```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;
```

```
123 }
124
125 priority_queue<process> Priority_NP_run(priority_queue<process> ready_queue,
126 ,                                       queue<process>* gantt, bool high)
127 {
128     priority_queue<process> completion_queue;
129     process p;
130     time_t clock = 0;
131     if (high == 1) {
132         while (!ready_queue.empty()) {
133             while (clock < ready_queue.top().AT) {
134                 p.temp_BT++;
135                 clock++;
136             }
137             if (p.temp_BT > 0) {
138                 p.p_no = -1;
139                 p.CT = clock;
140                 (*gantt).push(p);
141             }
142             p = pop_index(&ready_queue,
143                           max_priority_index(ready_queue,
144                                              clock, high));
145             p.set_RT(clock);
146
147             while (p.BT_left > 0) {
148                 p.temp_BT++;
149                 p.BT_left--;
150                 clock++;
151             }
152             p.set_CT(clock);
153             (*gantt).push(p);
154             p.temp_BT = 0;
155
156             completion_queue.push(p);
157         }
158     }
159     else {
160         while (!ready_queue.empty()) {
161             while (clock < ready_queue.top().AT) {
162                 p.temp_BT++;
163                 clock++;
164             }
165             if (p.temp_BT > 0) {
166                 p.p_no = -1;
167                 p.CT = clock;
168                 (*gantt).push(p);
169             }
170             p = pop_index(&ready_queue,
171                           max_priority_index(ready_queue,
172                                              clock, high));
173             p.set_RT(clock);
174
175             while (p.BT_left > 0) {
176                 p.temp_BT++;
177                 p.BT_left--;
178                 clock++;
179             }
180             p.set_CT(clock);
181             (*gantt).push(p);
182             p.temp_BT = 0;
183
```

```
184             completion_queue.push(p);
185         }
186     }
187     return completion_queue;
188 }
189 
190 priority_queue<process> set_sample_data()
191 {
192     priority_queue<process> ready_queue;
193     int n;
194     cout<<"\nEnter the number of processes: "; cin>>n;
195 
196     for(int i=0;i<n;i++)
197     {
198           cout<<"\nEnter arrival time ,burst time and priority of process "<<i+1<<"
    : ";
199           process temp;
200             cin>>temp.AT>>temp.BT>>temp.priority;
201             temp.p_no = i+1;
202             temp.P_set();
203             ready_queue.push(temp);
204 
205     }
206     cout<<"\n";
207 
208     return ready_queue;
209 }
210 
211 double get_total_WT(priority_queue<process> processes)
212 {
213     double total = 0;
214     while (!processes.empty()) {
215         total += processes.top().WT;
216         processes.pop();
217     }
218     return total;
219 }
220 
221 double get_total_TAT(priority_queue<process> processes)
222 {
223     double total = 0;
224     while (!processes.empty()) {
225         total += processes.top().TAT;
226         processes.pop();
227     }
228     return total;
229 }
230 
231 double get_total_CT(priority_queue<process> processes)
232 {
233     double total = 0;
234     while (!processes.empty()) {
235         total += processes.top().CT;
236         processes.pop();
237     }
238     return total;
239 }
240 
241 double get_total_RT(priority_queue<process> processes)
242 {
243     double total = 0;
```

```
244     while (!processes.empty()) {
245         total += processes.top().RT;
246         processes.pop();
247     }
248     return total;
249 }
250 
251 void disp(priority_queue<process> main_queue, bool high)
252 {
253     int i = 0, temp, size = main_queue.size();
254     priority_queue<process> tempq = main_queue;
255     double temp1;
256     cout << "+-------------+--------------";
257     cout << "+------------+-----------------";
258     cout << "+-----------------+--------------+---------------+";
259     if (high == true)
260         cout << "----------+" << endl;
261     else
262         cout << endl;
263     cout << "| Process No. | Arrival Time ";
264     cout << "| Burst Time | Completion Time ";
265     cout << "| Turnaround Time | Waiting Time | Response Time |";
266     if (high == true)
267         cout << " Priority |" << endl;
268     else
269         cout << endl;
270     cout << "+-------------+--------------";
271     cout << "+------------+-----------------";
272     cout << "+-----------------+--------------+---------------+";
273     if (high == true)
274         cout << "----------+" << endl;
275     else
276         cout << endl;
277     while (!main_queue.empty()) {
278         temp = to_string(main_queue.top().p_no).length();
279         cout << '|' << string(6 - temp / 2 - temp % 2, ' ')
280             << main_queue.top().p_no << string(7 - temp / 2, ' ');
281         temp = to_string(main_queue.top().start_AT).length();
282         cout << '|' << string(7 - temp / 2 - temp % 2, ' ')
283             << main_queue.top().start_AT << string(7 - temp / 2, ' ');
284         temp = to_string(main_queue.top().BT).length();
285         cout << '|' << string(6 - temp / 2 - temp % 2, ' ')
