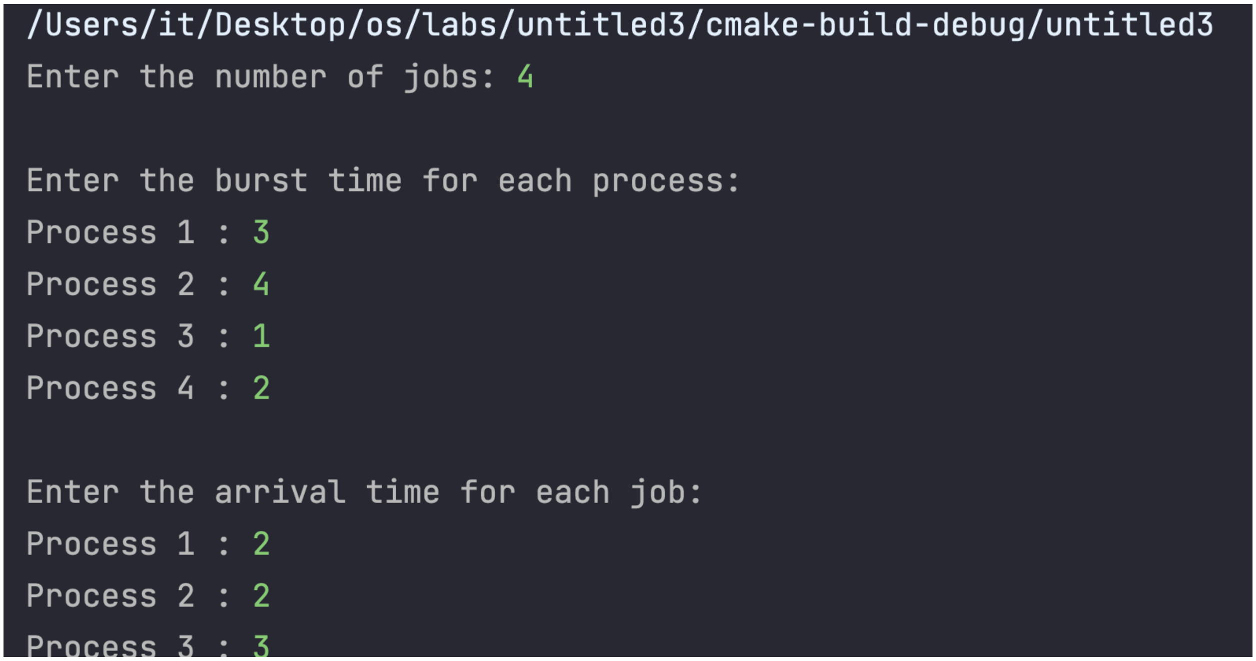
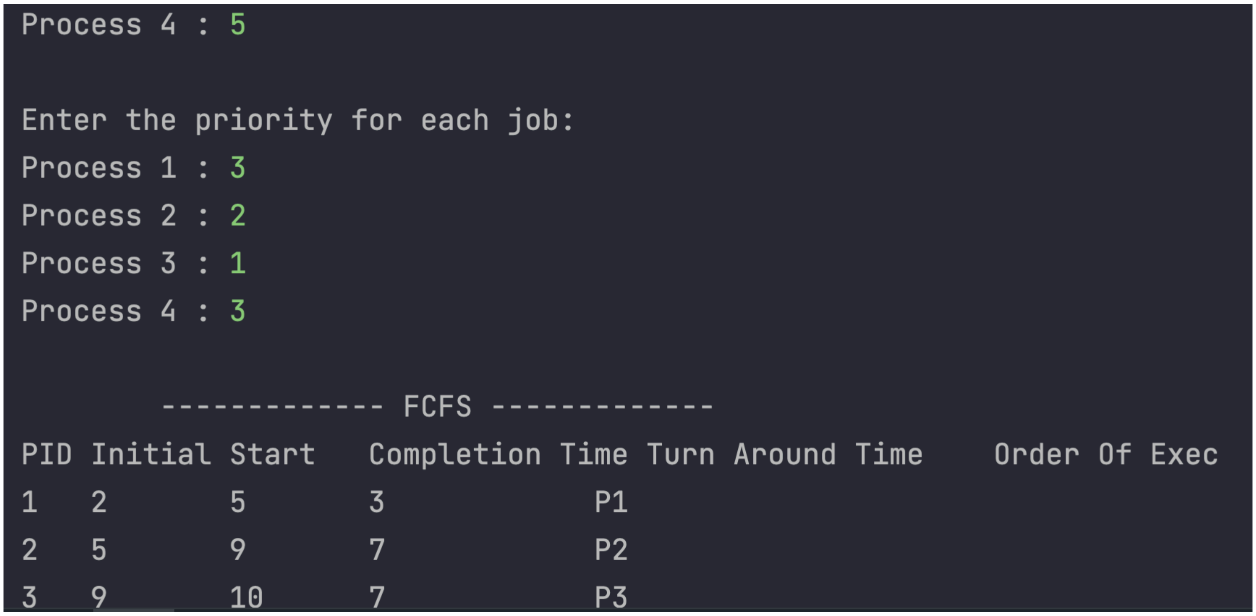
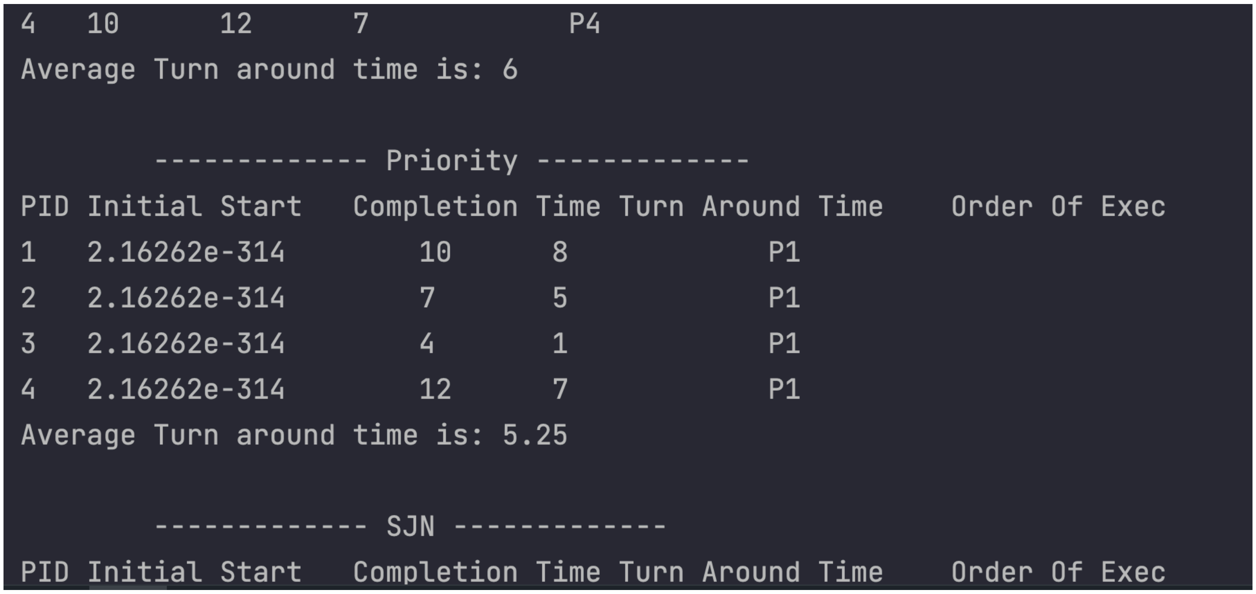


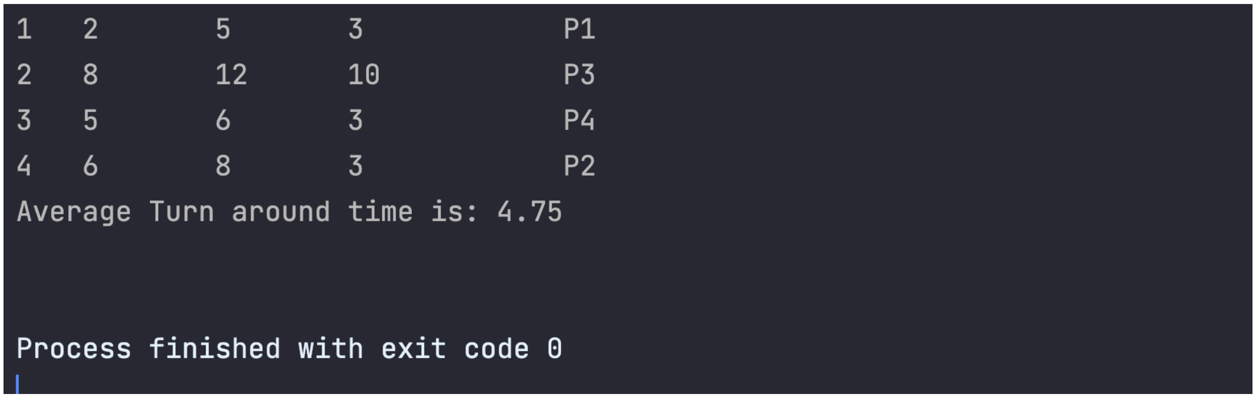
```
/Users/it/Desktop/os/labs/untitled3/cmake-build-debug/untitled3
Enter the number of jobs: 4

Enter the burst time for each process:
Process 1 : 3
Process 2 : 4
Process 3 : 1
Process 4 : 2

Enter the arrival time for each job:
Process 1 : 2
Process 2 : 2
Process 3 : 3
```



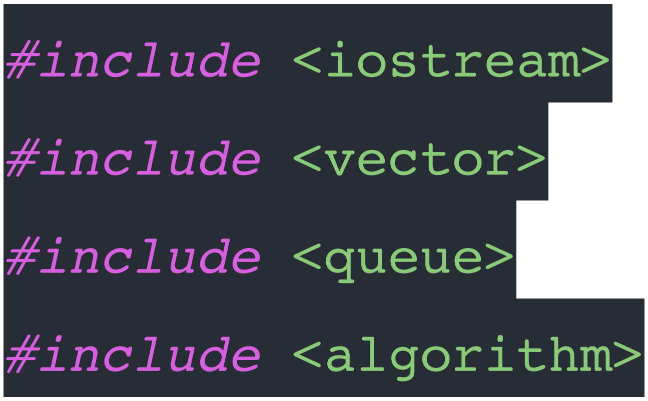
```
Process 4 : 5

Enter the priority for each job:
Process 1 : 3
Process 2 : 2
Process 3 : 1
Process 4 : 3

        ------------- FCFS -------------
PID Initial Start   Completion Time Turn Around Time    Order Of Exec
1   2       5       3            P1
2   5       9       7            P2
3   9       10      7            P3
```



```
4   10      12      7            P4
Average Turn around time is: 6

        ------------- Priority -------------
PID Initial Start   Completion Time Turn Around Time     Order Of Exec
1   2.16262e-314        10      8            P1
2   2.16262e-314        7       5            P1
3   2.16262e-314        4       1            P1
4   2.16262e-314        12      7            P1
Average Turn around time is: 5.25

        ------------- SJN -------------
PID Initial Start   Completion Time Turn Around Time     Order Of Exec
```



```
1   2       5       3            P1
2   8       12      10           P3
3   5       6       3            P4
4   6       8       3            P2
Average Turn around time is: 4.75


Process finished with exit code 0
```



```
#include <iostream>
#include <vector>
#include <queue>
#include <algorithm>
```

```cpp
#include <format>

using namespace std;

vector<class Job> job_vec; //vector of job objects

vector<class Job> setJob(); //creating a setJob function

void printContent(vector<Job> job_vec); //print function
void FCFS(vector<Job> job_vec); //FCFS function
void Priority(vector<Job> job_vec); // Priority algorithm function
void SJN(vector<Job> job_vec); //SJN algorithm function
```

```
class Job //Job class
{

private: //priority code
    int job_id;
    int job_priority;
    double turn_around_time;
    double start_time;
    double completion_time;
    double exec_time;
    double arrival_time;

public:
    Job() { job_id = 0, job_priority = 0, turn_around_time = 0, start_time = 0, completion_time = 0, exec_time = 0; }; //default constructor
    Job(int n, int j)
    {
        job_id = n;
        job_priority = j;
    }

    int getJob_id() //getter function
for job id
    {
        return job_id;
    }

    void setJob_id(int id) //setter
functino for job id
    {
        job_id = id;
    }
    int getArrival_time() const //
