

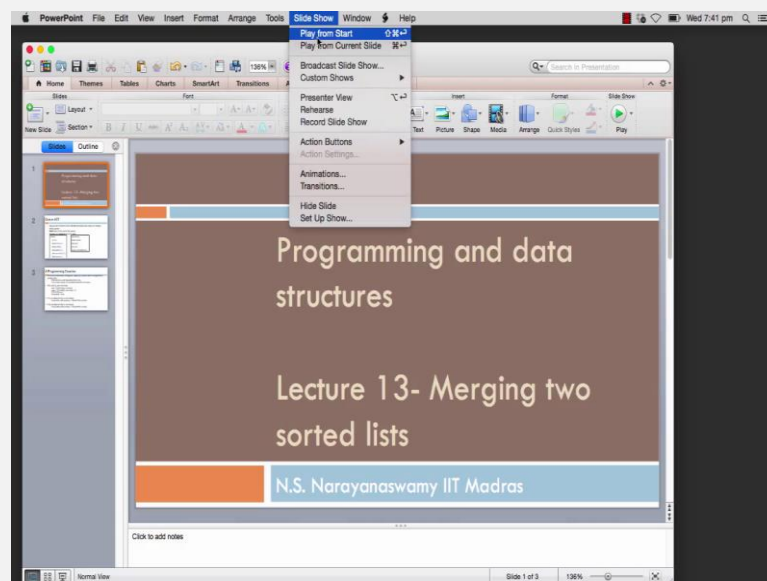
**Programming and Data Structures**  
**Prof. N.S. Narayanaswamy**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Madras**

**Lecture - 13**  
**Merging using Queue ADT and Queue types**

In the second lecture on the queue abstract data type, let us take a look at the programming exercise in which we are going to use the queue implementation that we finished in the previous lecture. So, recall that in the previous lecture, we looked at the queue of some data type, we looked at the methods, we came up the implementation, we compiled it, we made an object file and we made the appropriate interface files.

In today's lecture, we are going to take a look at this implementation and we are going to use it in an interesting programming exercise. Let us go to the description of the programming exercise, and then let us go to the program and see the implementation and the execution of that particular program. So, again this is likely to be just a short lecture approximately 20 or 25 minutes long and most of it will be an analysis of the code that we have written.

(Refer Slide Time: 00:58)



So, the exercise is to merge two sorted list.

(Refer Slide Time: 01:06)

## Queue ADT

- Queues are a First in First Out data structure-Like a line in a railway ticket counter
- **Add** items to the end of the queue
- **Access** and **remove** from the front

Interface

```
int size();  
boolean isEmpty();  
Boolean isFull();  
Object peekfront();  
void enqueue(Object o);  
Object dequeue();
```

Queue Array;

Integer capacity;

Index front;

Index rear;

Integer numberOfElements;

So, this was the queue abstract data type, now let us go directly to our problem.

(Refer Slide Time: 01:11)

## A Programming Exercise

- Given two sorted lists of integers, output  $Q_3$  a queue with a merged list in sorted order.  
First list is in  $Q_1$  and the second list is in  $Q_2$   
Front of each queue is the smallest element in the queue
- If  $Q_1$  and  $Q_2$  are not empty  
 $next = \min(front(Q_1), front(Q_2));$   
 $index = 1$  if  $front(Q_1)$ , else  $index = 2;$   
 $deQueue(Q_{index})$   
 $Enqueue(Q_3, next);$
- If  $Q_1$  is empty and  $Q_2$  is non-empty  
 $Enqueue(Q_3, deQueue(Q_2))$  – Repeat till  $Q_2$  is empty
- If  $Q_2$  is empty and  $Q_1$  is non-empty  
 $Enqueue(Q_3, deQueue(Q_1))$  – Repeat till  $Q_1$  is empty

So, the programming exercise the following you have given two sorted list of integers

and you have to output Q 3, a queue with a merged list in sorted order. So, you can imagine that this is standard exercise that you might find in physical education classes in a school, you have two rows of students were arranged in height order. And the physical education teacher ensures that the two rows are merged into a single row with all the students in the correct height order in increasing order of height. This is something that most of you has gone to school would have seen as a simple exercise. So, that is exactly our programming exercise.

So, the first list is in queue Q in Q 1, it is a sorted list of integers, you can think of it as in ascending order and the second one is again a sorted list, you can think of this is in ascending order. So, now what we should do is, let us change it, because I have written min here, the queue Q 1 is in descending order and Q 2 is in descending order and the resulting queue should also be brought into descending order.

Now, let us just look at the Pseudo code for this in terms of the functions that we have written. If the two queues are not empty, then you pick the front of the Q 1, front of Q 2, this is the peak front functions take the smallest of them, take them as next. Now, keep index is equal to 1 the front of, if next is front of Q 1, else index is 2. So, now what do we do, we dequeue the next element from Q 1 or Q 2 appropriately and enqueue it into Q 3. That is we removed this step, only finds the minimum by robbing into the, or peaking into the two queues. This sentence keeps track of which queue you peak into and this one dequeues from that Q and this one inserts that element into Q 3.

