LAB FILE

JAMIA MILLIA ISLAMIA

DEPARTMENT OF COMPUTER ENGINEERING

B.TECH (COMPUTER ENGINEERING)

4th SEMESTER

Operating Systems Lab (CEN-493)

Submitted To:
Mr. Jawahar Lal
Dr. Shahzad Alam
Dr. Waseem Ahmad

Submitted By:
Md Ibrahim Akhtar
21BCS007
B.Tech (4th Semester)

Index

Program 1: Write a program to implement **Priority Queue** scheduling algorithm and find the average turnaround time and average waiting time. Also print the Gantt chart.

```
#include <iostream>
using namespace std;
struct Process
{
    int pid;
    int priority;
    int burst_time;
    int completion;
};
struct Pnode
{
    Process process;
    Pnode *next;
};
Process processIn()
{
    Process process;
    int val;
    std::cout << "\nEnter process id: ";
    std::cin >> val;
    process.pid = val;
    std::cout << "Enter priority: ";
    std::cin >> val;
    process.priority = val;
    std::cout << "Enter burst time: ";
    std::cin >> val;
    process.burst_time = val;
    return process;
}
int main()
{
    std::cout << "\nProgram by Md Ibrahim Akhtar 21BCS007\n";
    int n;
    std::cout << "\nEnter number of processes: ";
    std::cin >> n;
    Pnode *processes = nullptr;
    Pnode *curr = nullptr;
    for (int i = 0; i < n; ++i)
    {
        if (curr == nullptr)
        {
            processes = new Pnode;
            processes->process = processIn();
```

```
            processes->next = nullptr;
            curr = processes;
        }
        else
        {
            curr->next = new Pnode;
            curr = curr->next;
            curr->process = processIn();
            curr->next = nullptr;
        }
    }
    Pnode *start = processes;
    Pnode *prevstart = nullptr;
    while (start->next != nullptr)
    {
        curr = start;
        Pnode *prev = nullptr;
        Pnode *minp = curr;
        Pnode *minprev = nullptr;
        while (curr != nullptr)
        {
            if (curr->process.priority < minp->process.priority)
            {
                minp = curr;
                minprev = prev;
            }
            prev = curr;
            curr = curr->next;
        }
        if (prevstart == nullptr)
        {
            processes = minp;
        }
        else
        {
            prevstart->next = minp;
        }
        if (minprev != nullptr)
        {
            minprev->next = minp->next;
            minp->next = start;
            prevstart = minp;
        }
        else
        {
```

```
            start = start->next;
            prevstart = start;
        }
    }
    curr = processes;
    std::cout << std::endl
              << " ";
    while (curr != nullptr)
    {
        std::cout << "P" << curr->process.pid << "\t";
        curr = curr->next;
    }
    std::cout << std::endl;
    int time = 0;
    while (processes != nullptr)
    {
        std::cout << time << "\t";
        time += processes->process.burst_time;
        auto temp = processes->next;
        delete (processes);
        processes = temp;
    }
    cout << "\t" << time << endl;
}
```

```
/tmp/RquHeAlVNJ.o
Program by Md Ibrahim Akhtar 21BCS007

Enter number of processes: 4
Enter process id: 1
Enter priority: 3
Enter burst time: 3
Enter process id: 2
Enter priority: 4
Enter burst time: 2
Enter process id: 3
Enter priority: 2
Enter burst time: 1
Enter process id: 4
Enter priority: 1
Enter burst time: 2
P4  P3  P1  P2
0   2   3   6       8
```

Program 2: Write a program to implement **Shortest Job First scheduling algorithm** and find the average turnaround time and average waiting time. Also print the Gantt chart.