getter function for arrivale time
```

```
    {
        return arrival_time;
    }
    void setArrivalTime(int at) //
setter function for arrival time
    {
        arrival_time = at;
    }
    int getJob_priority() const //
getter function for job priority
    {
        return job_priority;
    }
    void setJob_priority(int
job_pr) //setter function for job
prioirty
    {
        job_priority = job_pr;
    }
    double getTurn_around_time() //getter function for turn around time
    {
        return turn_around_time;
    }
    void setTurn_around_time(double t_a) //setter function for turn around time
    {
        turn_around_time = t_a;
    }
    double getStart_time() //getter function for start time
    {
        return start_time;
    }
    void setStart_time(double 
start) //setter functino for start 
time
    {
        start_time = start;
    }
    double getCompletion_time() //
getter function for completion time 
    {
        return completion_time;
    }
    void setCompletion_time(double 
comp) //setter functino for completion 
time
    {
        completion_time = comp;
    }
    double getExec_time() const //
getter function for executino time 
```

```
    {
        return exec_time;
    }
    void setExec_time(double exec) // setter functinon for execution time
    {
        exec_time = exec;
    }

    bool operator==(const Job &other) const
    {
        return job_id == other.job_id;
    }
};

struct compare_fcfs //struct for comparison used in FCFS function
```

```
{
    bool operator()(const Job &lhs, const Job &rhs)
    {
        return lhs.getArrival_time() > rhs.getArrival_time();
    }
};

struct compare_priority //struct for comparison used in priority function
{
    bool operator()(const Job &lhs, const Job &rhs)
    {
        return lhs.getJob_priority() > rhs.getJob_priority();
    }
};
```

```
struct compare_SJN //struct for comparison used in SJN algorithm
{
    bool operator()(const Job &lhs, const Job &rhs)
    {
        return lhs.getExec_time() > rhs.getExec_time();
    }
};

int main()
{
    vector<class Job> job_vec; // creating vetor of Job objects
    job_vec = setJob(); //using function to put info into the vector
    FCFS(job_vec); //calling fcfs function
    Priority(job_vec);//calling priority function
    SJN(job_vec);//calling sjn function

    return 0;
}

vector<Job> setJob() //set job functions
{
    vector<Job> job_vec;
    int arrival_time;
    int burst;
    int priority;
    int num_jobs;
    cout << "Enter the number of jobs: ";
    cin >> num_jobs;

    cout << "\nEnter the burst time for each process: \n";
    int bt;
    for (int i = 0; i < num_jobs; i++) //enetering the burst time for each process into the vector
    {
