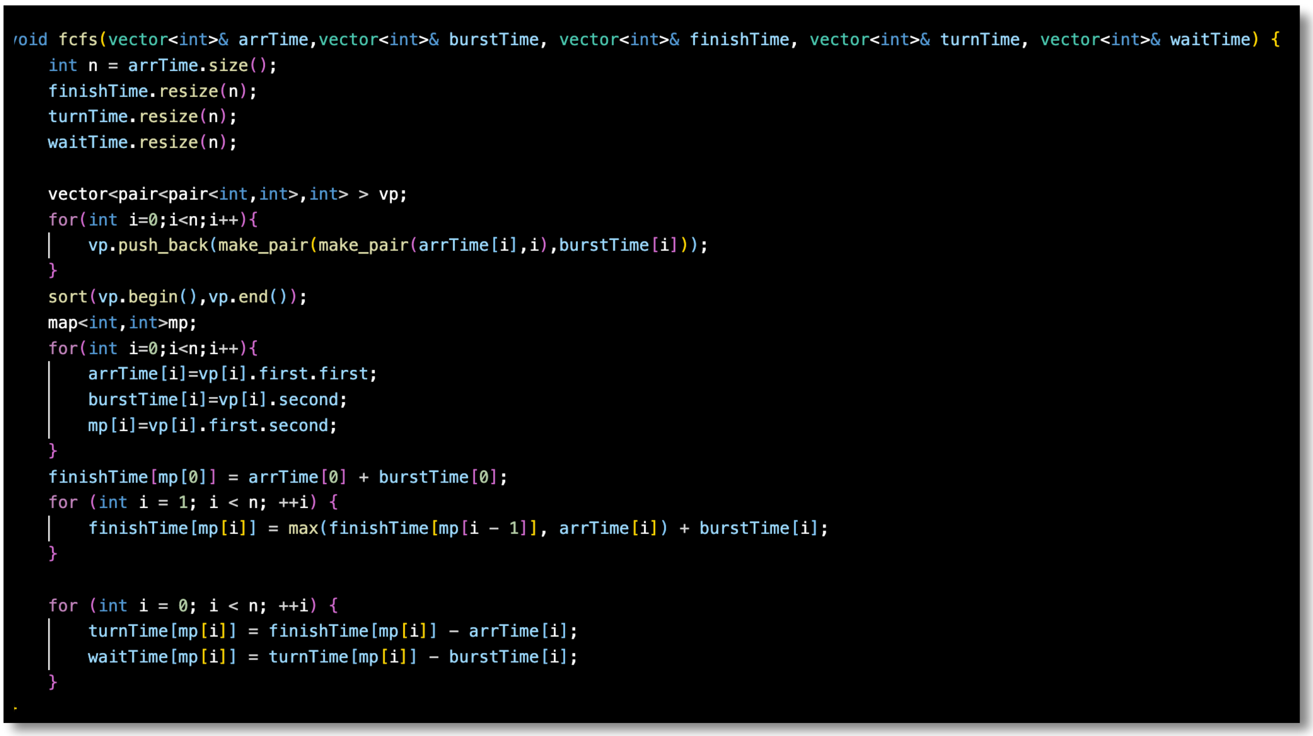
CPU-Scheduler

In this project we had to implement various CPU-scheduling algorithms and on the basis of various parameters like Finishtime,Turnaroundtime,waitime had to decide which is suitable for the given set of input.The paramerts of input which I have considered in my program are the arrival time,burst time, priority of an process and the timescale. On basis of these given inputs I have implemented First Come First Serve(FCFS),Shortest Job First(SJF), Priority Scheduling and Round Robin(RR) algorithm.

