	9	5	14	11	2	2
3	6	6	14	20	14	8	8

Average Turnaround Time = 10.00
Average Waiting Time = 3.33
Average Response Time = 3.33
CPU Utilization = 100.00%
Throughput = 0.15 process/unit time

accept

Test 1 Output:

```
test 1  edit  run  time: 4144ms
5
2 6
3 5
1 8
0 3
4 4
```

#P	AT	BT	ST	CT	TAT	WT	RT
1	2	6	11	17	15	9	9
2	3	5	17	22	19	14	14
3	1	8	3	11	10	2	2
4	0	3	0	3	3	0	0
5	4	4	22	26	22	18	18

Average Turnaround Time = 13.80
Average Waiting Time = 8.60
Average Response Time = 8.60
CPU Utilization = 100.00%
Throughput = 0.19 process/unit time

accept

next test

Program 5 : CPU SCHEDULING ALGORITHMS – SJF

Code :

```
#include <iostream>
#include <algorithm>
#include <iomanip>
#include <string.h>
using namespace std;

struct process {
    int pid;
    int arrival_time;
    int burst_time;
    int start_time;
    int completion_time;
    int turnaround_time;
    int waiting_time;
    int response_time;
};

int main() {

    int n;
    struct process p[100];
    float avg_turnaround_time;
    float avg_waiting_time;
    float avg_response_time;
    float cpu_utilisation;
    int total_turnaround_time = 0;
    int total_waiting_time = 0;
    int total_response_time = 0;
    int total_idle_time = 0;
    float throughput;
    int is_completed[100];
    memset(is_completed, 0, sizeof(is_completed));

    cout << setprecision(2) << fixed;

    cin >> n;

    for (int i = 0; i < n; i++) {
        cin >> p[i].arrival_time;
        cin >> p[i].burst_time;
        p[i].pid = i + 1;
        cout << endl;
    }

    int current_time = 0;
    int completed = 0;
    int prev = 0;

    while (completed != n) {
        int idx = -1;
        int mn = 10000000;
        for (int i = 0; i < n; i++) {
            if (p[i].arrival_time <= current_time && is_completed[i] == 0) {
                if (p[i].burst_time < mn) {
                    mn = p[i].burst_time;
                    idx = i;
                }
            }
            if (p[i].burst_time == mn) {
                if (p[i].arrival_time < p[idx].arrival_time) {

```

```

                                mn = p[i].burst_time;
                                idx = i;
                            }
                        }
                    }
                }
            }
        if (idx != -1) {
            p[idx].start_time = current_time;
            p[idx].completion_time = p[idx].start_time + p[idx].burst_time;
            p[idx].turnaround_time = p[idx].completion_time - p[idx].arrival_time;
            p[idx].waiting_time = p[idx].turnaround_time - p[idx].burst_time;
            p[idx].response_time = p[idx].start_time - p[idx].arrival_time;

            total_turnaround_time += p[idx].turnaround_time;
            total_waiting_time += p[idx].waiting_time;
            total_response_time += p[idx].response_time;
            total_idle_time += p[idx].start_time - prev;

            is_completed[idx] = 1;
            completed++;
            current_time = p[idx].completion_time;
            prev = current_time;
        }
        else {
            current_time++;
        }
    }

    int min_arrival_time = 10000000;
    int max_completion_time = -1;
    for (int i = 0; i < n; i++) {
        min_arrival_time = min(min_arrival_time, p[i].arrival_time);
        max_completion_time = max(max_completion_time, p[i].completion_time);
    }

    avg_turnaround_time = (float) total_turnaround_time / n;
    avg_waiting_time = (float) total_waiting_time / n;
    avg_response_time = (float) total_response_time / n;
    cpu_utilisation = ((max_completion_time - total_idle_time) / (float) max_completion_time) * 100;
    throughput = float(n) / (max_completion_time - min_arrival_time);

    cout << endl << endl;

    cout << "#P\t" << "AT\t" << "BT\t" << "ST\t" << "CT\t" << "TAT\t" << "WT\t" << "RT\t" << endl;

    for (int i = 0; i < n; i++) {
        cout << p[i].pid << "\t" << p[i].arrival_time << "\t" << p[i].burst_time << "\t" << p[i].start_time << "\t" <<
p[i].completion_time << "\t" << p[i].turnaround_time << "\t" << p[i].waiting_time << "\t" << p[i].response_time << "\t" <<
endl;
    }
    cout << "Average Turnaround Time = " << avg_turnaround_time << endl;
    cout << "Average Waiting Time = " << avg_waiting_time << endl;
    cout << "Average Response Time = " << avg_response_time << endl;
    cout << "CPU Utilization = " << cpu_utilisation << "%" << endl;
    cout << "Throughput = " << throughput << " process/unit time" << endl;

}

/*

```

AT - Arrival Time of the process
BT - Burst time of the process
ST - Start time of the process
CT - Completion time of the process
TAT - Turnaround time of the process
WT - Waiting time of the process
RT - Response time of the process

Formulas used:

$TAT = CT - AT$
 $WT = TAT - BT$
 $RT = ST - AT$

*/

Output:

The screenshot shows a C++ program in Sublime Text (UNREGISTERED) running. The code implements a scheduling algorithm. The output is displayed in two panels on the right, labeled 'test 0' and 'test 1'.

test 0 output:

#P	AT	BT	ST	CT	TAT	WT	RT
1	0	5	0	5	5	0	0
2	3	9	5	14	11	2	2
3	6	6	14	20	14	8	8

Average Turnaround Time = 10.00
Average Waiting Time = 3.33
Average Response Time = 3.33
CPU Utilization = 100.00%
Throughput = 0.15 process/unit time

test 1 output:

#P	AT	BT	ST	CT	TAT	WT	RT
1	2	6	3	9	7	1	1
2	5	3	9	12	7	4	4
3	1	8	16	24	23	15	15
4	0	3	0	3	3	0	0
5	4	4	12	16	12	8	8

Average Turnaround Time = 10.40
Average Waiting Time = 5.60
Average Response Time = 5.60
CPU Utilization = 100.00%
Throughput = 0.21 process/unit time

PROGRAM 6 - CPU SCHEDULING ALGORITHMS – ROUND ROBIN

Code:

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

struct process {
    int pid;
    int arrival_time;
    int burst_time;
    int start_time;
    int completion_time;
    int turnaround_time;
    int waiting_time;
    int response_time;
};

bool compare1(process p1, process p2)
{
    return p1.arrival_time < p2.arrival_time;
}

bool compare2(process p1, process p2)
{
    return p1.pid < p2.pid;
}

int main() {

    int n;
    int tq;
    struct process p[100];
    float avg_turnaround_time;
    float avg_waiting_time;
    float avg_response_time;
    float cpu_utilisation;
    int total_turnaround_time = 0;
    int total_waiting_time = 0;
    int total_response_time = 0;
    int total_idle_time = 0;
    float throughput;
    int burst_remaining[100];
    int idx;

    cout << setprecision(2) << fixed;
    cin >> n;
    cin >> tq;

    for (int i = 0; i < n; i++) {
        cin >> p[i].arrival_time;
        cin >> p[i].burst_time;
        burst_remaining[i] = p[i].burst_time;
        p[i].pid = i + 1;
    }

    sort(p, p + n, compare1);

    queue<int> q; // circular queue , round robin
    int current_time = 0;
    q.push(0);
    int completed = 0;
    int mark[100];
```

```
memset(mark, 0, sizeof(mark));
mark[0] = 1;
```

```
while (completed != n) {
    idx = q.front();
    q.pop();
```

```
    if (burst_remaining[idx] == p[idx].burst_time) {
        p[idx].start_time = max(current_time, p[idx].arrival_time);
        total_idle_time += p[idx].start_time - current_time;
        current_time = p[idx].start_time;
    }
```

```
    if (burst_remaining[idx] - tq > 0) {
        burst_remaining[idx] -= tq;
        current_time += tq;
    }
```

```
else {
```

```
    current_time += burst_remaining[idx];
    burst_remaining[idx] = 0;
    completed++;
```

```
    p[idx].completion_time = current_time;
    p[idx].turnaround_time = p[idx].completion_time - p[idx].arrival_time;
    p[idx].waiting_time = p[idx].turnaround_time - p[idx].burst_time;
    p[idx].response_time = p[idx].start_time - p[idx].arrival_time;
```

```
    total_turnaround_time += p[idx].turnaround_time;
    total_waiting_time += p[idx].waiting_time;
    total_response_time += p[idx].response_time;
}
```

```
for (int i = 1; i < n; i++) {
    if (burst_remaining[i] > 0 && p[i].arrival_time <= current_time && mark[i] == 0) {
        q.push(i);
        mark[i] = 1;
    }
}
```

```
if (burst_remaining[idx] > 0) {
    q.push(idx);
}
```

```
if (q.empty()) {
    for (int i = 1; i < n; i++) {
        if (burst_remaining[i] > 0) {
            q.push(i);
            mark[i] = 1;
            break;
        }
    }
}
```

```
}
```

```
avg_turnaround_time = (float) total_turnaround_time / n;
avg_waiting_time = (float) total_waiting_time / n;
avg_response_time = (float) total_response_time / n;
cpu_utilisation = ((p[n - 1].completion_time - total_idle_time) / (float) p[n - 1].completion_time) * 100;
throughput = float(n) / (p[n - 1].completion_time - p[0].arrival_time);
```

```
sort(p, p + n, compare2);
```



```

cout << endl;
cout << "#P\t" << "AT\t" << "BT\t" << "ST\t" << "CT\t" << "TAT\t" << "WT\t" << "RT\t" << endl;

for (int i = 0; i < n; i++) {
    cout << p[i].pid << "\t" << p[i].arrival_time << "\t" << p[i].burst_time << "\t" << p[i].start_time << "\t" <<
p[i].completion_time << "\t" << p[i].turnaround_time << "\t" << p[i].waiting_time << "\t" << p[i].response_time << "\t" << endl;
}
cout << "Average Turnaround Time = " << avg_turnaround_time << endl;
cout << "Average Waiting Time = " << avg_waiting_time << endl;
cout << "Average Response Time = " << avg_response_time << endl;
cout << "CPU Utilization = " << cpu_utilisation << "%" << endl;
cout << "Throughput = " << throughput << " process/unit time" << endl;

```

```

}
/*

```

AT - Arrival Time of the process

BT - Burst time of the process

ST - Start time of the process

CT - Completion time of the process

TAT - Turnaround time of the process

WT - Waiting time of the process

RT - Response time of the process

Formulas used:

TAT = CT - AT

WT = TAT - BT

RT = ST – AT*/

Output:

The screenshot shows a Sublime Text editor window with the following content:

ashish.cpp

```
51 }
52
53 sort(p, p + n, compare1);
54
55 queue<int> q; // circular queue , round robin
56 int current_time = 0;
57 q.push(0);
58 int completed = 0;
59 int mark[100];
60 memset(mark, 0, sizeof(mark));
61 mark[0] = 1;
62
63 while (completed != n) {
64     idx = q.front();
65     q.pop();
66
67     if (burst_remaining[idx] == p[idx].burst_time) {
68         p[idx].start_time = max(current_time, p[idx].arrival_time);
69         total_idle_time += p[idx].start_time - current_time;
70         current_time = p[idx].start_time;
71     }
72
73     if (burst_remaining[idx] - tq > 0) {
74         burst_remaining[idx] -= tq;
75         current_time += tq;
76     }
77     else {
78         current_time += burst_remaining[idx];
79         burst_remaining[idx] = 0;
80         completed++;
81
82         p[idx].completion_time = current_time;
83         p[idx].turnaround_time = p[idx].completion_time - p[idx].arrival_time;
84         p[idx].waiting_time = p[idx].turnaround_time - p[idx].burst_time;
85         p[idx].response_time = p[idx].start_time - p[idx].arrival_time;
86
87         total_turnaround_time += p[idx].turnaround_time;
88         total_waiting_time += p[idx].waiting_time;
89         total_response_time += p[idx].response_time;
90     }
91
92     for (int i = 1; i < n; i++) {
93         if (burst_remaining[i] > 0 && p[i].arrival_time <= current_time && mark[i] == 0) {
94             q.push(i);
95             mark[i] = 1;
96         }
97     }
98     if (burst_remaining[idx] > 0) {
99         q.push(idx);
100     }
}
```

ashish.cpp -run

test 4 edit run time: 5s

