Queues, Deques, and Priority Queues
Queue ADT

• Stores collection of items in the order they were added to the queue

• Supports:
  Queue()
  void enqueue(Item i)
  Item dequeue()
  boolean isEmpty()
  int size()
When an item is removed from a queue, it is...?

A. The one that has been in the queue for the longest time
B. The one that has been in the queue for the shortest time
C. The smallest one in the queue
D. The largest one in the queue
E. Not exactly one of the above
When an item is removed from a queue, it is...?

A. The one that has been in the queue for the longest time
B. The one that has been in the queue for the shortest time
C. The smallest one in the queue
D. The largest one in the queue
E. Not exactly one of the above
When using `enqueue()` to place the following items into a queue:
  `enqueue(32)`
  `enqueue(65)`
  `enqueue(0)`
  `enqueue(23)`
  `enqueue(-1)`
the output when dequeueing from the queue is:

A. -1, 23, 0, 65, 32
B. -1, 0, 23, 32, 65
C. 65, 32, 23, 0, -1
D. 32, 65, 0, 23, -1
E. None of the above
When using enqueue() to place the following items into a queue:
   enqueue(32)
enqueue(65)
enqueue(0)
enqueue(23)
enqueue(-1)
the output when dequeueing from the queue is:

A.  -1, 23, 0, 65, 32  D.  32, 65, 0, 23, -1
B.  -1, 0, 23, 32, 65  E.  None of the above
C.  65, 32, 23, 0, -1
Deque ADT

• Stores collection of items in the order they were added to the deque and allows access to either the newest or the oldest item

• Supports:
  – void addToFront(T), void addToBack(T)
  – T removeFront(), T removeBack()
  – T getFront(), T getBack()
  – boolean isEmpty()
  – void clear()
Priority Queue ADT

• Store collection of values and read them out in priority order (as given by a Comparator)

• Operations:
  – void enqueue(value)  //add value to collection
  – value dequeue()      //remove & return next val
  – value peek()         //return next val w/o removing
  – int size()           //get collection size
Queue implementation 1

• Array with front element of queue at index 0

• Attribute rear is index of last element
  – Empty queue denoted by rear being -1
public E dequeue(){
    // potential issue if empty, for now, assume not empty
    E e = array[front];
    <YOUR CODE HERE>
    return e;
}

Select the correct code to insert from below:

A. front++;  
B. rear = rear-1;
C. for(int i= 0; i<rear; i++)
    array[i] = array[i+1];
    rear = rear -1;
D. None of these are correct
public E dequeue(){
    // potential issue if empty, for now, assume not empty
    E e = array[front];
    <YOUR CODE HERE>
    return e;
}

Select the correct code to insert from below:

A. front++;  
B. rear = rear-1;
C. for(int i= 0; i<rear; i++)
    
    array[i] = array[i+1];
    
    rear = rear -1;
D. None of these are correct
Queue implementation 2

• Array with rear element of queue at index 0

• Attribute front is index of element at head of queue
  – Empty queue denoted by front being -1
public void enqueue(E element) {
    // potential issue if full, for now, assume room
    <YOUR CODE HERE>
    front++;
}

Select the correct code to insert from below:

A array[0] = e;

B array[front] = e;

C for(int i = 0; i < front; i++) {
       array[i+1] = array[i];
    }
    array[front] = e;

D None of these are correct.
public void enqueue( E element){
   // potential issue if full, for now, assume room
   <YOUR CODE HERE>
   front++;
}

Select the correct code to insert from below:

A    array[0] = e;
B    array[front] = e;
C    for(int i= 0; i<front; i++) {
       array[i+1] = array[i];
    }
    array[front] = e;
D    None of these are correct