```
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
struct Job
{
    int id;
    int arrival;
    int burst;
    int completion;
};
struct JobNode
{
    Job job;
    JobNode *next;
};
void insertJob(JobNode **jobs, Job job)
{
    if (*jobs == nullptr)
    {
        *jobs = new JobNode();
        (*jobs)->job = job;
        (*jobs)->next = nullptr;
        return;
    }
    JobNode *prev = nullptr;
    JobNode *curr = *jobs;
    while (curr != nullptr && curr->job.burst < job.burst)
    {
        prev = curr;
        curr = curr->next;
    }
    if (prev == nullptr)
    {
        auto temp = new JobNode();
        temp->job = job;
        temp->next = *jobs;
        *jobs = temp;
    }
    else
    {
        prev->next = new JobNode();
        prev->next->job = job;
```

```
        prev->next->next = curr;
    }
}
void removeJob(JobNode **jobs, Job job)
{
    JobNode *prev = nullptr;
    JobNode *curr = *jobs;
    while (curr != nullptr && curr->job.id != job.id)
    {
        prev = curr;
        curr = curr->next;
    }
    if (prev == nullptr)
    {
        auto temp = (*jobs);
        *jobs = temp->next;
        delete (temp);
    }
    else
    {
        if (curr == nullptr)
        {
            std::cout << "No job with job id: " << job.id << " found!" <<
std::endl;
            return;
        }
        prev->next = curr->next;
        delete (curr);
    }
}
Job chooseJob(JobNode *jobs, int time)
{
    while (jobs != nullptr)
    {
        if (jobs->job.arrival <= time)
        {
            return jobs->job;
        }
        jobs = jobs->next;
    }
    Job notFound;
    notFound.id = -1;
    return notFound;
}
void printJobs(JobNode *jobs)
```

```
{
    while (jobs != nullptr)
    {
        std::cout << "P" << jobs->job.id << std::endl;
        jobs = jobs->next;
    }
}
Job inputJob()
{
    Job job;
    std::cout << "Enter job id: ";
    std::cin >> job.id;
    std::cout << "Enter burst time: ";
    std::cin >> job.burst;
    std::cout << "Enter arrival time: ";
    std::cin >> job.arrival;
    return job;
}
static bool comp(Job &one, Job &two)
{
    return one.id < two.id;
}
int main()
{
    std::cout << "\nProgram By Md Ibrahim Akhtar 21BCS007\n";
    JobNode *jobs = nullptr;
    int n;
    std::cout << "Enter number of jobs: ";
    std::cin >> n;
    for (int i = 0; i < n; ++i)
    {
        insertJob(&jobs, inputJob());
    }
    int time = 0;
    std::string timeStr = "";
    std::string process = "";
    std::vector<Job> completed;
    while (jobs != nullptr)
    {
        Job job = chooseJob(jobs, time);
        if (job.id != -1)
        {
            timeStr += std::to_string(time) + " ";
            process += " P" + std::to_string(job.id) + " ";
            time += job.burst;
```

```
            removeJob(&jobs, job);
            job.completion = time;
            completed.push_back(job);
        }
        else
        {
            if (time == 0)
                timeStr += "0";
            time += 1;
        }
    }
    timeStr += std::to_string(time);
    std::cout << process << std::endl
              << timeStr << std::endl
              << std::endl;
    std::sort(completed.begin(), completed.end(), comp);
    std::cout << "Process\tBurst\tArrival\tWaiting\tCompletion\n";
    for (auto proc : completed)
    {
        std::cout << proc.id << "\t" << proc.burst << "\t" << proc.arrival << "\t"
                  << proc.completion - proc.arrival - proc.burst << "\t" <<
proc.completion << std::endl;
    }
    return 0;
}
```

```
/tmp/RquHeAlVNJ.o
Program By Md Ibrahim Akhtar 21BCS007
Enter number of jobs: 4
Enter job id: 1
Enter burst time: 2
Enter arrival time: 1
Enter job id: 2
Enter burst time: 3
Enter arrival time: 2
Enter job id: 3
Enter burst time: 4
Enter arrival time: 3
Enter job id: 4
Enter burst time: 1
Enter arrival time: 6
P1  P2  P4  P3
01 3 6 7 11

Process Burst   Arrival Waiting Completion
1   2   1   0   3
2   3   2   1   6
3   4   3   4   11
4   1   6   0   7
```

Program 3: Write a program to implement **Shortest Remaining Time First scheduling algorithm** and find the average turnaround time and average waiting time. Also print the Gantt chart.