Now, if Q 1 is empty and Q 2 is non empty, then what you do is, you just enqueue by dequeue from Q 2 repeatedly till Q 2 becomes empty, you enqueue to Q 3. And if Q 2 is empty and Q 1 is non empty, you repeatedly do the same thing by dequeuing from Q 1 and entering it into Q 3 and repeated till Q 1 is empty. So, this was really the, ((Refer Time: 03:57)) this is the programming exercise and we are going to as you can see from the Pseudo code itself, it is a crisp piece of code, where we essentially use the functions, we have designed in the interface. In the Q implementation, whatever methods we are implemented are exactly what is used expect for a min function kind of a thing that I have listed here.

(Refer Slide Time: 04:28)

```
bash
DLList-interface.h ListMethods.o merge stack-array.o week1-prog2.c
DLList.h PerFDLL merge.c stack-interface.h week1-prog3
DLListMethods.c PerFDLL.c queue-array.c stack-list.c week1-prog3.c
NSNarayanawamys-MacBook-Air:code narayanawamys$ vi queue-array.c
NSNarayanawamys-MacBook-Air:code narayanawamys$ vi queue-interface.h
NSNarayanawamys-MacBook-Air:code narayanawamys$ vi queue.h
NSNarayanawamys-MacBook-Air:code narayanawamys$ vi merge.c
NSNarayanawamys-MacBook-Air:code narayanawamys$ ls
BigAddSub DLListMethods.o balanced queue-array.o stack-list.o
BigAddSub.c List-interface.h balanced-parenthesis.c queue-interface.h stack.h
BinSchComp List.h in-to-post queue.h week1-prog1.c
BinSchComp.c ListMethods.c in-to-postfix.c stack-array.c week1-prog2
DLList-interface.h ListMethods.o merge stack-array.o week1-prog2.c
DLList.h PerFDLL merge.c stack-interface.h week1-prog3
DLListMethods.c PerFDLL.c queue-array.c stack-list.c week1-prog3.c
NSNarayanawamys-MacBook-Air:code narayanawamys$ cd ..
NSNarayanawamys-MacBook-Air:POS-mooc narayanawamys$ ls
code docs pres recs vids
NSNarayanawamys-MacBook-Air:POS-mooc narayanawamys$ cd code
NSNarayanawamys-MacBook-Air:code narayanawamys$ ls
BigAddSub List.h merge stack-list.c
BigAddSub.c ListMethods.c merge.c stack-list.o
BinSchComp ListMethods.o queue-array.c stack.h
BinSchComp.c PerFDLL queue-array.o week1-prog1.c
DLList-interface.h PerFDLL.c queue-interface.h week1-prog2
DLList.h balanced queue.h week1-prog2.c
DLListMethods.c balanced-parenthesis.c stack-array.c week1-prog3
DLListMethods.o in-to-post stack-array.o week1-prog3.c
List-interface.h in-to-postfix.c stack-interface.h
NSNarayanawamys-MacBook-Air:code narayanawamys$ vi queue.h
NSNarayanawamys-MacBook-Air:code narayanawamys$ vi queue-interface.h
NSNarayanawamys-MacBook-Air:code narayanawamys$ vi queue-array.c
NSNarayanawamys-MacBook-Air:code narayanawamys$ gcc -c queue-array.c
NSNarayanawamys-MacBook-Air:code narayanawamys$ ls -ld queue-array.o
-rw-r--r-- 1 narayanawamys staff 2488 Feb 11 19:34 queue-array.o
NSNarayanawamys-MacBook-Air:code narayanawamys$ vi merge.c
```

Now, let us go to our program, this program is in merge dot c.