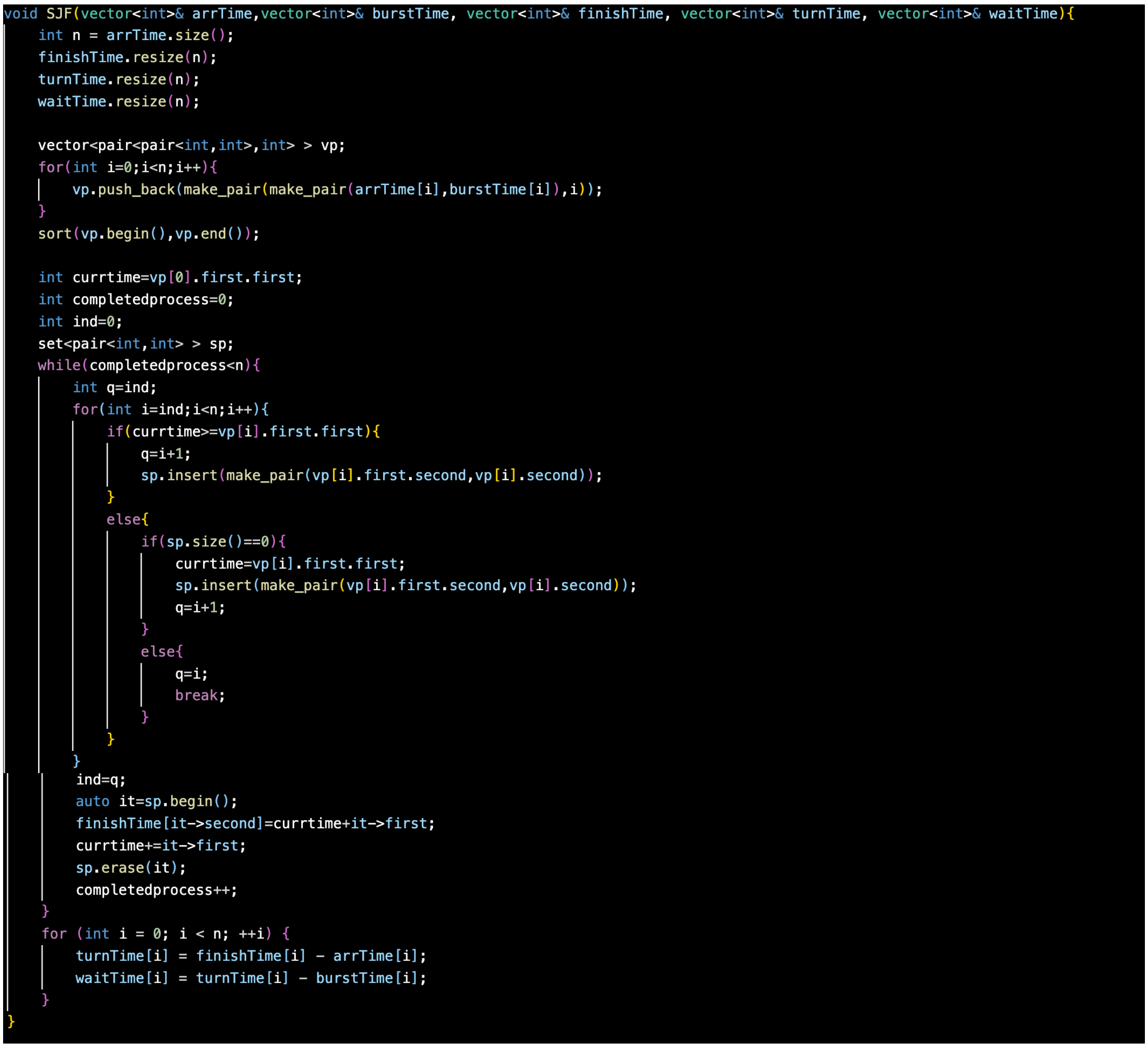
FCFS Algorithm:-



```
oid fcfs(vector<int>& arrTime,vector<int>& burstTime, vector<int>& finishTime, vector<int>& turnTime, vector<int>& waitTime) {
    int n = arrTime.size();
    finishTime.resize(n);
    turnTime.resize(n);
    waitTime.resize(n);

    vector<pair<pair<int,int>,int> > vp;
    for(int i=0;i<n;i++){
        vp.push_back(make_pair(make_pair(arrTime[i],i),burstTime[i]));
    }
    sort(vp.begin(),vp.end());
    map<int,int>mp;
    for(int i=0;i<n;i++){
        arrTime[i]=vp[i].first.first;
        burstTime[i]=vp[i].second;
        mp[i]=vp[i].first.second;
    }
    finishTime[mp[0]] = arrTime[0] + burstTime[0];
    for (int i = 1; i < n; ++i) {
        finishTime[mp[i]] = max(finishTime[mp[i - 1]], arrTime[i]) + burstTime[i];
    }

    for (int i = 0; i < n; ++i) {
        turnTime[mp[i]] = finishTime[mp[i]] - arrTime[i];
        waitTime[mp[i]] = turnTime[mp[i]] - burstTime[i];
    }
```