```
#include <algorithm>
#include <iostream>
#include <string>
#include <vector>
struct Job
{
    int id;
    int arrival;
    int burst;
    int completion;
    int response = -1;
    int exec = 0;
};
Job inputJob()
{
    Job job;
    std::cout << "Enter job id: ";
    std::cin >> job.id;
    std::cout << "Enter burst time: ";
    std::cin >> job.burst;
    std::cout << "Enter arrival time: ";
    std::cin >> job.arrival;
    return job;
}
void printJobs(Job *jobs, int n)
{
    for (int i = 0; i < n; ++i)
        std::cout << "P" << jobs[i].id << "\t";
}
static bool comp(Job &one, Job &two)
{
    if (one.burst - one.exec == two.burst - two.exec)
        return one.id > two.id;
    else
        return one.burst - one.exec > two.burst - two.exec;
}
void sort(Job *jobs, int n)
{
    for (int i = 0; i < n; ++i)
        for (int j = i; j < n; ++j)
            if (comp(jobs[i], jobs[j]))
            {
                auto temp = jobs[i];
```

```
                jobs[i] = jobs[j];
                jobs[j] = temp;
            }
}
Job *chooseJob(Job *jobs, int n, int time)
{
    for (int i = 0; i < n; ++i)
    {
        if (jobs[i].arrival <= time)
        {
            return jobs + i;
        }
    }
    return nullptr;
}
void removeJob(std::vector<Job> &jobs, Job *job_ptr)
{
    int index = job_ptr - &jobs[0];
    jobs.erase(std::next(jobs.begin(), index));
}
int main()
{
    std::cout << "\nProgram By Md Ibrahim Akhtar 21BCS007\n";
    int n;
    std::cout << "Enter number of jobs: ";
    std::cin >> n;
    std::vector<Job> jobs(n);
    for (int i = 0; i < n; ++i)
        jobs[i] = inputJob();
    sort(jobs.data(), jobs.size());
    int time = 0;
    std::string timeStr = "";
    std::string process = "";
    std::vector<Job> completed;
    int lastid = -1;
    while (jobs.size() != 0)
    {
        Job *job_ptr = chooseJob(jobs.data(), n, time);
        if (job_ptr != nullptr)
        {
            if (job_ptr->response == -1)
                job_ptr->response = time;
            if (job_ptr->id != lastid)
            {
                timeStr += std::to_string(time) + " ";
```

```
                process += " P" + std::to_string(job_ptr->id) + " ";
                lastid = job_ptr->id;
            }
            time += 1;
            job_ptr->exec += 1;
            if (job_ptr->burst <= job_ptr->exec)
            {
                n -= 1;
                job_ptr->completion = time;
                completed.push_back(*job_ptr);
                removeJob(jobs, job_ptr);
            }
            sort(jobs.data(), n);
        }
        else
        {
            if (time == 0)
                timeStr += "0";
            time += 1;
        }
    }
    timeStr += std::to_string(time);
    std::cout << process << std::endl
              << timeStr << std::endl
              << std::endl;
    std::sort(completed.begin(), completed.end(), [](Job &job1, Job &job2)
              { return job1.id < job2.id; });
    std::cout << "Process\tBurst\tArrival\tWaiting\tRes\tTAT\tCompletion\n";
    for (auto proc : completed)
    {
        std::cout << proc.id << "\t" << proc.burst << "\t" << proc.arrival <<
"\t"
                  << proc.completion - proc.arrival - proc.burst << "\t"
                  << proc.response << "\t" << proc.completion - proc.arrival
                  << "\t" << proc.completion << std::endl;
    }
    return 0;
}
```

```
/tmp/RquHeAlVNJ.o
Program By Md Ibrahim Akhtar 21BCS007
Enter number of jobs: 4
Enter job id: 1
Enter burst time: 4
Enter arrival time: 0
Enter job id: 2
Enter burst time: 3
Enter arrival time: 2
Enter job id: 3
Enter burst time: 7
Enter arrival time: 6
Enter job id: 4
Enter burst time: 2
Enter arrival time: 10
P1  P2  P3  P4  P3
0 4 7 10 12 16

Process Burst   Arrival Waiting Res TAT Completion
1   4   0   0   0   4   4
2   3   2   2   4   5   7
3   7   6   3   7   10  16
4   2   10  0   10  2   12
```

Program 4: Round Robin Write a program to implement **Round Robin scheduling algorithm** and find the average turnaround time and average waiting time. Also print the Gantt chart.