(Refer Slide Time: 04:30)

```
vim
#include <stdio.h>
#include "queue-interface.h"

/* method to merge two sorted queues */
Queue * merge(Queue * q1, Queue * q2)
{
    Queue * q3 = createQueue(q1->capacity + q2->capacity);
    // create a new queue of required capacity

    while(!isEmpty(q1) && !isEmpty(q2))
    // while both queues are not empty
    {
        if(peekFront(q1) < peekFront(q2))
        // compare the front elements of both queues
        {
            enqueue(q3, dequeue(q1));
        }
        // enqueue the smallest of both elements into the result queue
        else
        {
            enqueue(q3, dequeue(q2));
        }
    }
    // put all the remaining elements in any of the queue
    while(!isEmpty(q1))
    {
        enqueue(q3, dequeue(q1));
    }
    // into the remaining result queue
    while(!isEmpty(q2))
    {
        enqueue(q3, dequeue(q2));
    }
    return q3;
}

// return the result queue
void printQueue(Queue *q)
{
    while(!isEmpty(q))
    {
        printf("%d ", dequeue(q)); // while queue is not empty
    }
    printf("\n"); // dequeue and print each element;
}

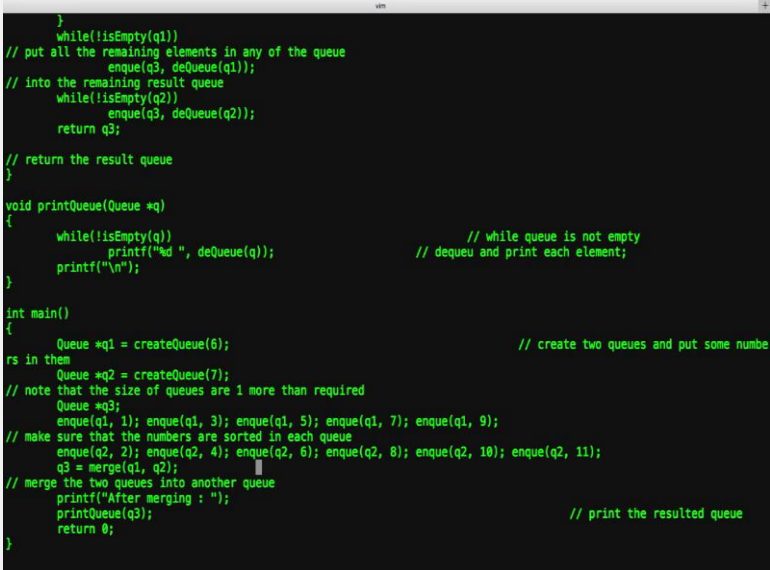
"merge.c" 51L, 1551C
```

I include interface dot h, let us look at the first method, it merges two sorted queues, merge q 1 and q 2. What does it do? First, it creates a queue called q 3, whose capacity is q 1 capacity plus q 2 capacity. Then, what does it do, while the two queues are non

empty, if the element in the head of q 1 is smaller than the element in the head of q 2, you enqueue into q 3, the element at the head of q 1.

Otherwise, you enqueue into q 3, the element at the head of q 2 and remove it from q 1 and q 2 respectively. Otherwise, while q 1 is not empty, you put all the remaining elements into q 1. Otherwise, you put all the elements of q 2 into the queue q 3 and return q 3; this is exactly the pseudo code that we had seen. Then, we have a print function, the print function is also interesting, because it is a use of the dequeue function. So, it takes q as an argument and while the q is not empty, it prints the elements of the q 1 after the other.

(Refer Slide Time: 05:52)



```
    }
    while(!isEmpty(q1))
    // put all the remaining elements in any of the queue
        enqueue(q3, dequeue(q1));
    // into the remaining result queue
    while(!isEmpty(q2))
        enqueue(q3, dequeue(q2));
    return q3;

// return the result queue
}

void printQueue(Queue *q)
{
    while(!isEmpty(q))
        printf("%d ", dequeue(q));           // while queue is not empty
    printf("\n");                             // dequeue and print each element;
}

int main()
{
    Queue *q1 = createQueue(6);                // create two queues and put some numbe
rs in them
    Queue *q2 = createQueue(7);
    // note that the size of queues are 1 more than required
    Queue *q3;
    enqueue(q1, 1); enqueue(q1, 3); enqueue(q1, 5); enqueue(q1, 7); enqueue(q1, 9);
    // make sure that the numbers are sorted in each queue
    enqueue(q2, 2); enqueue(q2, 4); enqueue(q2, 6); enqueue(q2, 8); enqueue(q2, 10); enqueue(q2, 11);
    q3 = merge(q1, q2);
    // merge the two queues into another queue
    printf("After merging : ");
    printQueue(q3);                             // print the resulted queue
    return 0;
}
```

Now, let us just looked at the main function, main function is written in such a way, so that you can get a flavor of the utilization of all the different methods that we have implemented. The first statement is to create q 1 with 6 elements. Now, observe that this 6 is not very important, you could have actually put a scanf here, let the value of the queue and then created this queue, which is perfectly fine, but that is not what I have done, because I just want to show you the utilization of this queue.

Now, you have q 2, now you create q with 7 elements and because we are doing a

circular queue implementation in our code, we have used one more location than what is required. Now, we have a pointer for q 3. So, now you do the following sequence of an enqueues, you enqueue 1 into q 1, then 3 into q 1, 5 into q 1, 7 into q 1 and 9 into q 1. This just done, just to make illustrate the concept that you can essentially enqueue a sequence of elements and you can even export this by interacting with the user and putting the elements into a loop.