The logic which I have used here is that I have intitally sorted all the incoming process according to their time of arrival first. Then according to FCFS whichever process comes first is executed. So the processes are being executed in the same order as their arrival time and if two processes are having the same arrival time then the process first in the input is executed first.I'm calculating the finish time while running the algorithm and once the finish time is calculated then I am using the expression to calculate the Turnaround time and the Wait time of the process.Turnaround time is the amount of time taken to complete or fulfill a process.Wait time is defined as the time a process has waited for other process executing.It can have the problem of process starvation in case when a large process comes in the very beginning of the execution.

SJF Algorithm:-

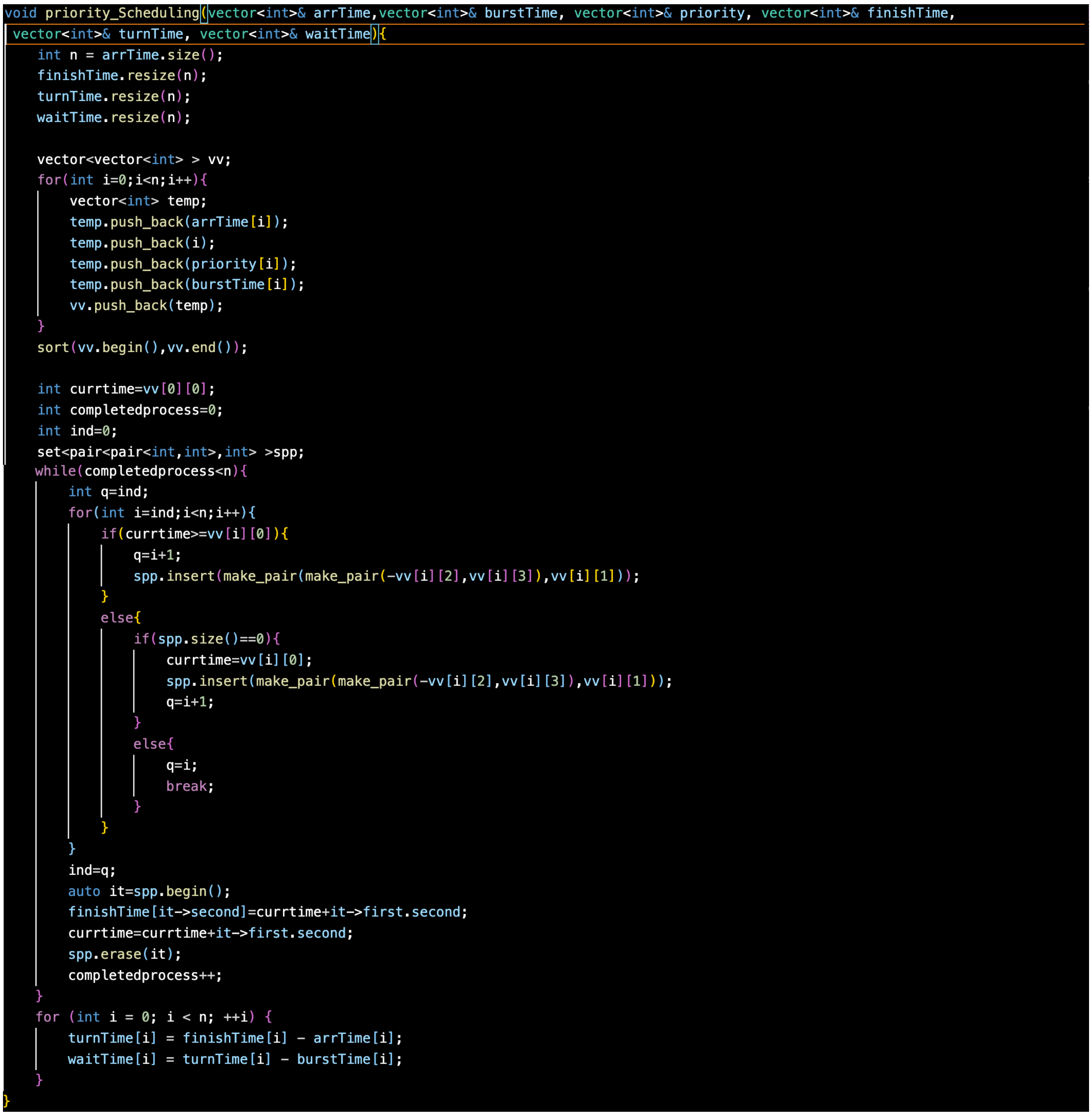


```cpp
void SJF(vector<int>& arrTime,vector<int>& burstTime, vector<int>& finishTime, vector<int>& turnTime, vector<int>& waitTime){
    int n = arrTime.size();
    finishTime.resize(n);
    turnTime.resize(n);
    waitTime.resize(n);

    vector<pair<pair<int,int>,int> > vp;
    for(int i=0;i<n;i++){
        vp.push_back(make_pair(make_pair(arrTime[i],burstTime[i]),i));
    }
    sort(vp.begin(),vp.end());

    int currtime=vp[0].first.first;
    int completedprocess=0;
    int ind=0;
    set<pair<int,int> > sp;
    while(completedprocess<n){
        int q=ind;
        for(int i=ind;i<n;i++){
            if(currtime>=vp[i].first.first){
                q=i+1;
                sp.insert(make_pair(vp[i].first.second,vp[i].second));
            }
            else{
                if(sp.size()==0){
                    currtime=vp[i].first.first;
                    sp.insert(make_pair(vp[i].first.second,vp[i].second));
                    q=i+1;
                }
                else{
                    q=i;
                    break;
                }
            }
        }
        ind=q;
        auto it=sp.begin();
        finishTime[it->second]=currtime+it->first;
        currtime+=it->first;
        sp.erase(it);
        completedprocess++;
    }
    for (int i = 0; i < n; ++i) {
        turnTime[i] = finishTime[i] - arrTime[i];
        waitTime[i] = turnTime[i] - burstTime[i];
    }
}
```