```
#include <algorithm>
#include <iostream>
#include <string>
#include <vector>
struct Job
{
    int id;
    int arrival;
    int burst;
    int completion;
    int response = -1;
    int exec = 0;
};
Job inputJob()
{
    Job job;
    std::cout << "Enter job id: ";
    std::cin >> job.id;
    std::cout << "Enter burst time: ";
    std::cin >> job.burst;
    std::cout << "Enter arrival time: ";
    std::cin >> job.arrival;
    return job;
}
void printJobs(Job *jobs, int n)
{
    for (int i = 0; i < n; ++i)
        std::cout << "P" << jobs[i].id << "\t";
}
void removeJob(std::vector<Job> &jobs, Job *job_ptr)
{
    int index = job_ptr - &jobs[0];
    jobs.erase(std::next(jobs.begin(), index));
}
int main()
{
    std::cout << "\nProgram By Md Ibrahim Akhtar 21BCS007\n";
    int tq;
    std::cout << "Enter time quantum: ";
    std::cin >> tq;
    int n;
    std::cout << "Enter number of jobs: ";
    std::cin >> n;
```

```
std::vector<Job> jobs(n);
for (int i = 0; i < n; ++i)
    jobs[i] = inputJob();
int time = 0;
std::string timeStr = "";
std::string process = " ";
std::vector<Job> completed;
int lastid = -1;
int i = 0;
while (jobs.size() != 0)
{
    Job *job_ptr = &jobs[i % n];
    if (job_ptr != nullptr)
    {
        if (job_ptr->response == -1)
            job_ptr->response = time;
        if (job_ptr->id != lastid)
        {
            timeStr += std::to_string(time) + " ";
            process += " P" + std::to_string(job_ptr->id) + " ";
            lastid = job_ptr->id;
        }
        if (job_ptr->burst - job_ptr->exec < tq)
        {
            time += job_ptr->burst - job_ptr->exec;
        }
        else
        {
            time += tq;
        }
        job_ptr->exec += tq;
        if (job_ptr->burst <= job_ptr->exec)
        {
            n -= 1;
            i -= 1;
            job_ptr->completion = time;
            completed.push_back(*job_ptr);
            removeJob(jobs, job_ptr);
        }
    }
    else
    {
        if (time == 0)
            timeStr += "0";
        time += 1;
```

```
        }
        i++;
    }
    timeStr += std::to_string(time);
    std::cout << process << std::endl
              << timeStr << std::endl
              << std::endl;
    std::sort(completed.begin(), completed.end(), [](Job &job1, Job &job2)
              { return job1.id < job2.id; });
    std::cout << "Process\tBurst\tArrival\tWaiting\tRes\tTAT\tCompletion\n";
    for (auto proc : completed)
    {
        std::cout << proc.id << "\t" << proc.burst << "\t" << proc.arrival <<
"\t"
                  << proc.completion - proc.arrival - proc.burst << "\t"
                  << proc.response << "\t" << proc.completion - proc.arrival
                  << "\t" << proc.completion << std::endl;
    }
    return 0;
}
```

```
/tmp/RquHeAlVNJ.o
Program By Md Ibrahim Akhtar 21BCS007
Enter time quantum: 4
Enter number of jobs: 4
Enter job id: 1
Enter burst time: 3
Enter arrival time: 0
Enter job id: 2
Enter burst time: 6
Enter arrival time: 3
Enter job id: 3
Enter burst time: 5
Enter arrival time: 7
Enter job id: 4
Enter burst time: 4
Enter arrival time: 11
P1  P2  P3  P4  P2  P3
0 3 7 11 15 17 18

Process Burst   Arrival Waiting Res TAT Completion
1   3   0   0   0   3   3
2   6   3   8   3   14  17
3   5   7   6   7   11  18
4   4   11  0   11  4   15
```

Program 5: Write a program to implement First fit, Best fit and Worst fit **memory allocation algorithms**.

```
#include <iostream>
#include <vector>
#include <limits>
void printStats(const std::vector<int> &blocks, const std::vector<int>
&processes, const std::vector<int> &allocations, const std::vector<bool> &taken)
{
    int internalFrag = 0;
    for (int i = 0; i < allocations.size(); ++i)
    {
        if (allocations[i] != -1)
        {
            std::cout << "Process " << i << " of size " << processes[i] << " is
allocated to block " << allocations[i] << " of size " << blocks[allocations[i]]
<< std::endl;
            internalFrag += blocks[allocations[i]] - processes[i];
        }
        else
            std::cout << "Process " << i << " of size " << processes[i] << " was
not allocated" << std::endl;
    }
    std::cout << "Internal Fragmentation = " << internalFrag << std::endl;
    int externalFrag = 0;
    for (int j = 0; j < blocks.size(); ++j)
    {
        externalFrag += taken[j] ? 0 : blocks[j];
    }
    std::cout << "External Fragmentation = " << externalFrag;
}
void bestFit(const std::vector<int> &blocks, const std::vector<int> &
                                                    processes)
