```cpp
#include<bits/stdc++.h>
using namespace std;
struct process {

    pid_t p_no = 0;
    time_t start_AT = 0, AT = 0,
        BT_left = 0, BT = 0, temp_BT = 0,
        CT = 0, TAT = 0, WT = 0, RT = 0;
    int priority = 0;

    void set_CT(time_t time)
    {
        CT = time;
        set_TAT();
        set_WT();
    }

    void set_TAT()
    {
        TAT = CT - start_AT;
    }

    void set_WT()
    {
        WT = TAT - BT;
    }
    void P_set()
    {
        start_AT = AT;
        BT_left = BT;
    }
    void set_RT(time_t time)
    {
        RT = time - start_AT;
    }


    friend bool operator<(const process& a, const process& b)
    {
        return a.AT > b.AT;
    }
};

process pop_index(priority_queue<process>* main_queue,
,                int index)
{
    priority_queue<process> rm_index;
    int i;
    process p;

    switch (index) {
    case 0:
        p = (*main_queue).top();
        (*main_queue).pop();
        break;
    default:
        for (i = 0; i < index; i++) {
            rm_index.push((*main_queue).top());
            (*main_queue).pop();
        }
        p = (*main_queue).top();
        (*main_queue).pop();

        while (!(*main_queue).empty()) {
            rm_index.push((*main_queue).top());
            (*main_queue).pop();
        }
        (*main_queue) = rm_index;
        break;
    }
    return p;
}


int max_priority(priority_queue<process> main_priority_queue,
,               int limit, bool high)
{
    int max = -1;
    if (high == 1) {
        while (!main_priority_queue.empty()
            && main_priority_queue.top().AT <= limit) {
            if (main_priority_queue.top().priority > max)
                max = main_priority_queue.top().priority;
            main_priority_queue.pop();
        }
    }
    else {
        while (!main_priority_queue.empty()
            && main_priority_queue.top().AT <= limit) {
            if (max == -1 || main_priority_queue.top().priority < max)
                max = main_priority_queue.top().priority;
            main_priority_queue.pop();
        }
    }
    return max;
}

int max_priority_index(priority_queue<process> main_queue, int limit, bool high)
{
    int max = -1, i = 0, index = 0;
    if (high == 1) {
        while (!main_queue.empty() && main_queue.top().AT <= limit) {
            if (main_queue.top().priority > max) {
                max = main_queue.top().priority;
                index = i;
            }
            main_queue.pop();
            i++;
        }
    }
    else {
        while (!main_queue.empty()
            && main_queue.top().AT <= limit) {
            if (max == -1 || main_queue.top().priority < max) {
                max = main_queue.top().priority;
                index = i;
            }
            main_queue.pop();
            i++;
        }
    }
    return index;
```

```
}

priority_queue<process> Priority_P_run(priority_queue<process> ready_queue,
queue<process>* gantt, bool high)
{
    int temp;
    priority_queue<process> completion_queue;
    process p;
    time_t clock = 0;
    if (high == 1) {
        while (!ready_queue.empty()) {
            while (clock < ready_queue.top().AT) {
                p.temp_BT++;
                clock++;
            }
            if (p.temp_BT > 0) {
                p.p_no = -1;
                p.CT = clock;
                (*gantt).push(p);
            }
            p = pop_index(&ready_queue,
                        max_priority_index(ready_queue, clock, high));
            if (p.AT == p.start_AT)
                p.set_RT(clock);
            while (p.BT_left > 0
                && (ready_queue.empty()
                    || clock < ready_queue.top().AT
                    || p.priority >= max_priority(ready_queue, clock, high))) {
                p.temp_BT++;
                p.BT_left--;
                clock++;
            }
            if (p.BT_left == 0) {
                p.AT = p.start_AT;
                p.set_CT(clock);
                (*gantt).push(p);
                p.temp_BT = 0;