        cout << "Process " << i + 1 << " : ";
        cin >> bt;
        job_vec.push_back(Job(i, bt)); //adding to the vector
```

```
job_vec[i].setExec_time(bt); //using setter function
    }
    cout << endl;

    cout << "Enter the arrival time for each job: \n";

    for (int i = 0; i < num_jobs; i++) //enetering the arrival time into the vector
    {
        cout << "Process " << i + 1 << " : ";
        cin >> arrival_time;

job_vec[i].setArrivalTime(arrival_time); //using setter function
```

```
    }

    cout << endl;

    cout << "Enter the priority for each job: \n";

    for (int i = 0; i < num_jobs; i++) //adding priority for each job into the vector
    {
        cout << "Process " << i + 1 << " : ";
        cin >> priority;

job_vec[i].setJob_priority(priority); //using setter function
    }
    cout << endl;

    return job_vec;
}

void printContent(vector<class Job>
job_vec) //printing the output
function
{
    int arr[job_vec.size()]; //
creating array the size of the vector
of job objects
    for (int i = 0; i <
job_vec.size(); i++) //traversing
through the vector and adding it to a
array
    {
        arr[i] =
job_vec[i].getStart_time(); //adding
```

```
the start times of each object to the array
    }
    int n = sizeof(arr) / sizeof(arr[0]);
    sort(arr, arr + n); //sorting the array filled with start times of each object

    int arr2[job_vec.size()]; // creating another array
    for (int i = 0; i < sizeof(arr); i++) //traversing throughh the array
    { //arr is sorted and arr2 gets the order of execution by chacking for matching start times
        if (arr[i] == job_vec[i].getStart_time()) //checking to see if the start time in the first
```

```
array is equal to the current start
time
        {
            arr2[i] =
job_vec[i].getJob_id() + 1; //if so
then add to the second array
        }
        else
        {
            for (int j = 0; j <
sizeof(arr); j++) //if not traverse
through the rest of the array to see
where they match
            {
                if (arr[i] ==
job_vec[j].getStart_time())
                {
                    arr2[i] =
job_vec[j].getJob_id() + 1; //if they
match add them to the second array
                }
            }
        }
    }
```

```
    double average_turnaround_time =
0;
    cout << "PID\tInitial
Start\tCompletion Time\tTurn Around
Time\tOrder Of Exec" << endl;
    for (int i = 0; i <
job_vec.size(); i++) //traversing
through the vector
    {
        cout <<
(job_vec[i].getJob_id() + 1) << "\t"
<< job_vec[i].getStart_time() <<
"\t\t" <<
job_vec[i].getCompletion_time() <<
"\t\t" <<
job_vec[i].getTurn_around_time() <<
"\t\t\t P" << arr2[i];
        cout << endl; //printing out
the infromation needed in the output