Then, you enqueue into q 2, the elements 2, 4, 6, 8, 10 and 11, so observe that in 1 queue, we have only 6 elements, in the 2nd queue, we have only 5 elements, that is in q 1, we have 5 elements and in q 2, we have only 6 elements. Now, we merge the two queues and we print out q 3, so of course, there are other methods and I am not showing you, I am not checking, whether the queue is full and so on and so forth, but is empty is being check. Of course, here I should have check, if the queue is full or not, but since it is just TA example, I am not check if the queue is full.

(Refer Slide Time: 08:06)

```

bash
DLLList.h      PerFDLL      merge.c      stack-interface.h  week1-prog3
DLLListMethods.c  PerFDLL.c    queue-array.c  stack-list.c      week1-prog3.c
NSNarayanawamys-MacBook-Air:code narayanawamys$ vi queue-array.c
NSNarayanawamys-MacBook-Air:code narayanawamys$ vi queue-interface.h
NSNarayanawamys-MacBook-Air:code narayanawamys$ vi queue.h
NSNarayanawamys-MacBook-Air:code narayanawamys$ vi merge.c
NSNarayanawamys-MacBook-Air:code narayanawamys$ ls
BigAddSub      DLLListMethods.o  balanced      queue-array.o      stack-list.o
BigAddSub.c    List-Interface.h  balanced-paranthesis.c  queue-interface.h  stack.h
BinSchComp    List.h            in-to-post    queue.h            week1-prog1.c
BinSchComp.c  ListMethods.c     in-to-postfix.c  stack-array.c      week1-prog2.c
DLLList-Interface.h  ListMethods.o    merge.c        stack-array.o      week1-prog2.c
DLLList.h     PerFDLL           merge.c        stack-interface.h  week1-prog3
DLLListMethods.c  PerFDLL.c        queue-array.c  stack-list.c      week1-prog3.c
NSNarayanawamys-MacBook-Air:code narayanawamys$ cd ..
NSNarayanawamys-MacBook-Air:PDS-mooc narayanawamys$ ls
code  docs  pres  recs  vids
NSNarayanawamys-MacBook-Air:PDS-mooc narayanawamys$ cd code
NSNarayanawamys-MacBook-Air:code narayanawamys$ ls
BigAddSub      List.h            merge          stack-list.c
BigAddSub.c    ListMethods.c     merge.c        stack-list.o
BinSchComp    ListMethods.o     queue-array.c  stack.h
BinSchComp.c  PerFDLL           queue-array.o  week1-prog1.c
DLLList-Interface.h  PerFDLL.c        queue-interface.h  week1-prog2.c
DLLList.h     balanced          queue.h        week1-prog2.c
DLLListMethods.c  balanced-paranthesis.c  stack-array.c  week1-prog3
DLLListMethods.o  in-to-post        stack-array.o  week1-prog3.c
List-Interface.h  in-to-postfix.c   stack-interface.h
NSNarayanawamys-MacBook-Air:code narayanawamys$ vi queue.h
NSNarayanawamys-MacBook-Air:code narayanawamys$ vi queue-interface.h
NSNarayanawamys-MacBook-Air:code narayanawamys$ vi queue-array.c
NSNarayanawamys-MacBook-Air:code narayanawamys$ gcc -c queue-array.c
NSNarayanawamys-MacBook-Air:code narayanawamys$ ls -ld queue-array.o
-rw-r--r--  1 narayanawamys  staff  2488 Feb 11 19:34 queue-array.o
NSNarayanawamys-MacBook-Air:code narayanawamys$ vi merge.c
NSNarayanawamys-MacBook-Air:code narayanawamys$

```

That is a TA merge program to merge two fixed queues just to show you the example, but it is very easily modifiable by interacting with the user, who runs the program. So, let us compile merge dot c and you will recall that one has to include into this compilation or link it with the object file that we have, otherwise, a compilation will fail.

(Refer Slide Time: 08:39)

```
NSNarayanawamys-MacBook-Air:code narayanawamys$ ls
BigAddSub      List.h          merge           stack-list.c
BigAddSub.c    ListMethods.c  merge.c         stack-list.o
BinSchComp     ListMethods.o  queue-array.c   stack.h
BinSchComp.c   PerfDLL.c      queue-array.o   week1-prog1.c
DLLList-Interface.h  PerfDLL.c      queue-interface.h week1-prog2.c
DLLList.h       balanced       queue.h         week1-prog2.c
DLLListMethods.c balanced-paranthesis.c stack-array.c   week1-prog3.c
DLLListMethods.o in-to-post     stack-array.o   week1-prog3.c
List-Interface.h in-to-postfix.c stack-interface.h
NSNarayanawamys-MacBook-Air:code narayanawamys$ vi queue.h
NSNarayanawamys-MacBook-Air:code narayanawamys$ vi queue-interface.h
NSNarayanawamys-MacBook-Air:code narayanawamys$ vi queue-array.c
NSNarayanawamys-MacBook-Air:code narayanawamys$ gcc -c queue-array.c
NSNarayanawamys-MacBook-Air:code narayanawamys$ ls -ld queue-array.o
-rw-r--r--  1 narayanawamys  staff  2488 Feb 11 19:34 queue-array.o
NSNarayanawamys-MacBook-Air:code narayanawamys$ vi merge.c
NSNarayanawamys-MacBook-Air:code narayanawamys$ gcc merge.c -o merge
Undefined symbols for architecture x86_64:
  "_createQueue", referenced from:
    _merge in merge-e0e948.o
    _main in merge-e0e948.o
  "_deQueue", referenced from:
    _merge in merge-e0e948.o
    _printQueue in merge-e0e948.o
  "_enqueue", referenced from:
    _merge in merge-e0e948.o
    _main in merge-e0e948.o
  "_isEmpty", referenced from:
    _merge in merge-e0e948.o
    _printQueue in merge-e0e948.o
  "_peekFront", referenced from:
    _merge in merge-e0e948.o
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see invocation)
NSNarayanawamys-MacBook-Air:code narayanawamys$
```