286             << main_queue.top().BT << string(6 - temp / 2, ' ');
287         temp = to_string(main_queue.top().CT).length();
288         cout << '|' << string(8 - temp / 2 - temp % 2, ' ')
289             << main_queue.top().CT << string(9 - temp / 2, ' ');
290         temp = to_string(main_queue.top().TAT).length();
291         cout << '|' << string(8 - temp / 2 - temp % 2, ' ')
292             << main_queue.top().TAT << string(9 - temp / 2, ' ');
293         temp = to_string(main_queue.top().WT).length();
294         cout << '|' << string(7 - temp / 2 - temp % 2, ' ')
295             << main_queue.top().WT << string(7 - temp / 2, ' ');
296         temp = to_string(main_queue.top().RT).length();
297         cout << '|' << string(7 - temp / 2 - temp % 2, ' ')
298             << main_queue.top().RT << string(8 - temp / 2, ' ');
299         if (high == true) {
300             temp = to_string(main_queue.top().priority).length();
301             cout << '|' << string(5 - temp / 2 - temp % 2, ' ')
302                 << main_queue.top().priority << string(5 - temp / 2, ' ');
303         }
304         cout << "|\n";
```

```
305         main_queue.pop();
306     }
307     cout << "+------------+-------------";
308     cout << "+------------+-----------------";
309     cout << "+-----------------+--------------+---------------+";
310     if (high == true)
311         cout << "----------+";
312     cout << endl;
313     temp1 = get_total_CT(tempq);
314     cout << "\nTotal completion time :- " << temp1 << endl;
315     cout << "Average completion time :- " << temp1 / size << endl;
316     temp1 = get_total_TAT(tempq);
317     cout << "\nTotal turnaround time :- " << temp1 << endl;
318     cout << "Average turnaround time :- " << temp1 / size << endl;
319     temp1 = get_total_WT(tempq);
320     cout << "\nTotal waiting time :- " << temp1 << endl;
321     cout << "Average waiting time :- " << temp1 / size << endl;
322     temp1 = get_total_RT(tempq);
323     cout << "\nTotal response time :- " << temp1 << endl;
324     cout << "Average response time :- " << temp1 / size << endl;
325 }
326 
327 void disp_gantt_chart(queue<process> gantt)
328 {
329     int temp, prev = 0;
330     queue<process> spaces = gantt;
331     cout << "\n\nGantt Chart (IS indicates ideal state) :- \n\n+";
332     while (!spaces.empty()) {
333         cout << string(to_string(spaces.front().p_no).length() +
(spaces.front().p_no != -1) + 2 * spaces.front().temp_BT, '-') << "+";
334         spaces.pop();
335     }
336     cout << "\n|";
337     spaces = gantt;
338     while (!spaces.empty()) {
339         cout << string(spaces.front().temp_BT, ' ');
340         if (spaces.front().p_no == -1)
341             cout << "IS" << string(spaces.front().temp_BT, ' ') << '|';
342         else
343             cout << "P" << spaces.front().p_no
344                 << string(spaces.front().temp_BT, ' ') << '|';
345         spaces.pop();
346     }
347     spaces = gantt;
348     cout << "\n+";
349     while (!spaces.empty()) {
350         cout << string(to_string(spaces.front().p_no).length() +
(spaces.front().p_no != -1) + 2 * spaces.front().temp_BT, '-')
351             << "+";
352         spaces.pop();
353     }
354     spaces = gantt;
355     cout << "\n0";
356     while (!spaces.empty()) {
357         temp = to_string(spaces.front().CT).length();
358         cout << string(to_string(spaces.front().p_no).length() +
(spaces.front().p_no != -1) + 2 * spaces.front().temp_BT - temp / 2 - prev, ' ')
359             << spaces.front().CT;
360         prev = temp / 2 - temp % 2 == 0;
361         spaces.pop();
362     }
363     cout << "\n\n";
```

```
364 }
365
366 int main()
367 {
368     priority_queue<process> ready_queue, completion_queue;
369
370     queue<process> gantt;
371     ready_queue = set_sample_data();
372
373     completion_queue = Priority_NP_run(ready_queue, &gantt, true);
374
375     disp(completion_queue, true);
376
377     disp_gantt_chart(gantt);
378     return 0;
379 }
```

```
Enter the number of processes: 7

Enter arrival time ,burst time and priority of process 1 : 0 4 2

Enter arrival time ,burst time and priority of process 2 : 1 2 4

Enter arrival time ,burst time and priority of process 3 : 2 3 6

Enter arrival time ,burst time and priority of process 4 : 3 5 10

Enter arrival time ,burst time and priority of process 5 : 4 1 8

Enter arrival time ,burst time and priority of process 6 : 5 4 12

Enter arrival time ,burst time and priority of process 7 : 6 6 9
```

Process No.	Arrival Time	Burst Time	Completion Time	Turnaround Time	Waiting Time	Response Time	Priority
1	0	4	4	4	0	0	2
2	1	2	25	24	22	22	4
3	2	3	23	21	18	18	6
4	3	5	9	6	1	1	10
5	4	1	20	16	15	15	8
6	5	4	13	8	4	4	12
7	6	6	19	13	7	7	9

```
Total completion time :- 113
Average completion time :- 16.1429

Total turnaround time :- 92
Average turnaround time :- 13.1429

Total waiting time :- 67
Average waiting time :- 9.57143

Total response time :- 67
Average response time :- 9.57143


Gantt Chart (IS indicates ideal state) :-

+----------+------------+----------+--------------+----+--------+------+
|    P1    |     P4     |    P6    |      P7      | P5 |   P3   |  P2  |
+----------+------------+----------+--------------+----+--------+------+
0          4            9         13             19   20       23     25
```