{
    // return index of block allocated for each process, if not allocated index
is -1
    std::vector<int> allocations;
    std::vector<bool> taken(blocks.size(), false);
    for (int i = 0; i < processes.size(); ++i)
    {
        int minSizeDiff = std::numeric_limits<int>::max();
        int minBlock = -1;
        for (int j = 0; j < blocks.size(); ++j)
        {
            int diff = blocks[j] - processes[i];
```

```
            if (diff >= 0 && taken[j] == false)
            {
                if (diff < minSizeDiff)
                {
                    minBlock = j;
                    minSizeDiff = diff;
                }
            }
        }
        if (minBlock != -1)
            taken[minBlock] = true;
        allocations.push_back(minBlock);
    }
    printStats(blocks, processes, allocations, taken);
}
void worstFit(const std::vector<int> &blocks, const std::vector<int> &
                                                    processes)
{
    // return index of block allocated for each process, if not allocated index
is -1
    std::vector<int> allocations;
    std::vector<bool> taken(blocks.size(), false);
    for (int i = 0; i < processes.size(); ++i)
    {
        int maxSizeDiff = std::numeric_limits<int>::min();
        int maxBlock = -1;
        for (int j = 0; j < blocks.size(); ++j)
        {
            int diff = blocks[j] - processes[i];
            if (diff >= 0 && taken[j] == false)
            {
                if (diff > maxSizeDiff)
                {
                    maxBlock = j;
                    maxSizeDiff = diff;
                }
            }
        }
        if (maxBlock != -1)
            taken[maxBlock] = true;
        allocations.push_back(maxBlock);
    }
    printStats(blocks, processes, allocations, taken);
}
void firstFit(const std::vector<int> &blocks, const std::vector<int> &processes)
```

```
{
    // return index of block allocated for each process, if not allocated index
is -1
    std::vector<int> allocations;
    std::vector<bool> taken(blocks.size(), false);
    for (int i = 0; i < processes.size(); ++i)
    {
        int blockIndex = -1;
        for (int j = 0; j < blocks.size(); ++j)
        {
            int diff = blocks[j] - processes[i];
            if (diff >= 0 && taken[j] == false)
            {
                blockIndex = j;
                break;
            }
        }
        if (blockIndex != -1)
            taken[blockIndex] = true;
        allocations.push_back(blockIndex);
    }
    printStats(blocks, processes, allocations, taken);
}
void nextFit(const std::vector<int> &blocks, const std::vector<int> &
                                                    processes)
{
    // return index of block allocated for each process, if not allocated index
is -1
    std::vector<int> allocations;
    std::vector<bool> taken(blocks.size(), false);
    int blockStart = 0;
    for (int i = 0; i < processes.size(); ++i)
    {
        int blockIndex = -1;
        for (int j = blockStart; j < blocks.size(); ++j)
        {
            int diff = blocks[j] - processes[i];
            if (diff >= 0 && taken[j] == false)
            {
                blockIndex = j;
                break;
            }
        }
        if (blockIndex != -1)
        {
```

```
            taken[blockIndex] = true;
            blockStart = blockIndex;
        }
        allocations.push_back(blockIndex);
    }
    printStats(blocks, processes, allocations, taken);
}
int main()
{
    std::cout << "\nProgram By Md Ibrahim Akhtar 21BCS007\n";
    int num_blocks, num_process;
    std::cout << "Enter the number of blocks: ";
    std::cin >> num_blocks;
    std::cout << "Enter the number of processes: ";
    std::cin >> num_process;
    std::vector<int> blocks(num_blocks);
    std::vector<int> process(num_process);
    int i = 0;
    for (auto &block : blocks)
    {
        std::cout << "Enter size of block " << i << " in kilobytes: ";
        std::cin >> block;
        i++;
    }
    i = 0;
    for (auto &pro : process)
    {
        std::cout << "Enter size of process " << i << " in kilobytes: ";
        std::cin >> pro;
        i++;
    }
    std::cout << "BLOCKS: ";
    for (auto block : blocks)
        std::cout << block << " ";
    std::cout << std::endl;
    std::cout << "PROCESSES: ";
    for (auto pro : process)
        std::cout << pro << " ";
    std::cout << std::endl;
    while (1)
    {
        std::cout << "Enter:\n\t1 for best fit\n\t2 for worst fit\n\t3 for first
