```
/* 1) Consider the following jobs submitted to a system:
        Process  Arrival time(ms)  CPU burst time(ms)   Priority
        Print              0              7          3
        email              2              3          2
        File transfer      2              8          1
        Web service        3              4          4
     b) Schedule the above processes according to the associated priority where a low value
        indicates higher priority. The algorithm should preempt the current process if a
        higher priority process arrives. */

// Priority scheduling is a non-preemptive algorithm - each process is assigned a priority
// Process with the highest priority is to be executed first and so on

// Processes with the same priority are executed on first come first served basis

// Priority can be given or decided based on memory requirements, time requirements or
// resource requirement

// But problem to be solved is Priority scheduling AND its Premptive wrt priority
//   preempt the current process if a higher priority process arrives
//   where priority are given and a low value indicates higher priority

//   Hence the implementation should be premptive priority CPU scheduling

// What is the run time, and how can the code be improved?

#include <stdio.h>
#include <limits.h>

   int n = 4 , remain = n ; // n is number of and remain is remaining process
   int arrivalTime[10]   = { 0, 2, 2, 3 }; // arrival time, array is zero indexed
   int burstTime[10]     = { 7, 3, 8, 4 }; // burst time
   int remainingTime[10] = { 7, 3, 8, 4 }; // reamining time
   int priority[10]      = { 3, 2, 1, 4 }; // priority
   // read details from user instead

// return index of process in waiting queue with highest priority , -1 otherwise
int processWithHighestPriority( int elapsedTime )
 {
   int processPriority = INT_MAX;
   int process = -1 ;

   for( int i=0; i<n; i++ )
    {// process not completed and arrival time less than elapsed time and find higher priority
      if (remainingTime[i] > 0 && arrivalTime[i] <= elapsedTime
           &&  priority[i] < processPriority )
       { // assign new found priority and remember index/process
         processPriority = priority[i];
         process = i;
       }
    }
   return process ;
 }
```

```
int main()
 {
  int processNo, elapsedTime;
  int totalWaitTime = 0, totalTurnAroundTime = 0;

  printf("\n Process | Turnaround time | Waiting time\n");

  for( elapsedTime=0; remain!=0;) // For each unit of time
   { // find process in waiting queue with highest priority
    processNo = processWithHighestPriority(elapsedTime);

    elapsedTime ++; // add one unit of time to elapsed time , why not in for loop ?

    if( processNo == -1 )
     { // No process found in waiting queue
     }
    else
     {
      remainingTime[processNo]--; // remaining time - one unit of time i.e. remaining time - 1

      if( remainingTime[processNo]==0 ) // If any process has completed, then decrement
       { //  remaining process count and print Process | Turnaround time | Waiting time
        remain--; // Decrement remaining processes count
        printf("  p[%d]\t |\t %d\t   |\t%d\n", processNo,
                elapsedTime - arrivalTime[processNo],
                elapsedTime - arrivalTime[processNo] - burstTime[processNo] );
        // Update total waitingTime and turnAroundTime of the completed processes so far
        totalTurnAroundTime += elapsedTime - arrivalTime[processNo];
        totalWaitTime += elapsedTime - arrivalTime[processNo] - burstTime[processNo];
       }
     }
   }

  printf("\n Average turnaround time = %f\n", totalTurnAroundTime * 1.0 / n);
  printf("\n Average waiting time    = %f", totalWaitTime * 1.0 / n);

  return 0;
 }
/* Output :
 Process | Turnaround time | Waiting time
  p[3]       |      8       |    0
  p[2]       |      11      |    8
  p[1]       |      18      |    11
  p[4]       |      19      |    15

 Average waiting time    = 8.500000
 Average turnaround time = 14.000000 */

/* A major problem with priority scheduling is indefinite blocking or starvation

   A solution to the problem of indefinite blockage of the low-priority process is aging

   Aging is a technique of gradually increasing the priority of processes that wait in the
   system for a long period of time */
```