                completion_queue.push(p);
            }
            else {
                p.AT = clock;
                p.CT = clock;
                (*gantt).push(p);
                p.temp_BT = 0;
                ready_queue.push(p);
            }
        }
    }
    else {
        while (!ready_queue.empty()) {
            while (clock < ready_queue.top().AT) {
                p.temp_BT++;
                clock++;
            }
            if (p.temp_BT > 0) {
                p.p_no = -1;
                p.CT = clock;
                (*gantt).push(p);
            }
            p = pop_index(&ready_queue,
                        max_priority_index(ready_queue,
```

```cpp
                                            clock, high));

            if (p.AT == p.start_AT)
                p.set_RT(clock);
            temp = max_priority(ready_queue, clock, high);

            while (p.BT_left > 0 && (ready_queue.empty()
                                    || clock < ready_queue.top().AT
                                    || p.priority <= max_priority(ready_queue, clock, high))) {
                p.temp_BT++;
                p.BT_left--;
                clock++;
            }
            if (p.BT_left == 0) {
                p.AT = p.start_AT;
                p.set_CT(clock);
                (*gantt).push(p);
                p.temp_BT = 0;
                completion_queue.push(p);
            }
            else {
                p.AT = clock;
                p.CT = clock;
                (*gantt).push(p);
                p.temp_BT = 0;
                ready_queue.push(p);
            }
        }
    }

    return completion_queue;
}

priority_queue<process> set_sample_data()
{
    priority_queue<process> ready_queue;
    int n;
    cout<<"\nEnter the number of processes: "; cin>>n;

    for(int i=0;i<n;i++)
    {
          cout<<"\nEnter arrival time ,burst time and priority of process "<<i+1<<": ";
          process temp;
            cin>>temp.AT>>temp.BT>>temp.priority;
            temp.p_no = i+1;
            temp.P_set();
            ready_queue.push(temp);

    }
    cout<<"\n";

    return ready_queue;
}

double get_total_WT(priority_queue<process> processes)
{
    double total = 0;
    while (!processes.empty()) {
        total += processes.top().WT;
        processes.pop();
    }
    return total;
}

double get_total_TAT(priority_queue<process> processes)
{
    double total = 0;
    while (!processes.empty()) {
        total += processes.top().TAT;
        processes.pop();
    }
    return total;
}

double get_total_CT(priority_queue<process> processes)
{
    double total = 0;
    while (!processes.empty()) {
        total += processes.top().CT;
        processes.pop();
    }
    return total;
}

double get_total_RT(priority_queue<process> processes)
{
    double total = 0;
    while (!processes.empty()) {
        total += processes.top().RT;
        processes.pop();
    }
    return total;
}

void disp(priority_queue<process> main_queue, bool high)
{
    int i = 0, temp, size = main_queue.size();
    priority_queue<process> tempq = main_queue;
    double temp1;
    cout << "+-------------+--------------";
    cout << "+------------+-----------------";
    cout << "+-----------------+--------------+---------------+";
    if (high == true)
        cout << "----------+" << endl;
    else
        cout << endl;
    cout << "| Process No. | Arrival Time ";
    cout << "| Burst Time | Completion Time ";
    cout << "| Turnaround Time | Waiting Time | Response Time |";
    if (high == true)
        cout << " Priority |" << endl;
    else
        cout << endl;
    cout << "+-------------+--------------";
    cout << "+------------+-----------------";
    cout << "+-----------------+--------------+---------------+";
    if (high == true)
        cout << "----------+" << endl;
    else
        cout << endl;
    while (!main_queue.empty()) {
        temp = to_string(main_queue.top().p_no).length();
        cout << '|' << string(6 - temp / 2 - temp % 2, ' ')