fit\n\t4 for next fit\n\t5 to exit\n";
        int opt = 0;
        std::cin >> opt;
```

```
        switch (opt)
        {
        case 1:
            std::cout << "BEST FIT: \n";
            bestFit(blocks, process);
            std::cout << std::endl;
            break;
        case 2:
            std::cout << "WORST FIT: \n";
            worstFit(blocks, process);
            std::cout << std::endl;
            break;
        case 3:
            std::cout << "FIRST FIT: \n";
            firstFit(blocks, process);
            std::cout << std::endl;
            break;
        case 4:
            std::cout << "NEXT FIT: \n";
            nextFit(blocks, process);
            std::cout << std::endl;
            break;
        case 5:
            std::cout << "\nexiting...\n";
            return 0;
        default:
            std::cout << "Enter valid option!\n";
            break;
        }
    }
    return 0;
}
```

```
/tmp/4Z3cI2d8Zy.o
Program By Md Ibrahim Akhtar 21BCS007
Enter the number of blocks: 6
Enter the number of processes: 4
Enter size of block 0 in kilobytes: 128
Enter size of block 1 in kilobytes: 256
Enter size of block 2 in kilobytes: 64
Enter size of block 3 in kilobytes: 200
Enter size of block 4 in kilobytes: 50
Enter size of block 5 in kilobytes: 100
Enter size of process 0 in kilobytes: 120
Enter size of process 1 in kilobytes: 240
Enter size of process 2 in kilobytes: 215
Enter size of process 3 in kilobytes: 75
BLOCKS: 128 256 64 200 50 100
PROCESSES: 120 240 215 75
Enter:
    1 for best fit
    2 for worst fit
    3 for first fit
    4 for next fit
    5 to exit
```

```
1
BEST FIT:
Process 0 of size 120 is allocated to block 0 of size 128
Process 1 of size 240 is allocated to block 1 of size 256
Process 2 of size 215 was not allocated
Process 3 of size 75 is allocated to block 5 of size 100
Internal Fragmentation = 49
External Fragmentation = 314
Enter:
    1 for best fit
    2 for worst fit
    3 for first fit
    4 for next fit
    5 to exit
2
WORST FIT:
Process 0 of size 120 is allocated to block 1 of size 256
Process 1 of size 240 was not allocated
Process 2 of size 215 was not allocated
Process 3 of size 75 is allocated to block 3 of size 200
Internal Fragmentation = 261
External Fragmentation = 342
```

```
3
FIRST FIT:
Process 0 of size 120 is allocated to block 0 of size 128
Process 1 of size 240 is allocated to block 1 of size 256
Process 2 of size 215 was not allocated
Process 3 of size 75 is allocated to block 3 of size 200
Internal Fragmentation = 149
External Fragmentation = 214
Enter:
    1 for best fit
    2 for worst fit
    3 for first fit
    4 for next fit
    5 to exit
4
NEXT FIT:
Process 0 of size 120 is allocated to block 0 of size 128
Process 1 of size 240 is allocated to block 1 of size 256
Process 2 of size 215 was not allocated
Process 3 of size 75 is allocated to block 3 of size 200
Internal Fragmentation = 149
External Fragmentation = 214
```

Program 6: Write a program to implement following **disk scheduling algorithms**:
a) FCFS, b) SSTF, c) SCAN, d) CSCAN, e) LOOK, f) CLOOK

```
#include <vector>
#include <limits>
#include <iostream>
using namespace std;
void mysort(vector<int> &vec, bool isAscending)
{
    for (int i = 0; i < vec.size(); ++i)
    {
        for (int j = i + 1; j < vec.size(); ++j)
        {
            bool shouldExchange = false;
            if (isAscending && vec[i] > vec[j])
                shouldExchange = true;
            else if (!isAscending && vec[i] < vec[j])
                shouldExchange = true;
            if (shouldExchange)
            {
                int temp = vec[i];
                vec[i] = vec[j];
                vec[j] = temp;
            }
        }
    }
}
int myabs(int val)
{
    int isPos = (int)(val >= 0);
    return (isPos * val + (1 - isPos) * (-val));
}
int FCFS(vector<int> tracks, int pos)
{
    int movement = 0;
    cout << pos << " >> ";
    for (auto &track : tracks)
    {
        movement += myabs(track - pos);
        pos = track;
        cout << track << " >> ";
    }
    cout << " |\n";
    return movement;
}
int SSTF(vector<int> tracks, int pos)
```

```
{
    int movement = 0;
    int len = tracks.size();
    cout << pos << " >> ";
    for (int __i = 0; __i < len; ++__i)
    {
        int chosen = -1;
        int minDist = std::numeric_limits<int>::max();
        for (int j = 0; j < tracks.size(); ++j)
        {
            auto &track = tracks[j];
            if (myabs(track - pos) < minDist)
            {
