Write a function splitQueue to be outside ArrayQueue class and accepts three parameters q1,q2 and q3 of type ArrayQueue. The function splits q1 into q2 and q3 by copying the elements of q1 alternately in q2 and q3. If q1 is empty the function returns false, otherwise, it returns true.

public static<E> boolean splitQueue(ArrayQueue<E> q1, ArrayQueue<E> q2, ArrayQueue<E> q3)

{

ArrayQueue<E> q11= new ArrayQueue<E> (q1);

Iterator<E> iter= q11.iterator();

if (!iter.hasNext()) return false;

E a ;

while( iter.hasNext())

{

q2.offer(iter.next());

if (! iter.hasNext()) return true;

q3.offer(iter.next());

}

return true;

}

**Solution without iterator**

public static<E> boolean splitQueue(ArrayQueue<E> q1, ArrayQueue<E> q2, ArrayQueue<E> q3)

ArrayQueue<E> q11= new ArrayQueue<E>(q1);

if (q11.size() == 0)

return false;

while (q11.size() !=0)

{

q2.offer(q11.poll());

if(q11.size() == 0) return true;

q2.offer(q11.poll());

}

return true;

}

Write a function mergeQueue to be outside ArrayQueue class and accepts three parameters q1,q2 and q3 of type ArrayQueue. The function merges q1 and q2 into q3 by copying one element from q1 then q2 and so on. The function returns false, if any of q1 or q2 is empty, otherwise, it returns true.

public static<E> boolean mergeQueue(ArrayQueue<E> q1, ArrayQueue<E> q2, ArrayQueue<E> q3)

{ ArrayQueue<E> q11= new <ArrayQueue>(q1);

ArrayQueue<E> q22= new <ArrayQueue> (q2);

Iterator<E> iter= q11.iterator();

Iterator<E> iter1 = q22.iterator();

if (!iter.hasNext() || !iter1.hasNext()) return false;

E a;

while ( iter.hasNext() && iter1.hasNext())

{

q3.offer(iter.next());

q3.offer(iter1.next());

}

while(iter.hasNext())

q3.offer(iter.next());

while(iter1.hasNext())

q3.offer(iter1.next());

return true;

}

**Solution without Iterator**

public static<E> boolean mergeQueue(ArrayQueue<E> q1, ArrayQueue<E> q2, ArrayQueue<E> q3)

{

ArrayQueue<E> q11= new <ArrayQueue>(q1);

ArrayQueue<E> q22= new <ArrayQueue> (q2);

if(q11.size() != 0 || q22.size() != 0)

return false;

while (q11.size() != 0 && q22.size()!=0)

{ q3.offer(q11.poll());

q3.offer(q22.poll());

}

while (q11.size() !=0)

q3.offer(q22.poll());

while (q22.size() !=0)

q3.offer(q22.poll());

return true;

}

Write a function createQueue to be outside ArrayQueue class that accepts two objects q1 and q2 (empty) of type ArrayQueue as parameters and have integers as their values. The function is also having a third parameter item of type integer. If the summation of the first two elements of q1 is greater than item then the function inserts the summation in q2, else it inserts item in q2. The same process will be repeated for the third and fourth element of q1 and so on. If the number of elements in q1 is odd, then ignore the last element of q1 for creating q2. If q1 is empty return false, otherwise return true.

public static boolean createQueue(ArrayQueue<Integer> q1, ArrayQueue<Integer> q2, int item)

{

ArrayQueue<Integer> q11= new ArrayQueue<Integer>(q1);

Iterator iter= q11.iterator();

if(! iter.hasNext()) return false;

int elem1;

int elem2;

int sum;

/\*This works well because with each poll the size is reduced until

the size becomes 0 and count also is 0 so hasNext becomes false.

No need to use next() \*/

while( iter.hasNext())

{

elem1= q11.poll();

if(iter.hasNext())

{

elem2 = q11.poll();

sum= elem1 + elem2;

if (sum > item)

q2.offer(sum);

else

q2.offer (item);

}

}

return true;

}

**Solution without Iterator**

public static boolean createQueue(ArrayQueue<Integer> q1, ArrayQueue<Integer> q2, int item)

{

if(q1.size() == 0)

return false;

int elem1, elem2;

while(q1.size() != 0)

{

elem1 = q1.poll();

if (q1.size() !=0)

{

elem2 = q1.poll();

sum = elem1 + elem2;

if (sum > item)

q2.offer(sum);

else

q2.offer(item);

}

}

return true;

}

Write a function addQueueWithCondition to be considered in the class ArrayQueue that accepts one parameter item of type E. The function adds item to the queue if it is greater than the integer average of the first element and the second element of the queue. If the queue is empty or has one element, it will not be added to the queue. You are not allowed to use any member function except reallocate() method.

public void addQueueWithCondition (E item)

{

if(size <= 1) return;

int V1 = (Integer)theData[front];

int V2 = (Integer)theData[(front+1)%capacity];

int avg = (V1+V2)/2;

if((Integer)item > avg)

{if (size == capacity) reallocate();

rear = (rear+1) % capacity;

theData[rear] = item;

size++;

}

}

Write a function **removeFromQueue** to be considered within the ArrayQueue class that has two parameters item of type E and q1 of type ArrayQueue. The function removes all the elements greater than item from the queue and copy them to q1.

public void removeFromQueue(E item, ArrayQueue<E> q1)

{

ArrayQueue<E> qcopy = new ArrayQueue<E>();

if (size==0) return;

E a;

while(size !=0)

{ a = poll();

if(((Comparable)(a)) .compareTo ((Comparable) (item)) > 0)

q1.offer(a);

else

qcopy.offer(a);

}

while (qcopy.size !=0)

offer(qcopy.poll());

}