```
3 2
0 10
0 5
0 85

#P AT BT ST CT TAT WT RT
1 0 10 0 23 23 13 0
2 0 5 2 15 15 10 2
3 0 85 4 100 100 15 4
Average Turnaround Time = 46.00
Average Waiting Time = 12.67
Average Response Time = 2.00
CPU Utilization = 100.00%
Throughput = 0.03 process/unit time
accept
```

test 5 edit run time: 3180ms

```
5 6
0 8
1 5
1 6
2 4
3 5

#P AT BT ST CT TAT WT RT
1 0 8 0 28 28 20 0
2 1 5 6 11 10 5 5
3 1 6 11 17 16 10 10
4 2 4 17 21 19 15 15
5 3 5 21 26 23 10 10
Average Turnaround Time = 19.20
Average Waiting Time = 13.60
Average Response Time = 9.60
CPU Utilization = 100.00%
Throughput = 0.19 process/unit time
accept
```

next test

ECG [✓], Line 137, Column 1 Tab Size: 4 C++

PROGRAM 7 - CPU SCHEDULING ALGORITHMS – PREEMPTIVE PRIORITY SCHEDULING

Code:

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

struct process {
    int pid;
    int arrival_time;
    int burst_time;
    int priority;
    int start_time;
    int completion_time;
    int turnaround_time;
    int waiting_time;
    int response_time;
};

int main() {

    int n;
    struct process p[100];
    float avg_turnaround_time;
    float avg_waiting_time;
    float avg_response_time;
    float cpu_utilisation;
    int total_turnaround_time = 0;
    int total_waiting_time = 0;
    int total_response_time = 0;
    int total_idle_time = 0;
    float throughput;
    int burst_remaining[100];
    int is_completed[100];
    memset(is_completed, 0, sizeof(is_completed));

    cout << setprecision(2) << fixed;
    cin >> n;

    for (int i = 0; i < n; i++) {
        cin >> p[i].arrival_time;
        cin >> p[i].burst_time;
        cin >> p[i].priority;
        p[i].pid = i + 1;
        burst_remaining[i] = p[i].burst_time;
    }

    int current_time = 0;
    int completed = 0;
    int prev = 0;

    while (completed != n) {
        int idx = -1;
        int mx = -1;
        for (int i = 0; i < n; i++) {
            if (p[i].arrival_time <= current_time && is_completed[i] == 0) {
                if (p[i].priority > mx) {
```

```

        mx = p[i].priority;
        idx = i;
    }
    if (p[i].priority == mx) {
        if (p[i].arrival_time < p[idx].arrival_time) {
            mx = p[i].priority;
            idx = i;
        }
    }
}

if (idx != -1) {
    if (burst_remaining[idx] == p[idx].burst_time) {
        p[idx].start_time = current_time;
        total_idle_time += p[idx].start_time - prev;
    }
    burst_remaining[idx] -= 1;
    current_time++;
    prev = current_time;

    if (burst_remaining[idx] == 0) {
        p[idx].completion_time = current_time;
        p[idx].turnaround_time = p[idx].completion_time - p[idx].arrival_time;
        p[idx].waiting_time = p[idx].turnaround_time - p[idx].burst_time;
        p[idx].response_time = p[idx].start_time - p[idx].arrival_time;

        total_turnaround_time += p[idx].turnaround_time;
        total_waiting_time += p[idx].waiting_time;
        total_response_time += p[idx].response_time;

        is_completed[idx] = 1;
        completed++;
    }
    else {
        current_time++;
    }
}

int min_arrival_time = 10000000;
int max_completion_time = -1;
for (int i = 0; i < n; i++) {
    min_arrival_time = min(min_arrival_time, p[i].arrival_time);
    max_completion_time = max(max_completion_time, p[i].completion_time);
}

avg_turnaround_time = (float) total_turnaround_time / n;
avg_waiting_time = (float) total_waiting_time / n;
avg_response_time = (float) total_response_time / n;
cpu_utilisation = ((max_completion_time - total_idle_time) / (float) max_completion_time) * 100;
throughput = float(n) / (max_completion_time - min_arrival_time);

cout << endl << endl;

cout << "#P\t" << "AT\t" << "BT\t" << "PRI\t" << "ST\t" << "CT\t" << "TAT\t" << "WT\t" << "RT\t" << endl;

for (int i = 0; i < n; i++) {
    cout << p[i].pid << "\t" << p[i].arrival_time << "\t" << p[i].burst_time << "\t" << p[i].priority << "\t" <<
p[i].start_time << "\t" << p[i].completion_time << "\t" << p[i].turnaround_time << "\t" << p[i].waiting_time << "\t" <<
p[i].response_time << "\t" << endl;
}
cout << "Average Turnaround Time = " << avg_turnaround_time << endl;
cout << "Average Waiting Time = " << avg_waiting_time << endl;
cout << "Average Response Time = " << avg_response_time << endl;

```

```
cout << "CPU Utilization = " << cpu_utilisation << "%" << endl;  
cout << "Throughput = " << throughput << " process/unit time" << endl;
```

```
}
```

```
/*
```

AT - Arrival Time of the process

BT - Burst time of the process

ST - Start time of the process

CT - Completion time of the process

TAT - Turnaround time of the process

WT - Waiting time of the process

RT - Response time of the process

Formulas used:

TAT = CT - AT

WT = TAT - BT

RT = ST - AT

```
*/
```

Output:

The screenshot shows a Sublime Text editor window with the following tabs: ashish.cpp, code.cpp, //round robin, untitled, input.txt, ashish.cpp -run, output.txt, and untitled. The main editor displays the C++ code for a round-robin scheduling algorithm. The code includes headers, variable declarations, input handling, and a scheduling loop. The output window on the right shows the results of the program execution.

```
18 int main() {
19     int n;
20     struct process p[100];
21     float avg_turnaround_time;
22     float avg_waiting_time;
23     float avg_response_time;
24     float cpu_utilisation;
25     int total_turnaround_time = 0;
26     int total_waiting_time = 0;
27     int total_response_time = 0;
28     int total_idle_time = 0;
29     float throughput;
30     int burst_remaining[100];
31     int is_completed[100];
32     memset(is_completed, 0, sizeof(is_completed));
33     cout << setprecision(2) << fixed;
34     cin >> n;
35     for (int i = 0; i < n; i++) {
36         cin >> p[i].arrival_time;
37         cin >> p[i].burst_time;
38         cin >> p[i].priority;
39         p[i].pid = i + 1;
40         burst_remaining[i] = p[i].burst_time;
41     }
42     int current_time = 0;
43     int completed = 0;
44     int prev = 0;
45     while (completed != n) {
46         int idx = -1;
47         int mx = -1;
48         for (int i = 0; i < n; i++) {
49             if (p[i].arrival_time <= current_time && is_completed[i] == 0) {
50                 if (p[i].priority > mx) {
51                     mx = p[i].priority;
52                     idx = i;
53                 }
54                 if (p[i].priority == mx) {
55                     if (p[i].arrival_time < p[idx].arrival_time) {
56                         mx = p[i].priority;
57                         idx = i;
58                     }
59                 }
60             }
61         }
62         current_time += p[idx].burst_time;
63         burst_remaining[idx] -= p[idx].burst_time;
64         if (burst_remaining[idx] == 0) {
65             is_completed[idx] = 1;
66             completed++;
67         }
68     }
69     avg_turnaround_time = (total_turnaround_time) / n;
70     avg_waiting_time = (total_waiting_time) / n;
71     avg_response_time = (total_response_time) / n;
72     cpu_utilisation = (current_time - total_idle_time) / current_time;
73     throughput = n / current_time;
74     cout << "Average Turnaround Time = " << avg_turnaround_time << endl;
75     cout << "Average Waiting Time = " << avg_waiting_time << endl;
76     cout << "Average Response Time = " << avg_response_time << endl;
77     cout << "CPU Utilization = " << cpu_utilisation << endl;
78     cout << "Throughput = " << throughput << " process/unit time" << endl;
79     return 0;
80 }
```

test 0 edit run time: 48s

```
5
4 6 2
5 3 1
5 1 3
1 4 2
3 4 5
```

#P	AT	BT	PRI	ST	CT	TAT	WT	RT
1	4	6	2	10	16	12	6	6
2	5	3	1	16	19	14	11	11
3	5	1	3	7	8	3	2	2
4	1	4	2	1	10	9	5	0
5	3	4	5	3	7	4	0	0

Average Turnaround Time = 8.40
Average Waiting Time = 4.80
Average Response Time = 3.80
CPU Utilization = 94.74%
Throughput = 0.28 process/unit time

accept

next test

ECC [✓] Line 42, Column 30 Tab Size: 4 C++

PROGRAM: 8 simulate Multilevel Feedback Queue scheduling algorithm.

Code :

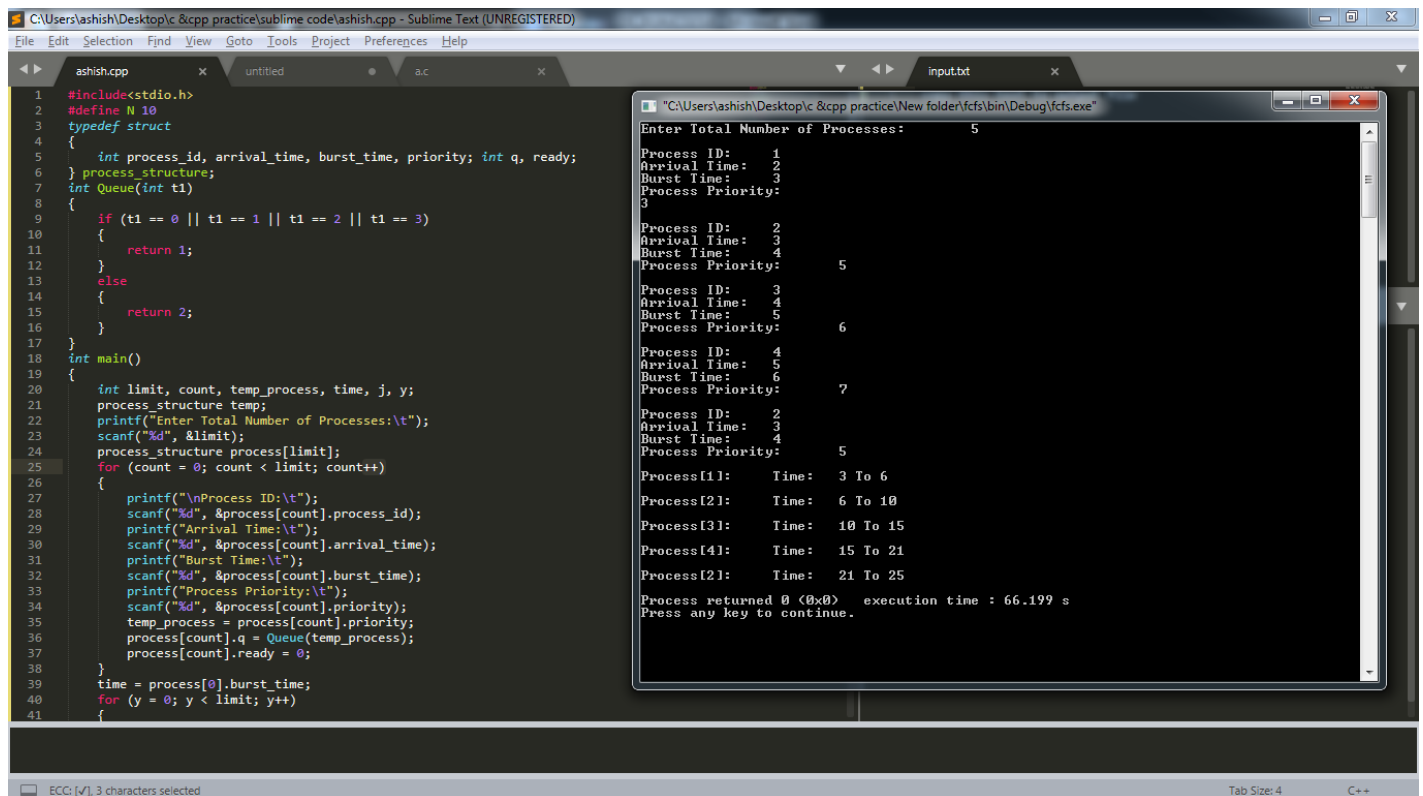
```
#include<stdio.h>
#define N 10
typedef struct
{
    int process_id, arrival_time, burst_time, priority; int q, ready;
} process_structure;
int Queue(int t1)
{
    if (t1 == 0 || t1 == 1 || t1 == 2 || t1 == 3)
    {
        return 1;
    }
    else
    {
        return 2;
    }
}
int main()
{
    int limit, count, temp_process, time, j, y;
    process_structure temp;
    printf("Enter Total Number of Processes:\t");
    scanf("%d", &limit);
    process_structure process[limit];
    for (count = 0; count < limit; count++)
    {
        printf("\nProcess ID:\t");
        scanf("%d", &process[count].process_id);
        printf("Arrival Time:\t");
        scanf("%d", &process[count].arrival_time);
        printf("Burst Time:\t");
        scanf("%d", &process[count].burst_time);
        printf("Process Priority:\t");
        scanf("%d", &process[count].priority);
        temp_process = process[count].priority;
        process[count].q = Queue(temp_process);
        process[count].ready = 0;
    }
    time = process[0].burst_time;
    for (y = 0; y < limit; y++)
    {
        for (count = y; count < limit; count++)
        {
            if (process[count].arrival_time < time)
            {
                process[count].ready = 1;
            }
        }
        for (count = y; count < limit - 1; count++)
        {
            for (j = count + 1; j < limit; j++)
            {
                if (process[count].ready == 1 && process[j].ready == 1)
                {
                    if (process[count].q == 2 && process[j].q == 1)
                    {
                        temp = process[count];
                        process[count] = process[j];
                        process[j] = temp;
                    }
                }
            }
        }
    }
}
```

```

    }
    }
}
for (count = y; count < limit - 1; count++)
{
    for (j = count + 1; j < limit; j++)
    {
        if (process[count].ready == 1 && process[j].ready == 1)
        {
            if (process[count].q == 1 && process[j].q == 1)
            {
                if (process[count].burst_time >
                    process[j].burst_time)
                {
                    temp = process[count];
                    process[count] = process[j];
                    process[j] = temp;
                }
                else
                {
                    break;
                }
            }
        }
    }
}
printf("\nProcess[%d]:\tTime:\t%d To %d\n", process[y].process_id, time, time +
    process[y].burst_time);
time = time + process[y].burst_time;
for (count = y; count < limit; count++)
{
    if (process[count].ready == 1)
    {
        process[count].ready = 0;
    }
}
}
return 0;
}