                minDist = myabs(track - pos);
                chosen = j;
            }
        }
        movement += myabs(tracks[chosen] - pos);
        pos = tracks[chosen];
        cout << tracks[chosen] << " >> ";
        tracks.erase(tracks.begin() + chosen);
    }
    cout << " |\n";
    return movement;
}
int SCAN(vector<int> tracks, int pos, int maxl, int minl, bool isRight)
{
    int movement = 0;
    mysort(tracks, true);
    cout << pos << " >> ";
    int start = -1;
    for (int i = 0; i < tracks.size(); ++i)
    {
        if (tracks[i] > pos)
        {
            start = i;
            break;
        }
    }
    if (isRight)
    {
        if (start == 0)
        {
            movement += myabs(tracks.back() - pos);
        }
```

```
        else
        {
            movement += maxl - pos;
            movement += maxl - tracks[0];
        }
        for (int i = start; i < tracks.size(); ++i)
        {
            cout << tracks[i] << " >> ";
        }
        cout << maxl << " >> ";
        for (int i = start - 1; i >= 0; i--)
        {
            cout << tracks[i] << " >> ";
        }
        cout << " |\n";
        return movement;
    }
    else
    {
        if (start == tracks.size() - 1)
        {
            movement += myabs(pos - tracks[0]);
        }
        else
        {
            movement += pos - minl;
            movement += tracks.back() - minl;
        }
        return movement;
    }
}
int CSCAN(vector<int> tracks, int pos, int maxl, int minl, bool isRight)
{
    int movement = 0;
    mysort(tracks, true);
    cout << pos << " >> ";
    int start = -1;
    for (int i = 0; i < tracks.size(); ++i)
    {
        if (tracks[i] > pos)
        {
            start = i;
            break;
        }
    }
```

```
    if (isRight)
    {
        if (start == 0)
        {
            movement += myabs(tracks.back() - pos);
        }
        else
        {
            movement += maxl - pos;
            movement += maxl - minl;
            movement += tracks[start - 1] - minl;
        }
        for (int i = start; i < tracks.size(); ++i)
        {
            cout << tracks[i] << " >> ";
        }
        cout << maxl << " >> " << minl << " >> ";
        for (int i = 0; i < start; i++)
        {
            cout << tracks[i] << " >> ";
        }
        cout << " |\n";
        return movement;
    }
    else
    {
        if (start == tracks.size() - 1)
        {
            movement += myabs(pos - tracks[0]);
        }
        else
        {
            movement += pos - minl;
            movement += maxl - minl;
            movement += maxl - tracks[start];
        }
        return movement;
    }
}
int LOOK(vector<int> tracks, int pos, int maxl, int minl, bool isRight)
{
    int movement = 0;
    mysort(tracks, true);
    int start = -1;
    for (int i = 0; i < tracks.size(); ++i)
```

```
    {
        if (tracks[i] > pos)
        {
            start = i;
            break;
        }
    }
    cout << pos << " >> ";
    if (isRight)
    {
        if (start == 0)
        {
            movement += myabs(tracks.back() - pos);
        }
        else
        {
            movement += tracks.back() - pos;
            movement += tracks.back() - tracks[0];
        }
        for (int i = start; i < tracks.size(); ++i)
        {
            cout << tracks[i] << " >> ";
        }
        for (int i = start - 1; i >= 0; i--)
        {
            cout << tracks[i] << " >> ";
        }
        cout << " |\n";
        return movement;
    }
    else
    {
        if (start == tracks.size() - 1)
        {
            movement += myabs(pos - tracks[0]);
        }
        else
        {
            movement += pos - tracks[0];
            movement += tracks.back() - tracks[0];
        }
        return movement;
    }
}
int CLOOK(vector<int> tracks, int pos, int maxl, int minl, bool isRight)
```

```
{
    int movement = 0;
    mysort(tracks, true);
    int start = -1;
    for (int i = 0; i < tracks.size(); ++i)
    {
        if (tracks[i] > pos)
        {
            start = i;
            break;
        }
    }
    cout << pos << " >> ";
    if (isRight)
    {
        if (start == 0)
        {
            movement += myabs(tracks.back() - pos);
        }
        else
        {
            movement += tracks.back() - pos;
            movement += tracks.back() - tracks[0];
            movement += tracks[start - 1] - tracks[0];
        }
        for (int i = start; i < tracks.size(); ++i)
        {
            cout << tracks[i] << " >> ";