        average_turnaround_time +=
job_vec[i].getTurn_around_time();
    }

    average_turnaround_time =
average_turnaround_time /
job_vec.size(); //calculating average
turn around time
    cout << "Average Turn around time
is: " << average_turnaround_time <<
endl; //printing out the average turn
around time
```

```
}

void FCFS(vector<class Job> job_vec) //FCFS function
{
    int clock = 2; //initialize the clock variable
    priority_queue<Job, vector<Job>, compare_fcfs> FCFS; //creating a queue that will be used and takes in the created compare struct and vector

    for (int i = 0; i < job_vec.size(); i++) //traversing through the vector and adding the values to the queue
    {
        FCFS.push(job_vec[i]);
    }
    while (!FCFS.empty()) //loop to go
while the queue is not empty
    {
        Job target = FCFS.top(); //
variable that is set to the top of the
queue
        for (int i = 0; i <
job_vec.size(); i++) //loop to
traverse through the vector
        {
            if (target ==
job_vec[i]) //checks if the top of the
queue is equal to the job object
            {
                if
(job_vec[i].getArrival_time() > clock)
//checking to see if the object
arrival time is less than the clock
value
```

```
                {

job_vec[i].setArrivalTime(clock); //if
so, setting the arrival time to the
value of the clock
                }

job_vec[i].setStart_time(clock); //
setting the start time of the job
object to the clock value
                clock +=
job_vec[i].getExec_time(); //
incrementing clock by the execution
time of the job object

job_vec[i].setCompletion_time(clock);
//setting the completion time of a job
object to the value of the clock

job_vec[i].setTurn_around_time(job_vec
[i].getCompletion_time() -
job_vec[i].getArrival_time());//
setting the turn around time of a job
object
            }
        }
        FCFS.pop(); //popping off the
stack
    }
    cout << "\t\t------------- FCFS
-------------" << endl;
    printContent(job_vec); //printing
out the final vales by calling the
print function
    cout << endl;
}

void Priority(vector<Job> job_vec) //
priority algorithm function
```

```
{
    int clock = 0; //initializing the clock variable
    priority_queue<Job, vector<Job>, compare_priority> Priority_Queue; // creating a queue that takes in the vector and the comparison struct
    Job running; //creating job object
    running.setJob_id(-1); //setting the id of the object to -1
    while (clock < 20) //loop while the clock value is less than 20
    {
        for (int i = 0; i < job_vec.size(); i++) //traversing through the size of the vector
        {
            if (job_vec[i].getArrival_time() ==
clock) //if the arrival time of a
object matches the clock value