```


OUTPUT :



The screenshot shows a C++ program in Sublime Text and its execution output in a command prompt window.

Source Code (ashish.cpp):

```
1 #include<stdio.h>
2 #define N 10
3 typedef struct
4 {
5     int process_id, arrival_time, burst_time, priority; int q, ready;
6 } process_structure;
7 int Queue(int t1)
8 {
9     if (t1 == 0 || t1 == 1 || t1 == 2 || t1 == 3)
10    {
11        return 1;
12    }
13    else
14    {
15        return 2;
16    }
17 }
18 int main()
19 {
20     int limit, count, temp_process, time, j, y;
21     process_structure temp;
22     printf("Enter Total Number of Processes:\n");
23     scanf("%d", &limit);
24     process_structure process[limit];
25     for (count = 0; count < limit; count++)
26     {
27         printf("\nProcess ID:\n");
28         scanf("%d", &process[count].process_id);
29         printf("Arrival Time:\n");
30         scanf("%d", &process[count].arrival_time);
31         printf("Burst Time:\n");
32         scanf("%d", &process[count].burst_time);
33         printf("Process Priority:\n");
34         scanf("%d", &process[count].priority);
35         temp_process = process[count].priority;
36         process[count].q = Queue(temp_process);
37         process[count].ready = 0;
38     }
39     time = process[0].burst_time;
40     for (y = 0; y < limit; y++)
41     {
```

Output (Command Prompt):

```
Enter Total Number of Processes: 5
Process ID: 1
Arrival Time: 2
Burst Time: 3
Process Priority: 3
Process ID: 2
Arrival Time: 3
Burst Time: 4
Process Priority: 5
Process ID: 3
Arrival Time: 4
Burst Time: 5
Process Priority: 6
Process ID: 4
Arrival Time: 5
Burst Time: 6
Process Priority: 7
Process ID: 2
Arrival Time: 3
Burst Time: 4
Process Priority: 5
Process[1]: Time: 3 To 6
Process[2]: Time: 6 To 10
Process[3]: Time: 10 To 15
Process[4]: Time: 15 To 21
Process[2]: Time: 21 To 25
Process returned 0 (0x0) execution time : 66.199 s
Press any key to continue.
```

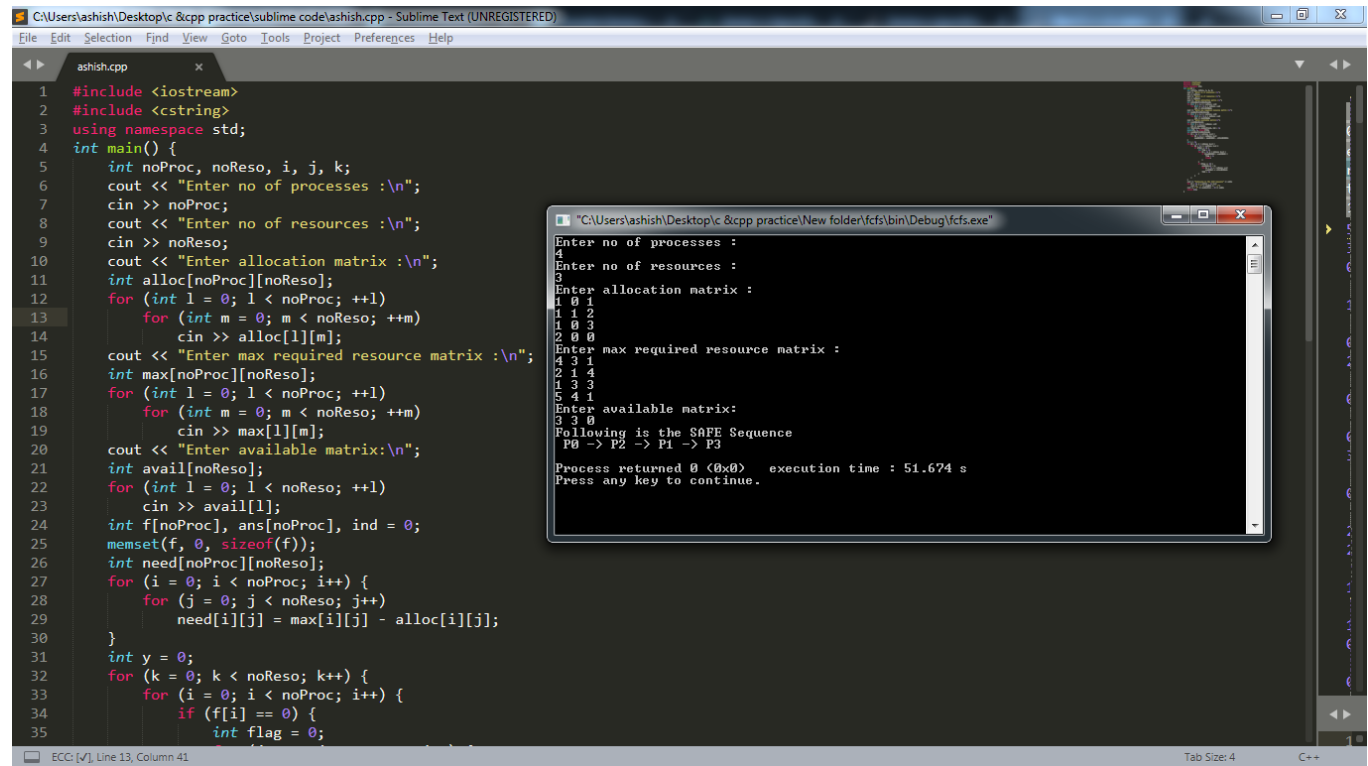
The output shows the input of 5 processes and their details (ID, Arrival Time, Burst Time, Priority). It then displays the execution time for each process in a queue, sorted by priority. The processes are executed in the order: Process 1, Process 2, Process 3, Process 4, and then Process 2 again. The total execution time is 66.199 seconds.

PROGRAM 9 - Program to stimulate deadlock detection

Code:

```
#include <iostream>
#include <cstring>
using namespace std;
int main() {
    int noProc, noReso, i, j, k;
    cout << "Enter no of processes :\n";
    cin >> noProc;
    cout << "Enter no of resources :\n";
    cin >> noReso;
    cout << "Enter allocation matrix :\n";
    int alloc[noProc][noReso];
    for (int l = 0; l < noProc; ++l)
        for (int m = 0; m < noReso; ++m)
            cin >> alloc[l][m];
    cout << "Enter max required resource matrix :\n";
    int max[noProc][noReso];
    for (int l = 0; l < noProc; ++l)
        for (int m = 0; m < noReso; ++m)
            cin >> max[l][m];
    cout << "Enter available matrix:\n";
    int avail[noReso];
    for (int l = 0; l < noReso; ++l)
        cin >> avail[l];
    int f[noProc], ans[noProc], ind = 0;
    memset(f, 0, sizeof(f));
    int need[noProc][noReso];
    for (i = 0; i < noProc; i++) {
        for (j = 0; j < noReso; j++)
            need[i][j] = max[i][j] - alloc[i][j];
    }
    int y = 0;
    for (k = 0; k < noReso; k++) {
        for (i = 0; i < noProc; i++) {
            if (f[i] == 0) {
                int flag = 0;
                for (j = 0; j < noReso; j++) {
                    if (need[i][j] > avail[j]) {
                        flag = 1;
                        break;
                    }
                }
                if (flag == 0) {
                    ans[ind++] = i;
                    for (y = 0; y < noReso; y++)
                        avail[y] += alloc[i][y];
                    f[i] = 1;
                }
            }
        }
    }
    cout << "Following is the SAFE Sequence" << endl;
    for (i = 0; i < noProc - 1; i++)
        cout << " P" << ans[i] << " ->";
    cout << " P" << ans[noProc - 1] << endl;
    return (0);
}
```

Output:



The screenshot shows a C++ program named `ashish.cpp` in the Sublime Text editor. The program implements a deadlock detection algorithm using a Resource Allocation Graph (RAG). It prompts the user to enter the number of processes, resources, allocation matrix, and maximum required resource matrix. It then calculates the available resources and determines a safe sequence of processes that can finish their execution without causing a deadlock.

```
1 #include <iostream>
2 #include <cstring>
3 using namespace std;
4 int main() {
5     int noProc, noReso, i, j, k;
6     cout << "Enter no of processes :\n";
7     cin >> noProc;
8     cout << "Enter no of resources :\n";
9     cin >> noReso;
10    cout << "Enter allocation matrix :\n";
11    int alloc[noProc][noReso];
12    for (int l = 0; l < noProc; ++l)
13        for (int m = 0; m < noReso; ++m)
14            cin >> alloc[l][m];
15    cout << "Enter max required resource matrix :\n";
16    int max[noProc][noReso];
17    for (int l = 0; l < noProc; ++l)
18        for (int m = 0; m < noReso; ++m)
19            cin >> max[l][m];
20    cout << "Enter available matrix:\n";
21    int avail[noReso];
22    for (int l = 0; l < noReso; ++l)
23        cin >> avail[l];
24    int f[noProc], ans[noProc], ind = 0;
25    memset(f, 0, sizeof(f));
26    int need[noProc][noReso];
27    for (i = 0; i < noProc; i++) {
28        for (j = 0; j < noReso; j++)
29            need[i][j] = max[i][j] - alloc[i][j];
30    }
31    int y = 0;
32    for (k = 0; k < noReso; k++) {
33        for (i = 0; i < noProc; i++) {
34            if (f[i] == 0) {
35                int flag = 0;
```

The console window shows the following input and output:

```
Enter no of processes :
4
Enter no of resources :
3
Enter allocation matrix :
4 0 1
1 1 2
1 0 3
2 0 0
Enter max required resource matrix :
4 3 1
2 1 4
1 3 3
5 4 1
Enter available matrix:
3 3 0
Following is the SAFE Sequence
P0 -> P2 -> P1 -> P3
Process returned 0 (0x0)   execution time : 51.674 s
Press any key to continue.
```