Now, let us see, what happens if I leave out that? As you can see, what it says it compiles, but then it says that it cannot understand the symbols create queue, it cannot understand dequeue, it cannot understand enqueue, it cannot understand isEmpty, it cannot understand peek front, etcetera. In the earlier lectures, I did not show this to you, but now I deliberately made an error by not including the object file or linking the object file during the compilation and you can see that the compilation fails to recognize these particular functions.

Though, the compilation did not have any errors, it says that you must read the message very carefully, it says that in the function merge create queue was not understood, in the function merge dequeue was not understood and so on and so forth, also in main. So, look at this error, it says linker command failed with exit code 1. So, it says that it was unable to link these particular functions which are available in the header file.

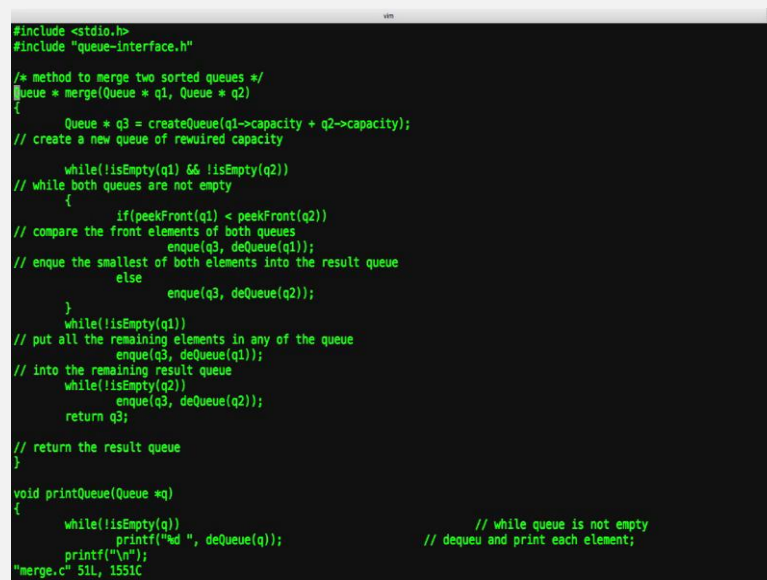
Now, let us solve the problem by linking queue array dot o, now the compilation has succeeded. Now, I have the merge program as you can see the merge program has merged the two arrays into a single array in ascending order, so this really treated as a queue. So, as you can see all the queue methods were used to merge the two queues, which were there. Recall that 1 queue had 1, 3, 5, 7, 9; the second queue had 2, 4, 6, 8,



10 and 11 and observe that both the queues have been merged into ascending order.

So, this really is a very simple, but many of you will not know this merging procedure would have seen it as a very important thing in sorting algorithms like merge sort, which are recursive sorting algorithms. So, what we have done is, we have implemented a very quick merge using the queue abstract data type. So, as you can see the code is very crisp and very easy to build from the different individual functions, which we have designed and implemented with the queue abstract data type.

(Refer Slide Time: 11:28)

A screenshot of a code editor showing a C program for merging two sorted queues. The code is written in a dark-themed editor with a light blue title bar. The code includes standard headers, a merge function that uses a while loop to compare and enqueue elements from two input queues into a new result queue, and a printQueue function that uses a while loop to dequeue and print elements. The file name 'merge.c' is visible at the bottom left.

```
#include <stdio.h>
#include "queue-interface.h"

/* method to merge two sorted queues */
Queue * merge(Queue * q1, Queue * q2)
{
    Queue * q3 = createQueue(q1->capacity + q2->capacity);
    // create a new queue of required capacity

    while(!isEmpty(q1) && !isEmpty(q2))
    // while both queues are not empty
    {
        if(peekFront(q1) < peekFront(q2))
        // compare the front elements of both queues
            enqueue(q3, dequeue(q1));
        // enqueue the smallest of both elements into the result queue
        else
            enqueue(q3, dequeue(q2));
    }
    while(!isEmpty(q1))
    // put all the remaining elements in any of the queue
        enqueue(q3, dequeue(q1));
    // into the remaining result queue
    while(!isEmpty(q2))
        enqueue(q3, dequeue(q2));
    return q3;
}

// return the result queue
}

void printQueue(Queue *q)
{
    while(!isEmpty(q))
    {
        printf("%d ", dequeue(q));           // while queue is not empty
        printf("\n");                         // dequeue and print each element;
    }
}

"merge.c" 51L, 1551C
```

