DESIGN

```
//structure

struct node
{
   int id ;
   int priority;
   struct pointer next;
}

struct pointer create_node( int JID , int priority)
{
   struct pointer head = malloc()
   head -> id = JID;
   head -> priority = priority;
   head -> next = nullptr;
   return head ;
}

add ( head , id , priority)
{
   if head == NULL
   {
      *head = create_node(JID , priority);
      return;
   }

   struct pointer new_node  = create_node(JID , priority);

   // find appropriate across differnt priority using temp and temp_prev
   while(temp && temp -> pri > new_node -> pri )
   {
      temp_prev = temp;
      temp = temp -> next;
   }

   //move across same priority
   while(temp && temp -> jid < new_node -> jid && temp -> pri == new_node -> pri )
   {
      temp_prev = temp;
      temp = temp -> next;
   }
```

```
    // if at front
    if(temp_prev == NULL && temp == *head)
    {
        new_node -> next = *head ;
        *head = new_node;
        return;
    }

    // at end
    if(!temp)
    {
        temp_prev -> next = new_node;
        return;
    }

    // at middle
        temp_prev -> next = new_node;
        new_node -> next = temp;
        return;
}

display(head)
{
    if head == nullptr printt "-1";
    else
        {
            temp = head;
            while(temp)
            {
                print(temp -> id , temp -> priority);
            }
        }
}

shedule(head)
{

    if(!head)
        '-1'
    else
     {
       print(head -> id)
       // remove first element
       temp = head ;
```

```
      head  = head - > next ;
      free(temp)
    }
    return
}

next_job(head)
{


   if(!head)  "-1\n"

   else "head -> id";

   return;
}

replace_priority(head,id, new_priority)
{
   // find that node

   temp = head;
   temp_prev = nullptr;
   while(temp -> id != id)
   {
      temp_prev = temp;
      temp = temp -> next;
   }
   if(!temp) return -1;

   // at front
   if( temp_prev == null)
      temp_prev = head ;
      head = head - > next;
      free(temp_prev);


   // at end
   else if(!temp -> next)
   {
      temp_prev -> next = nullptr;
      free(temp)

   }
```

```
    // at middle
    else
    {
        temp_prev -> next = temp -> next;
        free(temp);
    }

    // add new node

    {
        add(jid , priority);
        return;
    }


}

int main()
{
    char c;
    int JID , int priority;

    switch(c)

        if c == 'a'
            scan( jid , priority)
            call add(wait_queue , jid  , priority);
        if c == 'd'
            call display(wait_queue);
        if c == 'r'
            scan( jid , new_priority);
            call replace_priority(wait_queue , jid  , new_priority);
        if c == 'e'
            terminate
        if c == 'n'
            call next_job(wait_queue);
        if c == 's'
            call shedule(wait_queue);
}
```