The logic is that first I have sorted the inputs according to the time of arrival. Then at a given instant of time whichever is the shortest job ready to be executed the scheduler executes it and increases the time by the burst time of the executed process.In this time span if any other process are also ready to be executed they are added and again the shortest job is selected from the available process. I have implemented this using the "Set" data structure in c++.The finish time is calculated during the execution and then using the finish time and the expression available I am calculating the values of the Turnaround time and waittime of each process in the input.

Scope of improvement-I have implemeted a non-preemptive SJF algorithm, can implemented a preemptive SJF algorithm for even better results from SJF algorithm.

Priority Scheduling:-

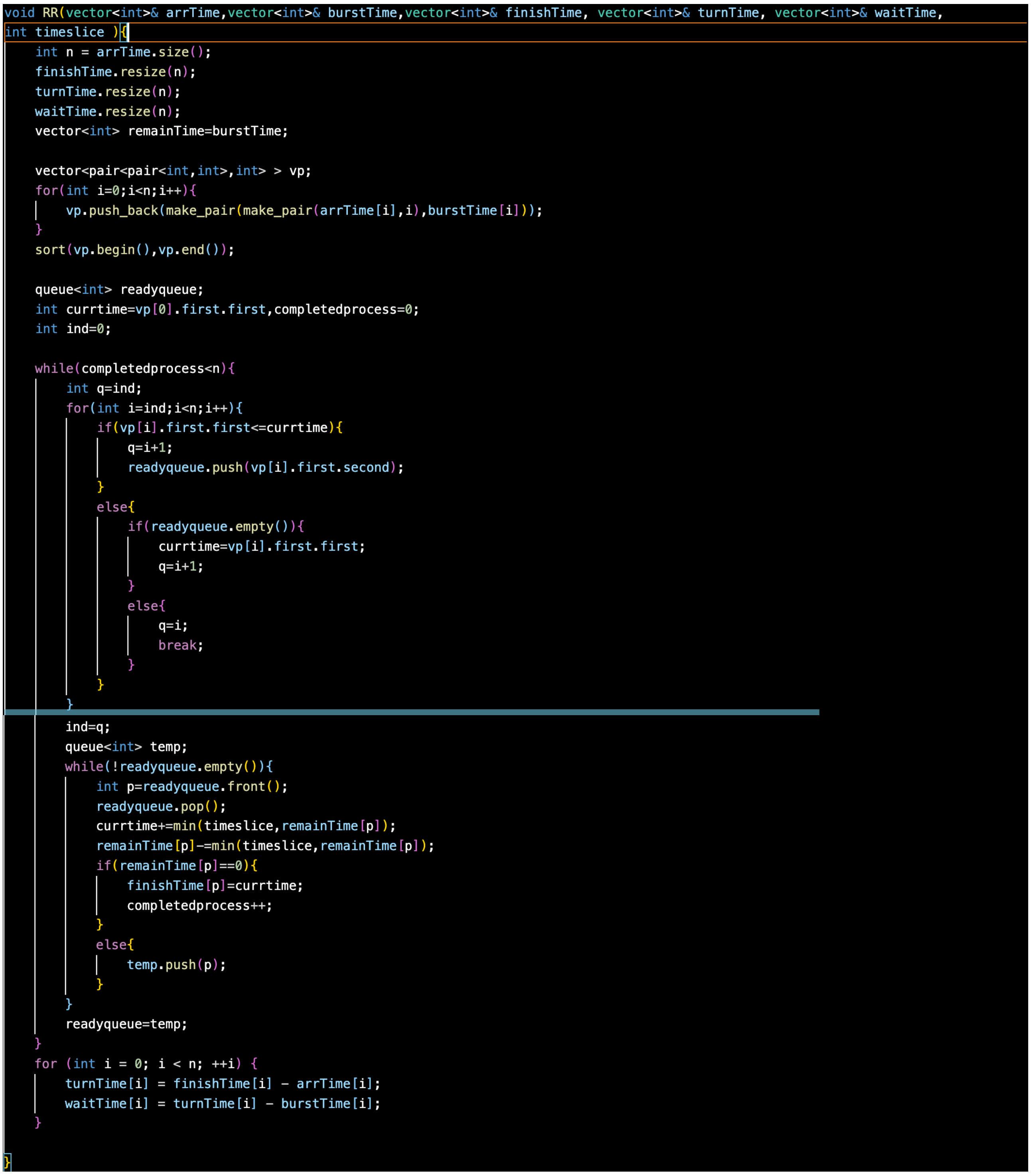


```cpp
void priority_Scheduling(vector<int>& arrTime,vector<int>& burstTime, vector<int>& priority, vector<int>& finishTime,
 vector<int>& turnTime, vector<int>& waitTime){
    int n = arrTime.size();
    finishTime.resize(n);
    turnTime.resize(n);
    waitTime.resize(n);

    vector<vector<int> > vv;
    for(int i=0;i<n;i++){