To the extent that the merging program is almost exactly what you see in the presentation, there is absolutely nothing new except for this create queue function and then we already saw this loop which was there and the most important thing is that this way of programming ensures that if you make a good presentation and if you understand your functions clearly, then writing a program is extremely simple.

Therefore, whenever you use an abstract data type, it is very important for you to define the data type completely and then use it in your implementations, this is a very important message. This completes the utilization of the queue abstract data type in a very important programming exercise, you will get a similar programming exercise for you to



program in this week itself, it will be fairly challenging and we will come to it by tomorrow.

In the next lecture, what we will do is, we will look at the different types of queues. This is a very fundamental difference between stacks and queues, there is only a single type of stack, but there are multiple types of queues that you would want to maintain and we will see what kind of queues are there and why they are important. And let us just continue on to that particular presentation itself ((Refer Time: 12:49)), so let us just looked at the different types of queues which are there.

(Refer Slide Time: 13:24)

### Queue ADT - Recall

- Queues are a First in First Out data structure-Like a line in a railway ticket counter
- **Add** items to the end of the queue
- **Access** and **remove** from the front

Interface	Queue Array;
int size();	Integer capacity;
boolean isEmpty();	Index front;
Boolean isFull();	Index rear;
Object peekfront();	Integer numberOfElements;
void enqueue(Object o);	
Object dequeue();	

So, this is the queue abstract data type, I do not have to go back and say point out, what are all the different data items which are maintained in the data type and what are all the different methods, we have seen it. There is enqueue, dequeue, there is peak front, there isEmpty, there isFull, there is size, these are all the methods and the data items are the array that contains the q and so on and so forth.

(Refer Slide Time: 13:46)

## Types of Queues

- Circular Queue - Ring buffer.

Circular **array implementation of a Queue**

Enqueue – increment rear by 1 modulo N; increment size.

Dequeue – increment front by 1 modulo N; decrement size.

isEmpty – if rear == front.

isFull – if rear and front differ by 1.

Better utilization of available array implementation

- Double Ended Queue – DeQueue

Insert and Delete from either end.

What are the different types of queues? As you saw, the implementation of the queue was actually of a circular queue. Now, it is important for you not to get confuse that a circular queue is a special type of a queue, a circular queue is actually a circular array implementation of the queue abstract data type. I repeat, the circular queue is nothing else, but a different implementation of the queue data type. What do I mean by this? The circular queue does not provide the user any new specific features, because of it is being a circular queue. It only provides you an efficient utilization of the arrays space to maintain the queue.

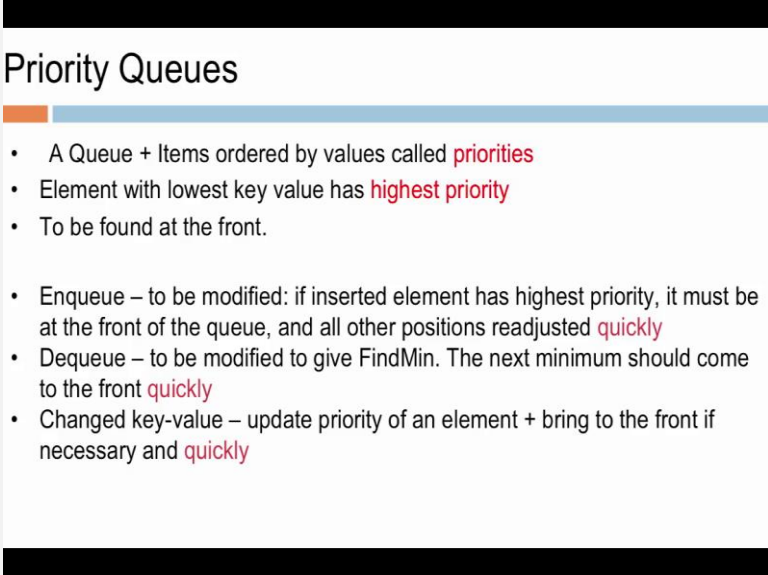
And the enqueue operation incremented the rear by 1 modulo N and the dequeue operation incremented front by 1 modulo N. The queue isEmpty if rear is equal to front and it was full if rear and front differ by 1 and it is a better utilization of the available array implementation, it is a better utilization of the available array in this particular implementation. So, therefore we have a special kind of an implementation of queue data type which is the circular queue, sometimes it is also called the ring buffer.