PROGRAM 10 - Program to stimulate deadlock avoidance

Code:

```
#include <iostream>
#include <cstring>
using namespace std;
int main() {
    int noProc, noReso, i, j, k;
    cout << "Enter no of processes :";
    cin >> noProc ;
    cout << "Enter no of resources :";
    cin >> noReso;
    cout << "Enter allocation matrix :\n";
    int alloc[noProc][noReso];
    for (int l = 0; l < noProc; ++l)
        for (int m = 0; m < noReso; ++m)
            cin >> alloc[l][m];
    cout << "Enter max resource matrix :\n";
    int max[noProc][noReso];
    for (int l = 0; l < noProc; ++l)
        for (int m = 0; m < noReso; ++m)
            cin >> max[l][m];
    cout << "Enter availability of resources:\n";
    int avail[noReso];
    for (int l = 0; l < noReso; ++l)
        cin >> avail[l];
    int f[noProc], ans[noProc], ind = 0;
    memset(f, 0, sizeof(f));
    int need[noProc][noReso];
    for (i = 0; i < noProc; i++) {
        for (j = 0; j < noReso; j++)
            need[i][j] = max[i][j] - alloc[i][j];
    }
    int finish[noProc], flag = 1;
    int deadlock[noProc];
    memset(finish, 0, sizeof(finish));
    //find need matrix
    for (i = 0; i < noProc; i++) {
        for (j = 0; j < noReso; j++) {
            need[i][j] = max[i][j] - alloc[i][j];
        }
    }
    while (flag) {
        flag = 0;
        for (i = 0; i < noProc; i++) {
            int c = 0;
            for (j = 0; j < noReso; j++)
                if ((finish[i] == 0) && (need[i][j] <= avail[j])) {
                    c++;
                    if (c == noReso) {
                        for (k = 0; k < noReso; k++) {
                            avail[k] +=
                                alloc[i][k];
                            finish[i] = 1;
                            flag = 1;
                        }
                        if (finish[i] == 1)
                            i = noProc;
                    }
                }
        }
    }
}
```

```
j = 0;
flag = 0;
for (i = 0; i < noProc; i++)
    if (finish[i] == 0) {
        deadlock[j++] = i;
        flag = 1;
    }
if (flag == 1) {
    cout << "\n\nSystem is in
Deadlock and the Deadlock process are\n";
    for (i = 0; i < noProc; i++) {
        cout << "P" <<
            deadlock[i] << " ";
    }
    } else {
        cout << "\nNo Deadlock
Occur";
    }
    return (0);
}
```

Output:

Sublime Text (UNREGISTERED) editor showing C++ code for a deadlock simulation. The code prompts for the number of processes (12) and resources (12), then for allocation and max resource matrices. The output window shows the system is in a deadlock state.

```
#include <iostream>
#include <string>
using namespace std;
int main() {
    int noProc, noReso, i, j, k;
    cout << "Enter no of processes :";
    cin >> noProc;
    cout << "Enter no of resources :";
    cin >> noReso;
    cout << "Enter allocation matrix :\n";
    int alloc[noProc][noReso];
    for (int l = 0; l < noProc; ++l)
        for (int m = 0; m < noReso; ++m)
            cin >> alloc[l][m];
    cout << "Enter max resource matrix :\n";
    int max[noProc][noReso];
    for (int l = 0; l < noProc; ++l)
        for (int m = 0; m < noReso; ++m)
            cin >> max[l][m];
    cout << "Enter availability of resources:\n";
    int avail[noReso];
    for (int l = 0; l < noReso; ++l)
        cin >> avail[l];
    int f[noProc], ans[noProc], ind = 0;
    memset(f, 0, sizeof(f));
    int need[noProc][noReso];
    for (i = 0; i < noProc; i++) {
        for (j = 0; j < noReso; j++)
            need[i][j] = max[i][j] - alloc[i][j];
    }
}
```

Enter no of processes and resources : [Cancelled]

System is in Deadlock and the Deadlock process are
P0 P1
Process returned 0 (0x0) execution time : 27.958 s
Press any key to continue.

```
graph TD
    r1[r1]
    r2[r2]
    p0((p0))
    p1((p1))
    deadlock[deadlock]
    p0 --> r1
    p1 --> r1
    p0 --> r2
    p1 --> r2
    deadlock --- r1
    deadlock --- r2
```

Sublime Text (UNREGISTERED) editor showing the same C++ code as above. The output window shows that no deadlock occurs. The RAG diagram below illustrates this state with three processes (p0, p1, p2) and two resources (r1, r2).

```
#include <iostream>
#include <string>
using namespace std;
int main() {
    int noProc, noReso, i, j, k;
    cout << "Enter no of processes :";
    cin >> noProc;
    cout << "Enter no of resources :";
    cin >> noReso;
    cout << "Enter allocation matrix :\n";
    int alloc[noProc][noReso];
    for (int l = 0; l < noProc; ++l)
        for (int m = 0; m < noReso; ++m)
            cin >> alloc[l][m];
    cout << "Enter max resource matrix :\n";
    int max[noProc][noReso];
    for (int l = 0; l < noProc; ++l)
        for (int m = 0; m < noReso; ++m)
            cin >> max[l][m];
    cout << "Enter availability of resources:\n";
    int avail[noReso];
    for (int l = 0; l < noReso; ++l)
        cin >> avail[l];
    int f[noProc], ans[noProc], ind = 0;
    memset(f, 0, sizeof(f));
    int need[noProc][noReso];
    for (i = 0; i < noProc; i++) {
        for (j = 0; j < noReso; j++)
            need[i][j] = max[i][j] - alloc[i][j];
    }
}
```

Enter no of processes and resources : [Cancelled]

No Deadlock Occur
Process returned 0 (0x0) execution time : 22.282 s
Press any key to continue.

```
graph TD
    r1[r1]
    r2[r2]
    p0((p0))
    p1((p1))
    p2((p2))
    no_deadlock[no deadlock]
    p0 --> r1
    p1 --> r1
    p0 --> r2
    p1 --> r2
    p2 --> r2
    no_deadlock --- r1
    no_deadlock --- r2
```

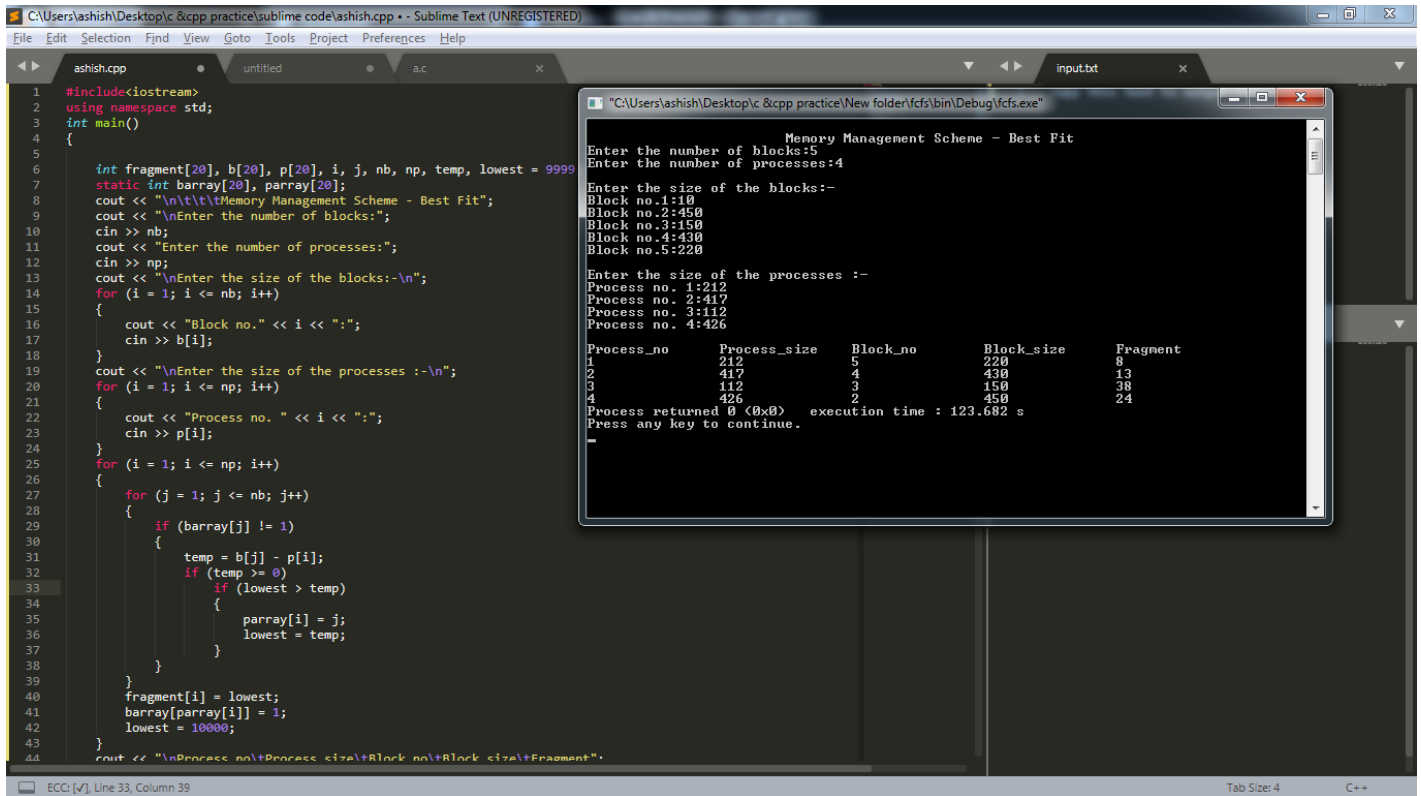
PROGRAM 11 : best-fit contiguous memory allocation.

Code :

```
#include<iostream>
using namespace std;
int main()
{

    int fragment[20], b[20], p[20], i, j, nb, np, temp, lowest = 9999;
    static int barray[20], parray[20];
    cout << "\n\t\tMemory Management Scheme - Best Fit";
    cout << "\nEnter the number of blocks:";
    cin >> nb;
    cout << "Enter the number of processes:";
    cin >> np;
    cout << "\nEnter the size of the blocks:-\n";
    for (i = 1; i <= nb; i++)
    {
        cout << "Block no." << i << ":";
        cin >> b[i];
    }
    cout << "\nEnter the size of the processes :-\n";
    for (i = 1; i <= np; i++)
    {
        cout << "Process no. " << i << ":";
        cin >> p[i];
    }
    for (i = 1; i <= np; i++)
    {
        for (j = 1; j <= nb; j++)
        {
            if (barray[j] != 1)
            {
                temp = b[j] - p[i];
                if (temp >= 0)
                {
                    if (lowest > temp)
                    {
                        parray[i] = j;
                        lowest = temp;
                    }
                }
            }
            fragment[i] = lowest;
            barray[parray[i]] = 1;
            lowest = 10000;
        }
        cout << "\nProcess_no\tProcess_size\tBlock_no\tBlock_size\tFragment";
        for (i = 1; i <= np && parray[i] != 0; i++)
            cout << "\n" << i << "\t\t" << p[i] << "\t\t" << parray[i] << "\t\t" << b[parray[i]] << "\t\t" << fragment[i];
        return 0;
    }
}
```

OUTPUT:



The screenshot shows a C++ program in Sublime Text and its execution output in a console window. The program implements a 'Best Fit' memory management scheme. It takes input for the number of blocks and processes, then assigns processes to blocks based on the best fit algorithm. The output shows the assignment of processes to blocks and the remaining fragment sizes.

```
#include<iostream>
using namespace std;
int main()
{
    int fragment[20], b[20], p[20], i, j, nb, np, temp, lowest = 9999;
    static int barray[20], parray[20];
    cout << "\n\t\t\tMemory Management Scheme - Best Fit";
    cout << "\nEnter the number of blocks:";
    cin >> nb;
    cout << "\nEnter the number of processes:";
    cin >> np;
    cout << "\nEnter the size of the blocks:-\n";
    for (i = 1; i <= nb; i++)
    {
        cout << "Block no. " << i << " ";
        cin >> b[i];
    }
    cout << "\nEnter the size of the processes :-\n";
    for (i = 1; i <= np; i++)
    {
        cout << "Process no. " << i << " ";
        cin >> p[i];
    }
    for (i = 1; i <= np; i++)
    {
        for (j = 1; j <= nb; j++)
        {
            if (barray[j] != 1)
            {
                temp = b[j] - p[i];
                if (temp >= 0)
                {
                    if (lowest > temp)
                    {
                        parray[i] = j;
                        lowest = temp;
                    }
                }
            }
        }
        fragment[i] = lowest;
        barray[parray[i]] = 1;
        lowest = 10000;
    }
    cout << "\nProcess no\tProcess size\tBlock no\tBlock size\tFragment";
```

Memory Management Scheme - Best Fit

Enter the number of blocks:5

Enter the number of processes:4

Enter the size of the blocks:-

Block no.1:10

Block no.2:450

Block no.3:150

Block no.4:430

Block no.5:220

Enter the size of the processes :-

Process no. 1:212

Process no. 2:417

Process no. 3:112

Process no. 4:426

Process_no	Process_size	Block_no	Block_size	Fragment
1	212	5	220	8
2	417	4	430	13
3	112	3	150	38
4	426	2	450	24

Process returned 0 (0x0) execution time : 123.682 s

Press any key to continue.