        }
        for (int i = 0; i < start; i++)
        {
            cout << tracks[i] << " >> ";
        }
        cout << " |\n";
        return movement;
    }
    else
    {
        if (start == tracks.size() - 1)
        {
            movement += myabs(pos - tracks[0]);
        }
        else
        {
            movement += pos - tracks[0];
```

```
            movement += tracks.back() - tracks[0];
            movement += tracks.back() - tracks[start];
        }
        return movement;
    }
}
int main()
{
    std::cout << "\nProgram By Md Ibrahim Akhtar 21BCS007\n";
    int initial;
    cout << "Enter initial position: ";
    cin >> initial;
    int inputLength;
    cout << "Enter length of array: ";
    cin >> inputLength;
    vector<int> inputs(inputLength);
    cout << "Enter element of array: ";
    for (int &input : inputs)
        cin >> input;
    cout << "Input array is: \n\t";
    for (int input : inputs)
        cout << input << "\t";
    cout << "\n";
    int choice;
    while (1)
    {
        cout << "\nEnter:\n\t";
        cout << "1 to perform FCFS Scheduling:\n\t";
        cout << "2 to perform SSTF Scheduling:\n\t";
        cout << "3 to perform SCAN Scheduling:\n\t";
        cout << "4 to perform CSCAN Scheduling:\n\t";
        cout << "5 to perform LOOK Scheduling:\n\t";
        cout << "6 to perform CLOOK Scheduling:\n\t";
        cout << "7 to exit:\n";
        cin >> choice;
        int totalMovements = -1;
        switch (choice)
        {
        case 1:
            totalMovements = FCFS(inputs, initial);
            break;
        case 2:
            totalMovements = SSTF(inputs, initial);
            break;
        case 3:
```

```
            totalMovements = SCAN(inputs, initial, 199, 0, true);
            break;
        case 4:
            totalMovements = CSCAN(inputs, initial, 199, 0, true);
            break;
        case 5:
            totalMovements = LOOK(inputs, initial, 199, 0, true);
            break;
        case 6:
            totalMovements = CLOOK(inputs, initial, 199, 0, true);
            break;
        case 7:
            cout << "Exiting...\n";
            return 0;
        default:
            cout << "Enter valid choice!\n";
            break;
        }
        if (totalMovements >= 0)
        {
            cout << "\n\tTotal Movements = " << totalMovements;
        }
    }
    return 0;
}
```

```
/tmp/4Z3cI2d8Zy.o
Program By Md Ibrahim Akhtar 21BCS007
Enter initial position: 50
Enter length of array: 7
Enter element of array: 82 170 43 140 24 16 190
Input array is:
    82  170 43  140 24  16  190

Enter:
    1 to perform FCFS Scheduling:
    2 to perform SSTF Scheduling:
    3 to perform SCAN Scheduling:
    4 to perform CSCAN Scheduling:
    5 to perform LOOK Scheduling:
    6 to perform CLOOK Scheduling:
    7 to exit:
1
50 >> 82 >> 170 >> 43 >> 140 >> 24 >> 16 >> 190 >>  |
Total Movements = 642
Enter:
    1 to perform FCFS Scheduling:
    2 to perform SSTF Scheduling:
    3 to perform SCAN Scheduling:
    4 to perform CSCAN Scheduling:
    5 to perform LOOK Scheduling:
    6 to perform CLOOK Scheduling:
    7 to exit:
```

```
2
50 >> 43 >> 24 >> 16 >> 82 >> 140 >> 170 >> 190 >>  |

    Total Movements = 208
Enter:
    1 to perform FCFS Scheduling:
    2 to perform SSTF Scheduling:
    3 to perform SCAN Scheduling:
    4 to perform CSCAN Scheduling:
    5 to perform LOOK Scheduling:
    6 to perform CLOOK Scheduling:
    7 to exit:
3
50 >> 82 >> 140 >> 170 >> 190 >> 199 >> 43 >> 24 >> 16 >>  |

    Total Movements = 332
Enter:
    1 to perform FCFS Scheduling:
    2 to perform SSTF Scheduling:
    3 to perform SCAN Scheduling:
    4 to perform CSCAN Scheduling:
    5 to perform LOOK Scheduling:
    6 to perform CLOOK Scheduling:
    7 to exit:
```

```
4
50 >> 82 >> 140 >> 170 >> 190 >> 199 >> 0 >> 16 >> 24 >> 43 >>  |

    Total Movements = 391
Enter:
    1 to perform FCFS Scheduling:
    2 to perform SSTF Scheduling:
    3 to perform SCAN Scheduling:
    4 to perform CSCAN Scheduling:
    5 to perform LOOK Scheduling:
    6 to perform CLOOK Scheduling:
    7 to exit:
5
50 >> 82 >> 140 >> 170 >> 190 >> 43 >> 24 >> 16 >>  |

    Total Movements = 314
Enter:
    1 to perform FCFS Scheduling:
    2 to perform SSTF Scheduling:
    3 to perform SCAN Scheduling:
    4 to perform CSCAN Scheduling:
    5 to perform LOOK Scheduling:
    6 to perform CLOOK Scheduling:
    7 to exit:
```

```
6
50 >> 82 >> 140 >> 170 >> 190 >> 16 >> 24 >> 43 >>  |

    Total Movements = 341
```