There is another queue which is called the double ended queue, we will come to this queue a little while later. A double ended queue is otherwise called a dequeue or a deq, you insert and delete from either end. The importance of this queue will become clear,

when you think of the following question that in normal queues, people enter the queue and decide to actually exit from the queue, at which point of time which is lead the queue from wherever they were.

Therefore, while we should be able to insert into the rear end of the queue, we should also be able to allow people to exit from the rear end of the queue. So, this is what a double ended queue is, where we would like to insert and delete elements from either end of the queue.

(Refer Slide Time: 15:56)



**Priority Queues**

- A Queue + Items ordered by values called **priorities**
- Element with lowest key value has **highest priority**
- To be found at the front.
- Enqueue – to be modified: if inserted element has highest priority, it must be at the front of the queue, and all other positions readjusted **quickly**
- Dequeue – to be modified to give FindMin. The next minimum should come to the front **quickly**
- Changed key-value – update priority of an element + bring to the front if necessary and **quickly**

Now, the very important queue is, what is called a priority queue and priority queues as a name suggest is not just a queue in which the elements are organize in a first in first out fashion, every element has a certain priority. So, clearly I mean, if a certain a very important person arise on a road, then such a person gets a higher priority to use a road and everybody else is arranged. To stand and weight for this higher priority person to go ahead in the queue, but while this person goes ahead in the queue in the remaining people with the same priority have the same relative positions in the queue.

So, this kind of a feature is associated by the is definitely provided by the priority queue, that is, it is a queue first of all, which means elements are enqueued and dequeued, but

items can be ordered by certain additional values called priorities. Element with a lowest key value has the highest priority and this one is always to be at the front, that is the element of the highest priority will always be at the front of the queue.

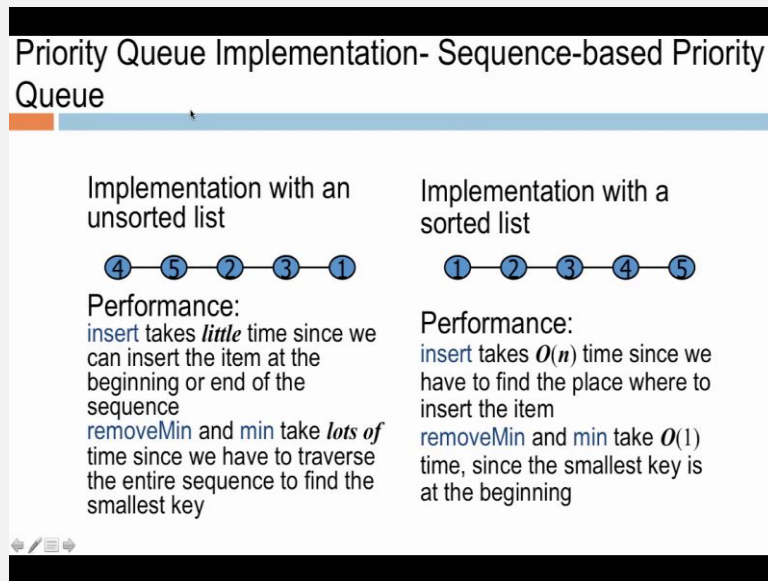
Therefore, what are all the things that we must do, enqueue function has to be modified. If you insert an element into the queue that is at the rear of queue and it has the highest priority, it must somehow be ensured, that it comes to the front of the queue and all other positions must be readjusted, very importantly it must be readjusted quickly. Similarly, if you want to dequeue, that is, we want to remove an element from the queue, then you must modified, so that you get the element of the least value.

In other words, the element of the higher priority and then the next element must be organize, so that the next minimum can be obtain very quickly, so this is the dequeue operation. So, you can again imagine that if somebody has a higher priority, such a person must be brought out of the queue first and the next person or the next data item, which as the best priority or the highest priority must be arranged. So, that it can be remove quickly, when the time arises for that element to be removed.

The other operations are that you might want to update the priority of an element based on certain things. Those of few, who are studying operating systems or who have some exposure to operating systems will recall that priority queues play a very important role and changing the priority of a element is a very important operation done by the operating system.

So, the two ways of implementing the priority queues, so let us just look at the methods, there are enqueue and dequeue to be modified, we must also have an additional method. So, in this sense a priority queues very different from a circular queue circular queue implementation of the queue itself or the circular array implementation, because provides an additional method, where the priority value or the key value of an element in the queue can be changed.