            << main_queue.top().p_no << string(7 - temp / 2, ' ');
        temp = to_string(main_queue.top().start_AT).length();
        cout << '|' << string(7 - temp / 2 - temp % 2, ' ')
            << main_queue.top().start_AT << string(7 - temp / 2, ' ');
        temp = to_string(main_queue.top().BT).length();
        cout << '|' << string(6 - temp / 2 - temp % 2, ' ')
            << main_queue.top().BT << string(6 - temp / 2, ' ');
        temp = to_string(main_queue.top().CT).length();
        cout << '|' << string(8 - temp / 2 - temp % 2, ' ')
            << main_queue.top().CT << string(9 - temp / 2, ' ');
        temp = to_string(main_queue.top().TAT).length();
        cout << '|' << string(8 - temp / 2 - temp % 2, ' ')
            << main_queue.top().TAT << string(9 - temp / 2, ' ');
        temp = to_string(main_queue.top().WT).length();
        cout << '|' << string(7 - temp / 2 - temp % 2, ' ')
            << main_queue.top().WT << string(7 - temp / 2, ' ');
        temp = to_string(main_queue.top().RT).length();
        cout << '|' << string(7 - temp / 2 - temp % 2, ' ')
            << main_queue.top().RT << string(8 - temp / 2, ' ');
        if (high == true) {
            temp = to_string(main_queue.top().priority).length();
            cout << '|' << string(5 - temp / 2 - temp % 2, ' ')
                << main_queue.top().priority << string(5 - temp / 2, ' ');
        }
        cout << "|\n";
        main_queue.pop();
    }
    cout << "+-------------+--------------";
    cout << "+------------+----------------";
    cout << "+----------------+--------------+--------------+";
    if (high == true)
        cout << "----------+";
    cout << endl;
    temp1 = get_total_CT(tempq);
    cout << "\nTotal completion time :- " << temp1 << endl;
    cout << "Average completion time :- " << temp1 / size << endl;
    temp1 = get_total_TAT(tempq);
    cout << "\nTotal turnaround time :- " << temp1 << endl;
    cout << "Average turnaround time :- " << temp1 / size << endl;
    temp1 = get_total_WT(tempq);
    cout << "\nTotal waiting time :- " << temp1 << endl;
    cout << "Average waiting time :- " << temp1 / size << endl;
    temp1 = get_total_RT(tempq);
    cout << "\nTotal response time :- " << temp1 << endl;
    cout << "Average response time :- " << temp1 / size << endl;
}

void disp_gantt_chart(queue<process> gantt)
{
    int temp, prev = 0;
    queue<process> spaces = gantt;
    cout << "\n\nGantt Chart (IS indicates ideal state) :- \n\n+";
    while (!spaces.empty()) {
        cout << string(to_string(spaces.front().p_no).length() +
(spaces.front().p_no != -1) + 2 * spaces.front().temp_BT, '-') << "+";
        spaces.pop();
    }
    cout << "\n|";
    spaces = gantt;
    while (!spaces.empty()) {
        cout << string(spaces.front().temp_BT, ' ');
        if (spaces.front().p_no == -1)
            cout << "IS" << string(spaces.front().temp_BT, ' ') << '|';
        else
            cout << "P" << spaces.front().p_no
                << string(spaces.front().temp_BT, ' ') << '|';
        spaces.pop();
    }
    spaces = gantt;
    cout << "\n+";
    while (!spaces.empty()) {
        cout << string(to_string(spaces.front().p_no).length() +
(spaces.front().p_no != -1) + 2 * spaces.front().temp_BT, '-')
            << "+";
        spaces.pop();
    }
    spaces = gantt;
    cout << "\n0";
    while (!spaces.empty()) {
        temp = to_string(spaces.front().CT).length();
        cout << string(to_string(spaces.front().p_no).length() +
(spaces.front().p_no != -1) + 2 * spaces.front().temp_BT - temp / 2 - prev, ' ')
            << spaces.front().CT;
        prev = temp / 2 - temp % 2 == 0;
        spaces.pop();
    }
    cout << "\n\n";
}

int main()
{
    priority_queue<process> ready_queue, completion_queue;

    queue<process> gantt;

    ready_queue = set_sample_data();

    completion_queue = Priority_P_run(ready_queue, &gantt, true);


    disp(completion_queue, true);


    disp_gantt_chart(gantt);
    return 0;
}
```