PROGRAM 12 - Program to stimulate FIFO page replacement algorithm

Code :

```
#include <iostream>
#include <stdio.h>

using namespace std;

//only give positive nos. as input
int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
#ifdef ONLINE_JUDGE
    freopen("input.txt", "r", stdin);
    freopen("error.txt", "w", stderr);
    freopen("output.txt", "w", stdout);
#endif

    int miss = 0, hits = 0;

    int arr[5], n, i, l, page[40];
    l = 4; //no of the pages

    cin >> n;

    for (i = 0; i < n; i++)
    {
        cin >> page[i];
    }

    cout << "Initial status : ";
    for (i = 0; i < l; i++)
    {
        arr[i] = -1;
        cout << arr[i] << " ";
    }
    cout << endl;
    int end_index;
    int j = 0;
    int count = 0, flag = 0;
    for (i = 0; i < n; i++)
    {
        int hit = 0;

        for (j = 0; j < l; j++)
        {
            if (page[i] == arr[j])
            {
                hits++;
                cout << "Page hit : ";
                hit = 1;
                break;
            }
            else if (arr[j] == -1)
                end_index = j;
            else
                end_index = l + 1;
        }

        if (hit == 0 && end_index <= l - 1) //only for the first element
        {
            miss++;
        }
    }
}
```



```

        cout << "Page fault/miss (memory not full) : ";
        count = count + 1;
        arr[end_index] = page[i];
    }
    else if (hit == 0 && end_index == l + 1) //memory full
    {
        miss++;
        cout << "Page fault/miss (memory full) : ";
        count = count + 1;
        for (j = 0; j < l - 1; j++)
        {
            arr[j] = arr[j + 1];
        }
        arr[j] = page[i];
    }

    for (int k = 0; k < l; k++)
    {
        cout << arr[k] << " ";
    }
    cout << endl;

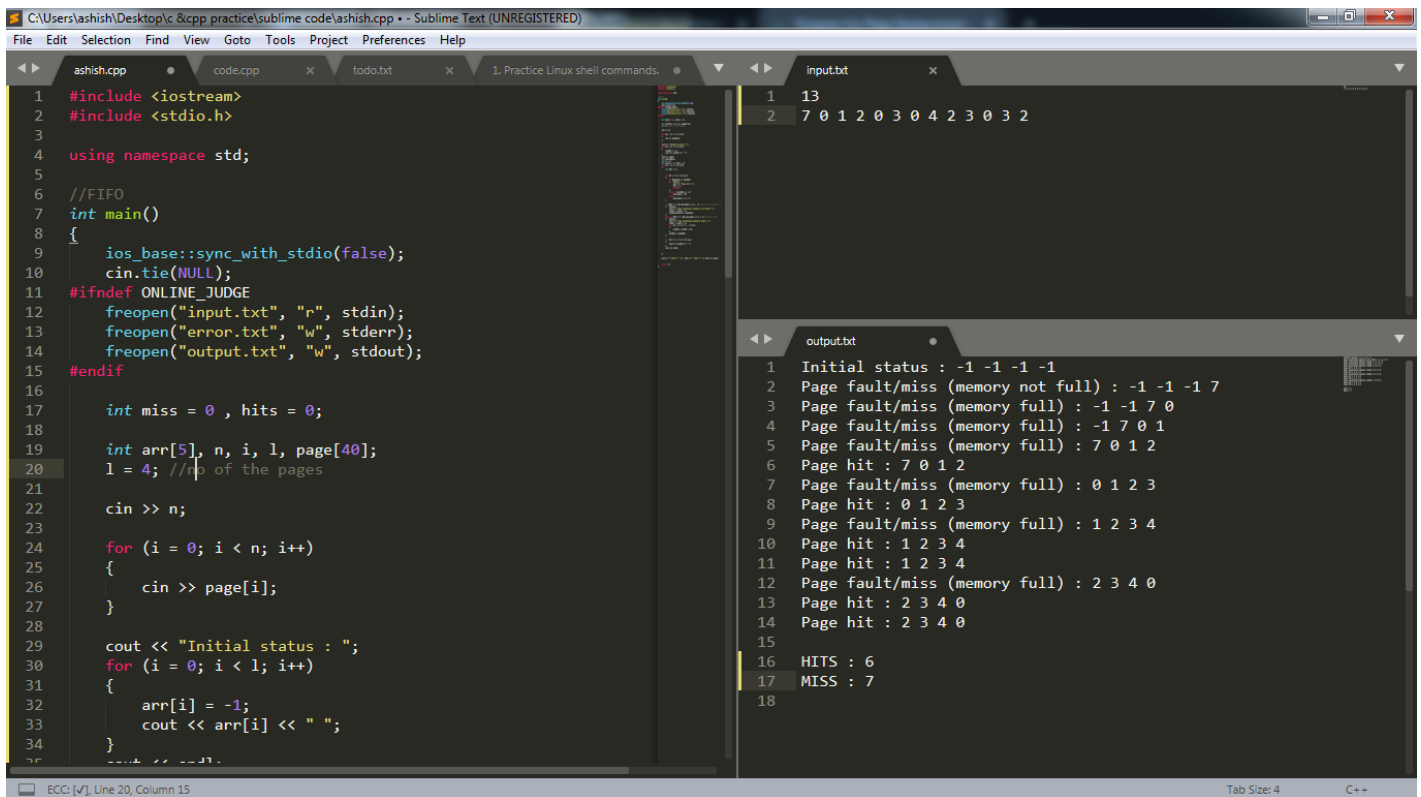
}

cout << "\nHITS : " << hits << " MISS : " << miss << endl;

return 0;
}

```

Output:



The screenshot shows a Sublime Text editor window with the following content:

```
C:\Users\ashish\Desktop\c & cpp practice\sublime code\ashish.cpp - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

ashish.cpp
1 #include <iostream>
2 #include <stdio.h>
3
4 using namespace std;
5
6 //FIFO
7 int main()
8 {
9     ios_base::sync_with_stdio(false);
10    cin.tie(NULL);
11    #ifndef ONLINE_JUDGE
12        freopen("input.txt", "r", stdin);
13        freopen("error.txt", "w", stderr);
14        freopen("output.txt", "w", stdout);
15    #endif
16
17    int miss = 0, hits = 0;
18
19    int arr[5], n, i, l, page[40];
20    l = 4; //no. of the pages
21
22    cin >> n;
23
24    for (i = 0; i < n; i++)
25    {
26        cin >> page[i];
27    }
28
29    cout << "Initial status : ";
30    for (i = 0; i < l; i++)
31    {
32        arr[i] = -1;
33        cout << arr[i] << " ";
34    }
35    cout << endl;
36
37    while (l < n)
38    {
39        if (arr[l] == -1)
40        {
41            hits++;
42            arr[l] = page[l];
43        }
44        else
45        {
46            miss++;
47            arr[l] = -1;
48        }
49        l++;
50    }
51
52    cout << "HITS : " << hits << endl;
53    cout << "MISS : " << miss << endl;
54
55    return 0;
56 }
```

The output.txt file contains the following text:

```
1 13
2 7 0 1 2 0 3 0 4 2 3 0 3 2
3
4 Initial status : -1 -1 -1 -1
5 Page fault/miss (memory not full) : -1 -1 -1 7
6 Page fault/miss (memory full) : -1 -1 7 0
7 Page fault/miss (memory full) : -1 7 0 1
8 Page fault/miss (memory full) : 7 0 1 2
9 Page hit : 7 0 1 2
10 Page fault/miss (memory full) : 0 1 2 3
11 Page hit : 0 1 2 3
12 Page fault/miss (memory full) : 1 2 3 4
13 Page hit : 1 2 3 4
14 Page fault/miss (memory full) : 2 3 4 0
15 Page hit : 2 3 4 0
16
17 HITS : 6
18 MISS : 7
```

The status bar at the bottom indicates: ECG [✓], Line 20, Column 15. Tab Size: 4. C++.

PROGRAM 13 - Program to stimulate LRU page replacement algorithm

Code:

```
#include<iostream>
#include<stdio.h>

using namespace std;
int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
#ifdef ONLINE_JUDGE
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
#endif

    int i, n, page[50], arr[5], k, s, j, large, large_index, dist[5], hit = 0, found = 0, f;
    int nhit = 0, nmiss = 0;
    int m = 0;
    cin >> n;
    f = 4 ;
    for (i = 0; i < n; i++)
        cin >> page[i];

    int finish = 0; //keeps track of the last frame that was inserted into the page table
    int pointer = 0; //keeps track of the last location filled in the page table

    //initializing the page table with -1
    for (i = 0; i < f; i++)
        arr[i] = -1;

    //filling the page table
    for (i = 0; i < n; i++) //iterating through the frames in physical memory
    {
        cout << "processing : " << page[i];
        for (j = 0; j <= i; j++) //iterating through page table to check if the input frame is already present in the page
table or not
        {
            if (page[i] != arr[j])
                hit = 0;
            else
            {
                hit = 1;
                nhit = nhit + 1;
                cout << " Page Hit! ";
                break;
            }
        }
        if (hit == 0) //if not presnt i.e. page miss ,then inserted into the page table
        {
            //cout << "\n j is:" << j;
            nmiss = nmiss + 1;
            cout << " Page Miss! ";
            arr[pointer] = page[i];
            pointer = pointer + 1;
        }

        //Displaying the current state of the array!

        for (k = 0; k < f; k++)
            cout << arr[k] << " ";
```

```

        cout << "\n";

        //checking if all locations of the page table have been filled
        if (pointer == f)
        {
            finish = i + 1;
            cout << "\nCompleted filling the frames\n\n";
            break;
        }
    }

    /*-----*/

    hit = 0;
    found = 0;

//optimal page replacement
    for (i = finish; i < n; i++) // traversing the frames
    {
        cout << "processing : " << page[i];
        for (j = 0; j < f; j) //traversing the page table
        {
            if (page[i] != arr[j])
            {
                j = j + 1;
                hit = 0;

            }
            else
            {
                hit = 1;
                break;
            }
        }

        if (hit == 1)
        {
//do nothing

            cout << " Page hit! ";
            nhit = nhit + 1;
        }
        else
        {
            //Page fault
            cout << " Page Miss! ";
            nmiss = nmiss + 1;

            int index = i;
            for (k = 0; k < f; k++)
            {

                // checking each element of the page table with the remaing array of frames, to check
                // for the one with the longest forward distance
                for (m = index; m > 0; m--)
                {
                    if (arr[k] == page[m])
                    {
                        dist[k] = index - m;
                        //      cout << "\n m is " << m;
                        //      cout << "\n" << arr[k] << " allotted a distance " << int(dist[k]);

                        found = 1;
                        break;
                    }
                }
            }
        }
    }