        vector<int> temp;
        temp.push_back(arrTime[i]);
        temp.push_back(i);
        temp.push_back(priority[i]);
        temp.push_back(burstTime[i]);
        vv.push_back(temp);
    }
    sort(vv.begin(),vv.end());

    int currtime=vv[0][0];
    int completedprocess=0;
    int ind=0;
    set<pair<pair<int,int>,int> >spp;
    while(completedprocess<n){
        int q=ind;
        for(int i=ind;i<n;i++){
            if(currtime>=vv[i][0]){
                q=i+1;
                spp.insert(make_pair(make_pair(-vv[i][2],vv[i][3]),vv[i][1]));
            }
            else{
                if(spp.size()==0){
                    currtime=vv[i][0];
                    spp.insert(make_pair(make_pair(-vv[i][2],vv[i][3]),vv[i][1]));
                    q=i+1;
                }
                else{
                    q=i;
                    break;
                }
            }
        }
        ind=q;
        auto it=spp.begin();
        finishTime[it->second]=currtime+it->first.second;
        currtime=currtime+it->first.second;
        spp.erase(it);
        completedprocess++;
    }
    for (int i = 0; i < n; ++i) {
        turnTime[i] = finishTime[i] - arrTime[i];
        waitTime[i] = turnTime[i] - burstTime[i];
    }
}
```