            {

Priority_Queue.push(job_vec[i]); //
pushing that value to the queue
            }
        }

        if (Priority_Queue.size() > 0)
//if the queue is not empty
        {
            if (running.getJob_id() ==
-1) //if the job id of the job oject
is -1
            {
                running =
Priority_Queue.top(); //setting the
job object to the top of the queue
```

```
                if
(running.getStart_time() == 0) //if
the start time of a job object is
equal to zero
                {

running.setStart_time(clock); //
setting the start time of a job obect
to the value of the clock variable
                }

Priority_Queue.pop(); //popping the
value off the queue
            }
            else if
(running.getJob_priority() >
Priority_Queue.top().getJob_priority()
) //comapring the job priority of the
running object to the job priority of
the object at the top of the queue
```

```
            {

Priority_Queue.push(running); //
pushing the running job object to the
queue
                running =
Priority_Queue.top(); //the running
object is now at the to pof the queue
                if
(running.getStart_time() == 0) //
checking if the startime of the
running object is zero
                {

running.setStart_time(clock); //
setting the starttime of the running
object tot the value of the clock
variable
                }
Priority_Queue.pop(); //popping off
the queue
            }
        }
        clock++; //incrementing the
clock
        if (running.getJob_id() != -1)
//checking of the job id of the
running object is not -1
        {
            int x =
running.getExec_time() - 1;

running.setExec_time(x); //setting the
execution time of the running object
to its original value subtracted by 1
```

```
            if (running.getExec_time()
== 0) //checking if the execution time
of the running object is zero
            {
                for (int i = 0; i <
job_vec.size(); i++) //traversing
through the size of the vector
                {
                    if (running ==
job_vec[i]) //checking to see if the
running object is equal to a object in
the vector
                    {

job_vec[i].setCompletion_time(clock);
//setting the completion time of a job
object to the value of the clock

job_vec[i].setTurn_around_time(job_vec
[i].getCompletion_time() -
job_vec[i].getArrival_time()); //
setting the turn around time of a
object

job_vec[i].setStart_time(running.getSt
art_time());//setting the start time
of a object to the start time of the
running object
                    }
                }
                running.setJob_id(-1);
//setting the job id of the running
object
            }
        }
    }
    cout << "\t\t-------------