```

```

        }
    }
    if (found == 0) //not found in frame array ,assigned a very large value
    {
        dist[k] = 99;
        //cout << "\n" << arr[k] << " alloted a distance 99";
    }
}

//finding the one with largest backward distance and then replacing it
large = dist[0];
large_index = 0;
for (s = 1; s < f; s++)
{
    if (large < dist[s])
    {
        large = dist[s];
        large_index = s;
    }
}
//cout << "\n" << arr[large_index] << " stands with largest distance";
arr[large_index] = page[i];

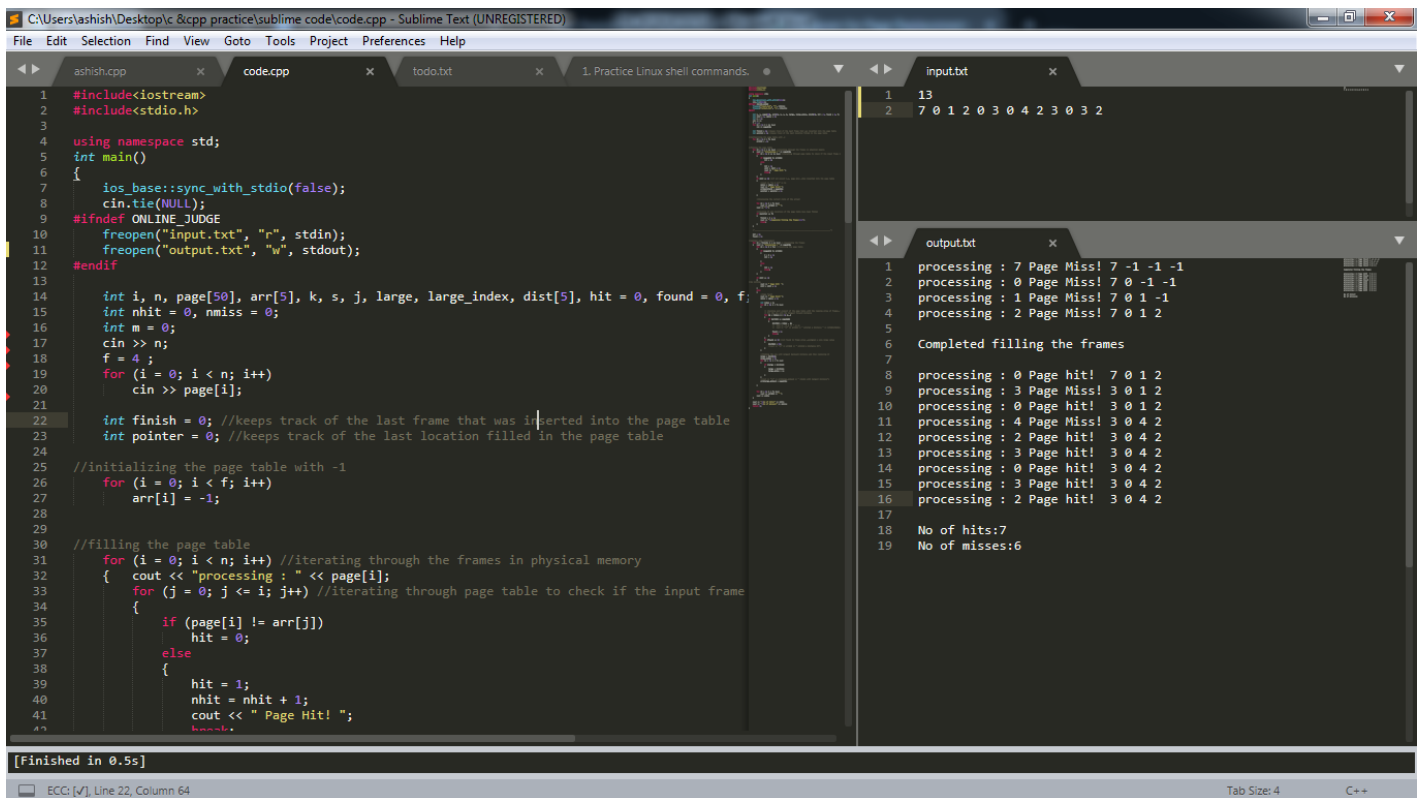
}

for (k = 0; k < f; k++)
    cout << arr[k] << " ";
cout << endl;
}

cout << "\nNo of hits:" << nhit;
cout << "\nNo of misses:" << nmiss;
return 0;
}

```

Output:



The screenshot shows a Sublime Text editor window with the following content:

code.cpp

```
1 #include<iostream>
2 #include<stdio.h>
3
4 using namespace std;
5 int main()
6 {
7     ios_base::sync_with_stdio(false);
8     cin.tie(NULL);
9     #ifndef ONLINE_JUDGE
10     freopen("input.txt", "r", stdin);
11     freopen("output.txt", "w", stdout);
12     #endif
13
14     int i, n, page[50], arr[5], k, s, j, large, large_index, dist[5], hit = 0, found = 0, f;
15     int nhit = 0, nmiss = 0;
16     int m = 0;
17     cin >> n;
18     f = 4;
19     for (i = 0; i < n; i++)
20         cin >> page[i];
21
22     int finish = 0; //keeps track of the last frame that was inserted into the page table
23     int pointer = 0; //keeps track of the last location filled in the page table
24
25     //initializing the page table with -1
26     for (i = 0; i < f; i++)
27         arr[i] = -1;
28
29
30     //filling the page table
31     for (i = 0; i < n; i++) //iterating through the frames in physical memory
32     {
33         cout << "processing : " << page[i];
34         for (j = 0; j <= i; j++) //iterating through page table to check if the input frame
35         {
36             if (page[i] != arr[j])
37                 hit = 0;
38             else
39             {
40                 hit = 1;
41                 nhit = nhit + 1;
42                 cout << " Page Hit! ";
43             }
44         }
45     }
```

input.txt

```
1 13
2 7 0 1 2 0 3 0 4 2 3 0 3 2
```

output.txt

```
1 processing : 7 Page Miss! 7 -1 -1 -1
2 processing : 0 Page Miss! 7 0 -1 -1
3 processing : 1 Page Miss! 7 0 1 -1
4 processing : 2 Page Miss! 7 0 1 2
5
6 Completed filling the frames
7
8 processing : 0 Page hit! 7 0 1 2
9 processing : 3 Page Miss! 3 0 1 2
10 processing : 0 Page hit! 3 0 1 2
11 processing : 4 Page Miss! 3 0 4 2
12 processing : 2 Page hit! 3 0 4 2
13 processing : 3 Page hit! 3 0 4 2
14 processing : 0 Page hit! 3 0 4 2
15 processing : 3 Page hit! 3 0 4 2
16 processing : 2 Page hit! 3 0 4 2
17
18 No of hits:7
19 No of misses:6
```

[Finished in 0.5s]

ECC: [✓] Line 22, Column 64 Tab Size: 4 C++

PROGRAM 14 - Second Chance page replacement algorithm.

CODE :

```
#include<stdio.h>

int n, nf;
int in[100];
int p[50];
int hit = 0;
int i, j, k;
int pgfaultcnt = 0;
void getData()
{
    printf("\nEnter length of page reference sequence:");
    scanf("%d", &n);
    printf("\nEnter the page reference sequence:");
    for (i = 0; i < n; i++)
        scanf("%d", &in[i]);
    printf("\nEnter no of frames:");
    scanf("%d", &nf);
}

void initialize()
{
    pgfaultcnt = 0;
    for (i = 0; i < nf; i++)
        p[i] = 9999;
}

int isHit(int data)
{
    hit = 0;
    for (j = 0; j < nf; j++)
    {
        if (p[j] == data)
        {
            hit = 1;
            break;
        }
    }
    return hit;
}

int getHitIndex(int data)
{
    int hitind;
    for (k = 0; k < nf; k++)
    {
        if (p[k] == data)
        {
            hitind = k;
            break;
        }
    }

    return hitind;
}

void dispPages() {
    for (k = 0; k < nf; k++)

    {
        if (p[k] != 9999)
            printf(" %d", p[k]);

    }
}
```

```

}
void dispPgFaultCnt() {
    printf("\nTotal no of page faults:%d", pgfaultcnt);
}
int main() {

#ifdef ONLINE_JUDGE
    freopen("input.txt", "r", stdin);
    freopen("error.txt", "w", stderr);
    freopen("output.txt", "w", stdout);
#endif

    getData();
    int usedbit[50];
    int victimptr = 0;
    initialize();
    for (i = 0; i < nf; i++)

        usedbit[i] = 0;
    for (i = 0; i < n; i++)

    {
        printf("\nFor %d:", in[i]);
        if (isHit(in[i]))

        {
            printf("No page fault!");
            int hitindex = getHitIndex(in[i]);
            if (usedbit[hitindex] == 0)
                usedbit[hitindex] = 1;

        }
        else

        {
            pgfaultcnt++;
            if (usedbit[victimptr] == 1)

            {
                do

                {
                    usedbit[victimptr] = 0;
                    victimptr++;
                    if (victimptr == nf)
                        victimptr = 0;

                }
                while (usedbit[victimptr] != 0);

            }
            if (usedbit[victimptr] == 0)

            {
                p[victimptr] = in[i];
                usedbit[victimptr] = 1;
                victimptr++;

            }

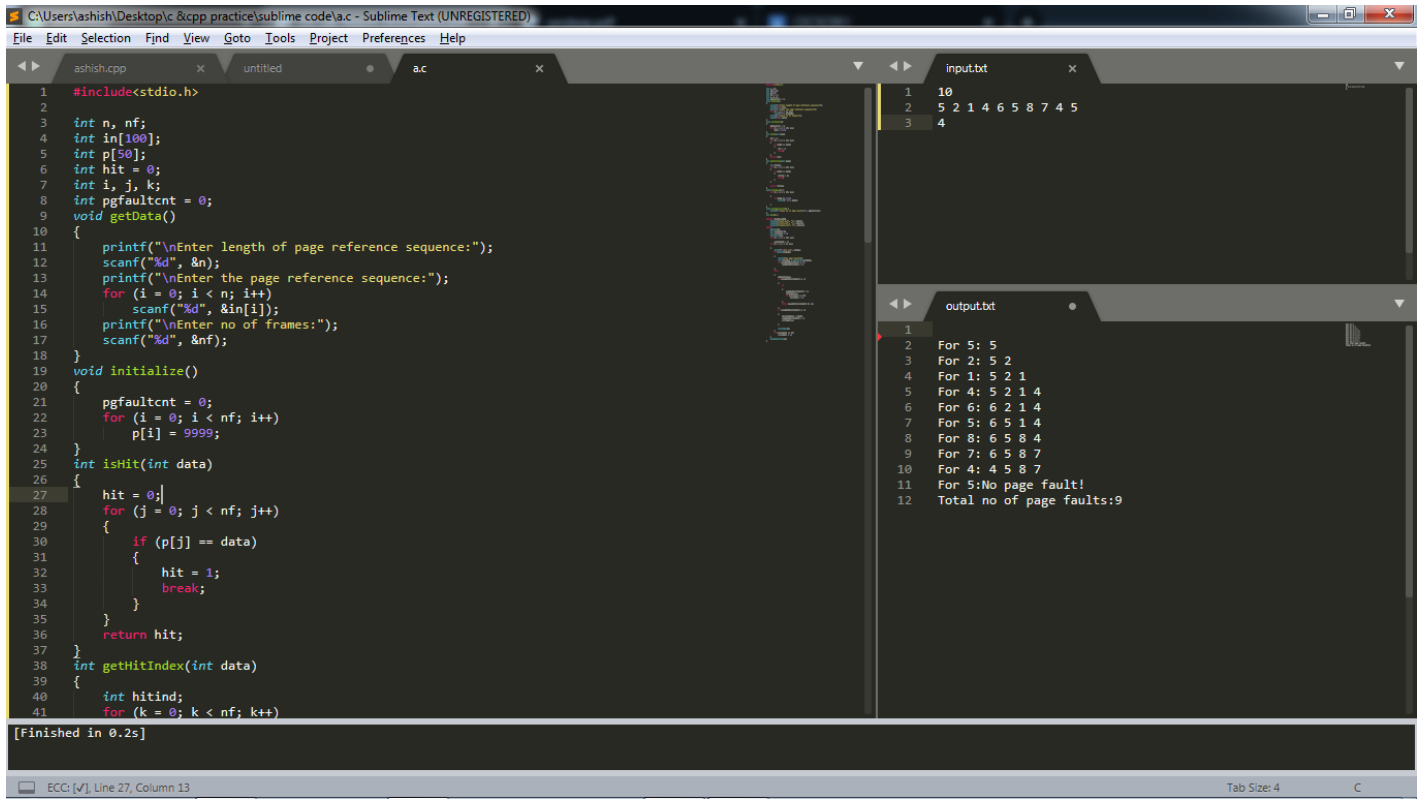
            dispPages();
        }
        if (victimptr == nf)
    }

```



```
        victimptr = 0;
    }
    dispPgFaultCnt();
}
```

OUTPUT :



PROGRAM : 15 Enhanced Second Chance page replacement algorithm.