(Refer Slide Time: 19:15)



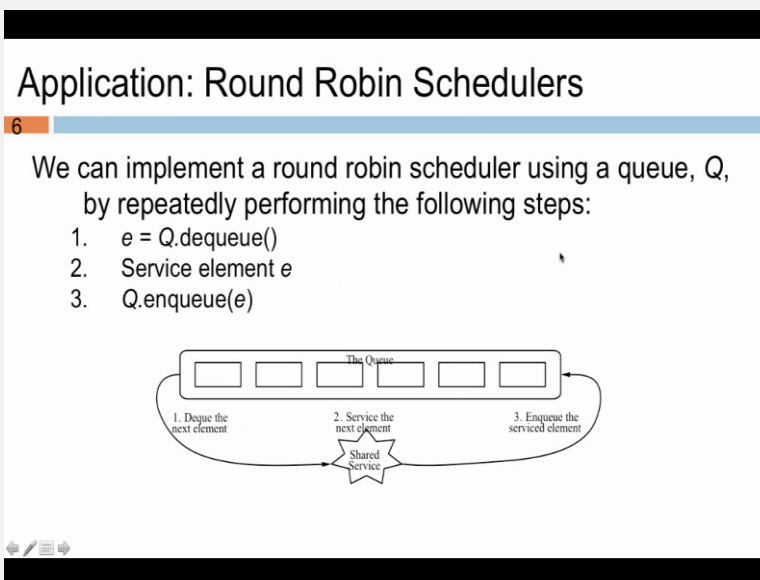
How do we implement priority queues, there are two ways of an implementing priority queues, one is that, you can implement it in an unsorted list. For example, let us this look at this, if this queue 4 is the head of the queue 5 is the next element of the queue 2, then and 1 is the last place. Then, whenever we want to insert, you just insert the met the beginning or the end of this particular sequence that should not be very hard. But, remove min or finding out the element of the lease priority will essentially require us to traverse the whole list and find out the element of least priority.

Therefore, to identify the element of least priority can take as much as the time taken to traverse the whole list. So, imagine that if you have queue of a 1000 people and you have to pick the element pick the person with the highest priority, then you have to inspect 1000 people's priorities before you pick them from the queue. Now, pick them from this unsorted list. If you perform an implementation with a sorted list, then whenever you want the element the higher priority you can immediately remove it in order of 1 time, you can immediately remove the element from the appropriating.

But, whenever you have to insert in element, you will have to go through the whole list and find the correct list insert the element into. The same is true with when you want to change the priority of elements, whenever you want to change the priority of the

elements, you will have to actually do quite a bit of work by inspecting each element, changing the priority and then reorganizing the list.

(Refer Slide Time: 20:48)



Where are priority queues, use the priority queues are use it what are called round robin schedules, the round robin scheduler, you dequeue the element, this element has the highest priority and while that element is being service, the priority of the other elements keeps increasing over a period of time. And after a certain amount of time, if the service of the element is being completed it course out of the queue, otherwise it enters back into the queue with a least priority.

In the mean while, the priority of the other elements have increase, among these you pick the element of the highest priority and then service. Therefore, these are called round robin schedulers. So, service you are scheduling the elements in this queue to be serviced here, you can imagine this to be a CPU and these are the processors manage by an operating system. So, round robin schedulers as are extremely popular schedulers.

(Refer Slide Time: 21:45)

## Applications of Queues

7

Waiting lists (e.g., printer)

- printer queue, keystroke queue

Multiprogramming

- Process Scheduling.

Auxiliary data structure for algorithms

- Priority queues used in efficient implementations

- Priority queues to be implemented in two weeks time.

Where do you find queues use like I was just saying, queues are views in process scheduling, that is an multi programming environment, where there is a single processor and multiple processors waiting to be executed. Then, multi programming provides each process, the illusions of being the single process using the CPU. So, process scheduling extensively uses the concept of queues and priority queues.

Similarly, printed queues and key stroke queues are essentially hardware queues that are maintained, for example, if you send a set of jobs to the printer, then to be printed, then they are all the printed in the order in which they have arrived at the queue. Queues also form an auxiliary data structure for different algorithms in one of the upcoming assignments, you will implementing queues and priority queues are use in efficient implementations of many algorithms. And we will study the implementation of a priority queue in about two weeks in that will be in your penalty made week of lectures for this online.

So, that so this brings to an end in this whole discussion, the sequence of 3 lectures on queues of course, two of the lectures were merged into 1, so that we have a single solid lecture to look into. So, we looked it the vanilla queue abstract data type, we came up with the circular queue implementation, we use this circular queue implementation in a



merging program, merging program is not just some ordinary program, it is use and recursion sorting algorithms like merge sort.

And then we looked at the different types of queues which are available and what their different purposes are. In subsequent lectures, we will move towards non-linear data types like trees and heaps, and then using heaps we will implement priority queues. We will definitely look at more implementations and this brings to an end, this week's lecture on the queues initially planned that it to be 3 lectures. But, I am into 2 lectures which will be uploaded enjoy these lectures and a look forward to a commencing questions also look forward for the programming assignment in a short.

Thank you.