Priority -------------" << endl;
```

```
    printContent(job_vec); //printing out the infomration relating to the algorithm
    cout << endl;
}

void SJN(vector<Job> job_vec) //SJN algorithm
{
    int clock = 0;//initializing the clock value
    priority_queue<Job, vector<Job>, compare_SJN> SJN_Queue; //creating a queue for SJN that takes in the vector as well as the comparison struct made for this algorithm
    Job running; //creating a job object called running
    running.setJob_id(-1); //setting the job id of the obect to -1
    priority_queue<Job, vector<Job>, compare_fcfs> FCFS_queue;//creating a queue for FCFS that takes in the vector as well as the comparison struct made for this algorithm

    for (int i = 0; i < job_vec.size(); i++) //traversing through the vector
    {

FCFS_queue.push(job_vec[i]); //adding the values to the FCFS queue
    }

    while (clock < 20) //chcing while the value of clock is less than 20
    {
        while (!FCFS_queue.empty() && FCFS_queue.top().getArrival_time() <=
clock) //while the FCFS queue is not
emty
        {
            for (int i = 0; i <
job_vec.size(); i++) //traversing
through the size of the vector
            {
                if (FCFS_queue.top()
== job_vec[i]) //checking if the top
of the queue is equal to the job
object in the vector
                {

SJN_Queue.push(job_vec[i]); //if so,
push that object to the SJN queue
                }
            }
            FCFS_queue.pop(); //
popping off the FCFS queue to get the
next object
```

```
        }

        if (SJN_Queue.size() > 0) // checking that the SJN queue is not empty
        {
            for (int i = 0; i < job_vec.size(); i++) //traversing through the size of the vector
            {
                if (SJN_Queue.top() == job_vec[i]) //checking if the top of the queue is equal to the job object in the vector
                {

job_vec[i].setStart_time(clock); // seting the start time of the job object to the value of the clock
```

```
                    clock +=
job_vec[i].getExec_time(); //updating
the clock value by adding the
execution time of the job object

job_vec[i].setCompletion_time(clock);
//setting the completion time of a job
object to the value of the clock
variable

job_vec[i].setTurn_around_time(job_vec
[i].getCompletion_time() -
job_vec[i].getArrival_time());//
setting the turn around time of a job
obect

SJN_Queue.pop(); //popping off the SJN
queue
                    break;
                }
```

```
            }
        }
        else
        {
            clock++; //incrementing the clock value
        }
    }
    cout << "\t\t------------- SJN -------------" << endl;
    printContent(job_vec); //printing out the information
    cout << endl;
}
```