CODE :

```
#include <iostream>
using namespace std;
void SecondChance_page_replacement(int n, int no_of_frames, int pages[], bool modified[])
{
    int pointer = 0;
    int page_faults = 0;
    int frames[no_of_frames];
    bool ref[no_of_frames] = {0};
    bool mod[no_of_frames] = {0};
    for (int i = 0; i < no_of_frames; ++i)
        frames[i] = -1;
    for (int i = 0; i < n; i++)
    {
        int j;
        for (j = 0; j < no_of_frames; j++)
        {
            if (frames[j] == -1)
            {
                page_faults++;
                frames[j] = pages[i];
                mod[j] = modified[i];
                cout << pages[i] << ": Page fault\t\t";
                break;
            }
            else if (frames[j] == pages[i])
            {
                ref[j] = 1;
                cout << pages[i] << ": Page hit\t\t";
                break;
            }
        }
        if (j == no_of_frames)
        {
            cout << "ENTERED FOR REPLACEMENT\n";
            int k = 0;

            while (true)
            {
                while (k < no_of_frames)
                {
                    if (ref[pointer] == 0 && mod[pointer] == 0)
                    {
                        cout << "00 FOUND\n";
                        frames[pointer] = pages[i];
                        mod[pointer] = modified[i];
                        pointer = (pointer + 1) % no_of_frames;
                        cout << pages[i] << ": Page fault\t\t";
                        break;
                    }
                    pointer = (pointer + 1) % no_of_frames;
                    k++;
                }
                if (k == no_of_frames)
                {
                    cout << "NO 00 FOUND\n";
                    while (k--)
                    {
                        if (ref[pointer] == 0 && mod[pointer] == 1)
```

```

        {
            cout << "01 FOUND\n";
            frames[pointer] = pages[i];
            mod[pointer] = modified[i];
            pointer = (pointer + 1) % no_of_frames;
            cout << pages[i] << ": Page fault\t\t";
            page_faults++;
            break;
        }
        else
        {
            ref[pointer] = 0;
            pointer = (pointer + 1) % no_of_frames;
        }
    }
    if (k != 0)
        break;
}
else
    break;

}

}

}

```

PROGRAM : 16 : LFU page replacement algorithm.

CODE :

```
#include <bits/stdc++.h>
using namespace std;
#define N 100
#define endl "\n"
int P[N], F[N], A[N], T[N];
int frames, pages, hits = 0;
void LFU() {
    int m, n, page, flag, k, minimum_time, temp;
    for (m = 0; m < pages; m++) {
        A[P[m]]++;
        T[P[m]] = m;
        flag = 1;
        k = F[0];
        for (n = 0; n < frames; n++) {
            if (F[n] == -1 || F[n] == P[m]) {
                if (F[n] != -1) {
                    hits++;
                }
                flag = 0;
                F[n] = P[m];
                break;
            }
            if (A[k] > A[F[n]]) {
                k = F[n];
            }
        }
        if (flag) {
            minimum_time = 25;
            for (n = 0; n < frames; n++) {
                if (A[F[n]] == A[k] && T[F[n]] < minimum_time) {
                    temp = n;
                    minimum_time = T[F[n]];
                }
            }
            A[F[temp]] = 0;
            F[temp] = P[m];
        }
        for (n = 0; n < frames; n++) {
            cout << F[n] << "\t";
        }
        cout << endl;
    }
    cout << "Total hits : " << hits << endl;
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
#ifdef ONLINE_JUDGE
    freopen("input.txt", "r", stdin);
    freopen("error.txt", "w", stderr);
    freopen("output.txt", "w", stdout);
#endif

    cin >> pages;

    cin >> frames;
    memset(F, -1, sizeof(F));
```

```

memset(A, 0, sizeof(A));

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

    cin >> P[i];
}
cout << endl;
LFU();
return 0;
}

```

OUTPUT :

```

CAUsers\ashish\Desktop\c & cpp practice\sublime code\ashish.cpp - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

ashish.cpp x untitled a.c x
5 int P[N], F[N], A[N], T[N];
6 int frames, pages, hits = 0;
7 void LFU() {
8     int m, n, page, flag, k, minimum_time, temp;
9     for (m = 0; m < pages; m++) {
10         A[P[m]]++;
11         T[P[m]] = m;
12         flag = 1;
13         k = F[0];
14         for (n = 0; n < frames; n++) {
15             if (F[n] == -1 || F[n] == P[m]) {
16                 if (F[n] != -1) {
17                     hits++;
18                 }
19                 flag = 0;
20                 F[n] = P[m];
21                 break;
22             }
23             if (A[k] > A[F[n]]) {
24                 k = F[n];
25             }
26         }
27         if (flag) {
28             minimum_time = 25;
29             for (n = 0; n < frames; n++) {
30                 if (A[F[n]] == A[k] && T[F[n]] < minimum_time) {
31                     {
32                         temp = n;
33                         minimum_time = T[F[n]];
34                     }
35                 }
36             }
37             A[F[temp]] = 0;
38             F[temp] = P[m];
39         }
40         for (n = 0; n < frames; n++) {
41             cout << F[n] << "\t";
42         }
43         cout << endl;
44     }
45     cout << "Total hits : " << hits << endl;
46 }

input.txt
1 10
2 4
3 1 3 5 2 6 7 4 2 1 4

output.txt
1
2 2 -1 -1 -1 -1
3 2 1 -1 -1 -1
4 2 1 4 -1 -1
5 2 1 4 6 -1
6 2 1 4 6 5
7 8 1 4 6 5
8 8 7 4 6 5
9 8 7 4 6 5
10 8 7 4 6 5
11 8 7 4 6 5
12
13 Total hits : 3
14

[Finished in 2.0s]
ECC: [M] Line 33, Column 44 Tab Size: 4 C++

```

PROGRAM 17 : FCFS disk scheduling algorithm.

CODE :

```
#include <bits/stdc++.h>
using namespace std;
void FCFSdiskScheduling(int request[], int no_of_requests, int initial_head)
{
    int seek_time = 0, i;
    cout << "\nSeek Sequence is \n";
    printf("%d", initial_head );
    for (i = 0; i < no_of_requests; i++)
    {
        if (i == no_of_requests - 1)
            printf("-> %d\n", request[i] );
        else
            printf("-> %d", request[i] );
        seek_time += abs(request[i] - initial_head);
        initial_head = request[i];
    }
    printf("Seek Time: %d", seek_time);
}
int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
#ifdef ONLINE_JUDGE
    freopen("input.txt", "r", stdin);
    freopen("error.txt", "w", stderr);
    freopen("output.txt", "w", stdout);
#endif

    int i, no_of_requests, initial_head;

    scanf("%d", &no_of_requests);
    int request[no_of_requests];

    for (i = 0; i < no_of_requests; ++i)
    {
        scanf("%d", &request[i]);
    }

    scanf("%d", &initial_head);
    FCFSdiskScheduling(request, no_of_requests, initial_head);
    return 0;
}
```

OUTPUT :

The screenshot shows a Sublime Text editor window with the following content:

```
CA\Users\ashish\Desktop\c & cpp practice\sublime code\ashish.cpp - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

ashish.cpp
1 #include <bits/stdc++.h>
2 using namespace std;
3 void FCFSdiskScheduling(int request[], int no_of_requests, int initial_head)
4 {
5     int seek_time = 0, i;
6     cout << "\nSeek Sequence is \n";
7     printf("%d", initial_head);
8     for (i = 0; i < no_of_requests; i++)
9     {
10         if (i == no_of_requests - 1)
11             printf("-> %d\n", request[i]);
12         else
13             printf("-> %d", request[i]);
14         seek_time += abs(request[i] - initial_head);
15         initial_head = request[i];
16     }
17     printf("Seek Time: %d", seek_time);
18 }
19 int main()
20 {
21     ios_base::sync_with_stdio(false);
22     cin.tie(NULL);
23     #ifndef ONLINE_JUDGE
24     freopen("input.txt", "r", stdin);
25     freopen("error.txt", "w", stderr);
26     freopen("output.txt", "w", stdout);
27     #endif
28     int i, no_of_requests, initial_head;
29
30     scanf("%d", &no_of_requests);
31     int request[no_of_requests];
32
33     for (i = 0; i < no_of_requests; ++i)
34     {
35         scanf("%d", &request[i]);
36     }
37
38     scanf("%d", &initial_head);
39     FCFSdiskScheduling(request, no_of_requests, initial_head);
40 }
41
[Finished in 2.0s]
```

The output files are shown in the right pane:

input.txt

```
1 8
2 176 79 34 60 92 11 41 114
3 50
```

output.txt

```
1
2 Seek Sequence is
3 50-> 176-> 79-> 34-> 60-> 92-> 11-> 41-> 114
4
5 Seek Time: 510
```

At the bottom, the status bar shows: ECC: [✓] Line 30, Column 1 Tab Size: 4 C++

PROGRAM 18 : SSTF disk scheduling algorithm.

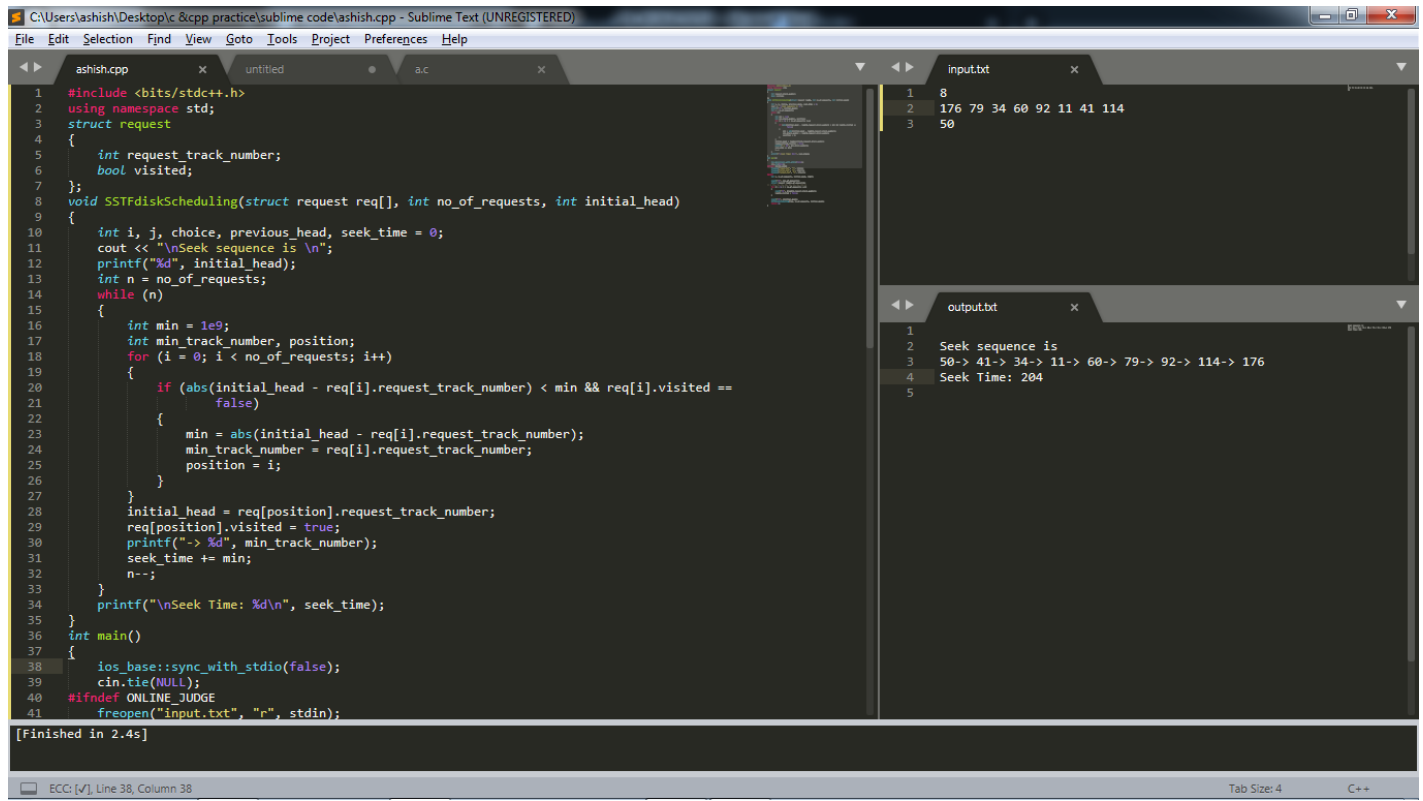
CODE :

```
#include <bits/stdc++.h>
using namespace std;
struct request
{
    int request_track_number;
    bool visited;
};
void SSTFDiskScheduling(struct request req[], int no_of_requests, int initial_head)
{
    int i, j, choice, previous_head, seek_time = 0;
    cout << "\nSeek sequence is \n";
    printf("%d", initial_head);
    int n = no_of_requests;
    while (n)
    {
        int min = 1e9;
        int min_track_number, position;
        for (i = 0; i < no_of_requests; i++)
        {
            if (abs(initial_head - req[i].request_track_number) < min && req[i].visited ==
                false)
            {
                min = abs(initial_head - req[i].request_track_number);
                min_track_number = req[i].request_track_number;
                position = i;
            }
        }
        initial_head = req[position].request_track_number;
        req[position].visited = true;
        printf("-> %d", min_track_number);
        seek_time += min;
        n--;
    }
    printf("\nSeek Time: %d\n", seek_time);
}
int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
#ifdef ONLINE_JUDGE
    freopen("input.txt", "r", stdin);
    freopen("error.txt", "w", stderr);
    freopen("output.txt", "w", stdout);
#endif
    int i, no_of_requests, initial_head, limit;

    scanf("%d", &no_of_requests);
    struct request req[no_of_requests];
    // { 176, 79, 34, 60, 92, 11, 41, 114 };
    for (i = 0; i < no_of_requests; ++i)
    {
        scanf("%d", &req[i].request_track_number);
        req[i].visited = false;
    }

    scanf("%d", &initial_head);
    SSTFDiskScheduling(req, no_of_requests, initial_head);
    return 0;
}
```