First I have sorted all the process according to their arrival time given in the input. At a given point of time among all the processes the process with highest priority is executed first. The time is increased by the burst time of process being executed.In the time span if any other processes are ready they are added and then the process with highest priority is chosen from the available process in the set . I have considered that the process with higher priority is to be excuted first.The finish time is calculated during the execution and then using the finish time and the expression available I am calculating the values of the Turnaround time and waittime of each process in the input.

Scope of improvement-I have implemented a non-preemptive Priority scheduling but can implement a preemptive Priority scheduling for even better results. Also I can take input from the user whether he wants to give high priority to processes with higher magnitude or lower magnitude. I have by deafult considered higher numbers have higher priority.

Round-Robin:-



```
void RR(vector<int>& arrTime,vector<int>& burstTime,vector<int>& finishTime, vector<int>& turnTime, vector<int>& waitTime,
int timeslice ){
    int n = arrTime.size();
    finishTime.resize(n);
    turnTime.resize(n);
    waitTime.resize(n);
    vector<int> remainTime=burstTime;

    vector<pair<pair<int,int>,int> > vp;
    for(int i=0;i<n;i++){
        vp.push_back(make_pair(make_pair(arrTime[i],i),burstTime[i]));
    }
    sort(vp.begin(),vp.end());

    queue<int> readyqueue;
    int currtime=vp[0].first.first,completedprocess=0;
    int ind=0;

    while(completedprocess<n){
        int q=ind;
        for(int i=ind;i<n;i++){
            if(vp[i].first.first<=currtime){
                q=i+1;
                readyqueue.push(vp[i].first.second);
            }
            else{
                if(readyqueue.empty()){
                    currtime=vp[i].first.first;
                    q=i+1;
                }
                else{
                    q=i;
                    break;
                }
            }
        }
        ind=q;
        queue<int> temp;
        while(!readyqueue.empty()){
            int p=readyqueue.front();
            readyqueue.pop();
            currtime+=min(timeslice,remainTime[p]);
            remainTime[p]-=min(timeslice,remainTime[p]);
            if(remainTime[p]==0){
                finishTime[p]=currtime;
                completedprocess++;
            }
            else{
                temp.push(p);
            }
        }
        readyqueue=temp;
    }
    for (int i = 0; i < n; ++i) {
        turnTime[i] = finishTime[i] - arrTime[i];
        waitTime[i] = turnTime[i] - burstTime[i];
    }

}
```

First I have sorted all the process according to their arrival time given in the input. At a given time all process ready are added to readyqueue. All the processes in ready queue will be executedd for time given by timeslice and then the next process in the queue is executed. Meanwhile any other process ready are also added to the queue for scheduilng. This helps us to reduce the response time for the processes but increases the finish time of all the processes.

How to run The Project:-

- Clone the repo from the GitHub repo link and run the codde for the given input file .

Learning Takeaways:-

- Learnt a lot of various scheduling algorithms , The various advantage and disadvanatges of the algorithms and how better algorithms have kept coming over time to solve various issues.
- Learnt about various terms used in OS scheduling and their importance when deciding upon the best scheduler.
- Learnt how imporatant scheduler is in OS so that all tasks are efficiently executed and the user doesnot have to wait for long in any case.
- Learnt to implement neat and clean code in cpp for various algorithms in particular and c++ in general.
- Learnt to integrate the front end and the backend so that the data from the backend can be shown to the user in the frontend.

References:-

- https://www.youtube.com/playlistlist=PLBlnK6fEyqRitWSE_AyyySWfhRgyA-rHk

- https://www.doc-developpement-durable.org/file/Projets-informatiques/cours-&-manuels-informatiques/Linux/Linux%20Kernel%20Development,%203rd%20Edition.pdf

- https://www.researchgate.net/publication/49619229_An_Improved_Round_Robin_Schedduling_Algorithm_for_CPU_Scheduling