OUTPUT :



The screenshot shows a C++ program in Sublime Text implementing the SSTF (Shortest Seek Time First) disk scheduling algorithm. The program takes an initial head position and a list of requests as input and outputs the seek sequence and total seek time.

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 struct request
4 {
5     int request_track_number;
6     bool visited;
7 };
8 void SSTFDiskScheduling(struct request req[], int no_of_requests, int initial_head)
9 {
10     int i, j, choice, previous_head, seek_time = 0;
11     cout << "\nSeek sequence is \n";
12     printf("%d", initial_head);
13     int n = no_of_requests;
14     while (n)
15     {
16         int min = 1e9;
17         int min_track_number, position;
18         for (i = 0; i < no_of_requests; i++)
19         {
20             if (abs(initial_head - req[i].request_track_number) < min && req[i].visited ==
21                 false)
22             {
23                 min = abs(initial_head - req[i].request_track_number);
24                 min_track_number = req[i].request_track_number;
25                 position = i;
26             }
27             initial_head = req[position].request_track_number;
28             req[position].visited = true;
29             printf("-> %d", min_track_number);
30             seek_time += min;
31             n--;
32         }
33         printf("\nSeek Time: %d\n", seek_time);
34     }
35 }
36 int main()
37 {
38     ios_base::sync_with_stdio(false);
39     cin.tie(NULL);
40     #ifndef ONLINE_JUDGE
41         freopen("input.txt", "r", stdin);
42     #endif
43 }
```

The input file (input.txt) contains the following data:

```
1 8
2 176 79 34 60 92 11 41 114
3 50
```

The output file (output.txt) shows the execution results:

```
1
2 Seek sequence is
3 50-> 41-> 34-> 11-> 60-> 79-> 92-> 114-> 176
4 Seek Time: 204
5
```

The status bar at the bottom indicates the file is 'ECC: [M] Line 38, Column 38', the tab size is 4, and the language is C++.

PROGRAM 19 : C-SCAN disk scheduling algorithm.

CODE :

```
#include <bits/stdc++.h>
using namespace std;
int disk_size = 200;

void CSCAN(int arr[], int head, int size)
{
    int seek_count = 0;
    int distance, cur_track;
    vector<int> left, right;
    vector<int> seek_sequence;
    seek_sequence.push_back(head);
    // appending end values which has to be visited
    // before reversing the direction
    left.push_back(0);
    right.push_back(disk_size - 1);
    // tracks on the left of the head will be serviced when
    // once the head comes back to the beginning (left end).
    for (int i = 0; i < size; i++)
    {
        if (arr[i] < head)
            left.push_back(arr[i]);
        if (arr[i] > head)
            right.push_back(arr[i]);
    }
    // sorting left and right vectors
    sort(left.begin(), left.end());
    sort(right.begin(), right.end());
    // first service the request on the right side of the head.
    for (int i = 0; i < right.size(); i++)
    {
        cur_track = right[i];

        // appending current track to seek sequence
        seek_sequence.push_back(cur_track);
        distance = abs(cur_track - head); // calculate absolute distance
        seek_count += distance; // increase the total count
        head = cur_track; // accessed track is now new head
    }
    // once reached the right end, jump to the beginning.
    head = 0;
    // adding seek count for head returning from 199 to 0
    if (left.size())
        seek_count += (disk_size - 1);
    // Now service the requests again which are left.
    for (int i = 0; i < left.size(); i++)
    {
        cur_track = left[i];

        // appending current track to seek sequence
        seek_sequence.push_back(cur_track);
        distance = abs(cur_track - head); // calculate absolute distance
        seek_count += distance; // increase the total count
        head = cur_track; // accessed track is now the new head
    }
    cout << "Total seek time = " << seek_count << endl;
    cout << "Seek Sequence is" << endl;
    for (int i = 0; i < seek_sequence.size() - 1; i++)
    {
```

```

        cout << seek_sequence[i] << " -> ";
    }
    cout << seek_sequence.back();
}
int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
#ifdef ONLINE_JUDGE
    freopen("input.txt", "r", stdin);
    freopen("error.txt", "w", stderr);
    freopen("output.txt", "w", stdout);
#endif

    int head, n;
    cin >> n;
    int arr[n];
    for (int i = 0; i < n; i++)
        cin >> arr[i];
    cin >> head;
    CSCAN(arr, head, n);
    return 0;
}

```

OUTPUT :

```

C:\Users\ashish\Desktop\c & cpp practice\sublime code\ashish.cpp - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

ashish.cpp x untitled a.c x
1 #include <bits/stdc++.h>
2 using namespace std;
3 int disk_size = 200;
4
5 void CSCAN(int arr[], int head, int size)
6 {
7     int seek_count = 0;
8     int distance, cur_track;
9     vector<int> left, right;
10    vector<int> seek_sequence;
11    seek_sequence.push_back(head);
12    // appending end values which has to be visited
13    // before reversing the direction
14    left.push_back(0);
15    right.push_back(disk_size - 1);
16    // tracks on the left of the head will be serviced when
17    // once the head comes back to the beginning (left end).
18    for (int i = 0; i < size; i++)
19    {
20        if (arr[i] < head)
21            left.push_back(arr[i]);
22        if (arr[i] > head)
23            right.push_back(arr[i]);
24    }
25    // sorting left and right vectors
26    sort(left.begin(), left.end());
27    sort(right.begin(), right.end());
28    // first service the request on the right side of the head.
29    for (int i = 0; i < right.size(); i++)
30    {
31        cur_track = right[i];
32
33        // appending current track to seek sequence
34        seek_sequence.push_back(cur_track);
35        distance = abs(cur_track - head); // calculate absolute distance
36        seek_count += distance; // increase the total count
37        head = cur_track; // accessed track is now new head
38    }
39    // once reached the right end, jump to the beginning.
40    head = 0;
41    // adding seek count for head returning from 199 to 0

```

```

input.txt x
1 8
2 176 79 34 60 92 11 41 114
3 50

```

```

output.txt x
1 Total seek time = 389
2 Seek Sequence is
3 50 -> 60 -> 79 -> 92 -> 114 -> 176 -> 199 -> 0 -> 11
  -> 34 -> 41

```

[Finished in 2.0s]

ECC: [✓], Line 17, Column 56 Tab Size: 4 C++

PROGRAM : 20 LOOK disk scheduling algorithm.

CODE :

```
#include <bits/stdc++.h>
using namespace std;
int disk_size = 200;
void input(int arr[], int n)
{
    for (int i = 0; i < n; i++)
        cin >> arr[i];
}
void LOOK(int arr[], int head, int size, string direction)
{
    int seek_count = 0;
    int distance, cur_track;
    vector<int> left, right;
    vector<int> seek_sequence;
    seek_sequence.push_back(head);
    // appending values which are currently at left
    //and right direction from the head.
    for (int i = 0; i < size; i++)
    {
        if (arr[i] < head)
            left.push_back(arr[i]);
        else
            right.push_back(arr[i]);
    }
    // sorting left and right vectors for
    // servicing tracks in the correct sequence.
    std::sort(left.begin(), left.end());
    std::sort(right.begin(), right.end());
    // run the while loop two times.
    // to look in both direction
    int run = 2;
    while (run--)
    {
        if (direction == "left")
        {
            for (int i = left.size() - 1; i >= 0; i--)
            {
                cur_track = left[i];
                // appending current track to seek sequence
                seek_sequence.push_back(cur_track);
                distance = abs(cur_track - head); // calculate absolute distance
                seek_count += distance; // increase the total count
                head = cur_track; // accessed track is now the new head
            }
            direction = "right"; // reversing the direction
        }
        else if (direction == "right")
        {
            for (int i = 0; i < right.size(); i++)
            {
                cur_track = right[i];
                seek_sequence.push_back(cur_track);
                distance = abs(cur_track - head);
                seek_count += distance;
                head = cur_track;
            }
        }
    }
}
```

```

        direction = "left";// reversing the direction
    }
}
cout << "Total seek time = " << seek_count << endl;
cout << "Seek Sequence is" << endl;
for (int i = 0; i < seek_sequence.size() - 1; i++)
{
    cout << seek_sequence[i] << " -> ";
}
cout << seek_sequence.back();
}
int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
#ifdef ONLINE_JUDGE
    freopen("input.txt", "r", stdin);
    freopen("error.txt", "w", stderr);
    freopen("output.txt", "w", stdout);
#endif
    int head, n;
    cin >> n;
    int arr[n];
    input(arr, n);
// { 176, 79, 34, 60, 92, 11, 41, 114 };
    cin >> head;
    string direction = "right";

    cin >> direction;
    LOOK(arr, head, n, direction);
    return 0;
}

```

OUTPUT :

```

C:\Users\ashish\Desktop\c & cpp practice\sublime code\ashish.cpp - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
ashish.cpp x input.txt x
1 using namespace std;
2 int disk_size = 200;
3 void input(int arr[], int n)
4 {
5     for (int i = 0; i < n; i++)
6         cin >> arr[i];
7 }
8 void LOOK(int arr[], int head, int size, string direction)
9 {
10     int seek_count = 0;
11     int distance, cur_track;
12     vector<int> left, right;
13     vector<int> seek_sequence;
14     seek_sequence.push_back(head);
15     // appending values which are currently at left
16     // and right direction from the head.
17     for (int i = 0; i < size; i++)
18     {
19         if (arr[i] < head)
20             left.push_back(arr[i]);
21         else
22             right.push_back(arr[i]);
23     }
24     // sorting left and right vectors for
25     // servicing tracks in the correct sequence.
26     std::sort(left.begin(), left.end());
27     std::sort(right.begin(), right.end());
28     // run the while loop two times.
29     // to look in both direction
30     int run = 2;
31     while (run--)
32     {
33         if (direction == "left")
34         {
35             for (int i = left.size() - 1; i >= 0; i--)
36             {
37                 cur_track = left[i];
38                 // appending current track to seek sequence
39                 seek_sequence.push_back(cur_track);
40                 distance = abs(cur_track - head); // calculate absolute distance

```

input.txt

```

1 8
2 176 79 34 60 92 11 41 114
3 50
4 right

```

output.txt

```

1 Total seek time = 291
2 Seek Sequence is
3 50 -> 60 -> 79 -> 92 -> 114 -> 176 -> 41 -> 34 -> 11

```

[Finished in 2.1s]

ECC: [✓] Line 36, Column 55 Tab Size: 4 C++