**HW3**
**name: Shahad Almuhizi**
**id: 436201525**
**section:44126**

Problem 1:
1.

```
public static <T> T serveLast(Queue<T> q) {

T val;
for (int i=0;i<q.length();i++){
val=serve();
enqueue(val);}

val=serve();
return val;}
```

2.

```
public static <T> T retrieveLast(Queue<T> q) {

T val;
for (int i=0;i<q.length();i++){
val=serve();
enqueue(val);}
return val;}
```

3.

```
public static <T> Queue<T> merge (Queue<T> q1, Queue<T> q2) {

Queue<T> q=new Queue<T>();

T val;

int count=0;

while (! q.Full()){
```

```
if (count!=q1.length()){

val=q1.serve();

q1.enqueue(val);

q.enqueue(val);}

if(count!=q2.length()){

val=q2.serve();

q2.enqueue(val);

q.enqueue(val);}

count++;}}
```

4.

```
1 public static <T> boolean isPalindrome (Queue<T> q) {
2 int size=q.length() /2;
3 int count=q.length();
4 T val1, val2;
5 val1=serve();
6 enqueue(val1);
7 while (size!=0){
8 for (int i=0;i<count;i++){
9 val2=serve();
10 enqueue(val2);}
11 if (val1==val2){
12 size--;
13 count-3;}
14 else
15 return false;
16
```

```
17 q.findNext();
18 val1=serve();
19 enqueue(val1);}
20 return true;}
```

Problem 2:

2.1:

```
1 public T serveLast(){
2 Node<T> c;
3 c=tail;
4 tail=tail-1;
5 return c;}
```

2.2

```
1 public void remove(int i){
2 int count=0;
3 Node <T> c;
4 while (count!=i){
5 c=c.next;
6 count++;}
7 c=null;}
```

2.3

```
1 private void insert(Queue<T> q, int i){
2 int size;
3 Node<T> w;
4 while (w!=null){
5 size++;
6 w=w.next;}
7 for (int x=i;x<size; x++)
8 nodes[x]=nodes[q.data];}
```

Problem 3:

3.1

```
public class PQNode<T> {
   public T data;
   public Priority priority;
```

```
    public PQNode<T> next;

    public PQNode() {
                next = null;
    }

    public PQNode(T e, Priority p) {
                data = e;
                priority = p;
    }

}
public class LinkedPQ<T> {
    private int size;
    private PQNode<T> head;


    public LinkedPQ() {
                head = null;
                size = 0;
    }
public int length (){
                return size;
    }



    public boolean full () {
                return false;
```

```
    }
public void enqueue(T e) {
            if(tail == null){
                    head = tail = new Node<T>(e);
            }
            else {
                    tail.next = new Node<T>(e);
                    tail = tail.next;
            }
            size++;
    }

public PQElement<T> serve(){
            PQNode<T> node = head;
            PQElement<T> pqe=new PQElement<T>(node.data,node.p);
            head = head.next;
            size--;
            return pqe;
    }
```

3.2:

```
1 public class ArrayPQ<T>{
2 private int maxsize;
3 private int size;
4 private int head;
5 private PQElement<T>[] data;
6
```

```
 7 public ArrayPQ(int n){
 8 head=0;
 9 size= 0;
10 maxsize= n;
11 data= (PQElement<T>[]) newPQElement<?>[n];}
12
13 public boolean full(){
14 return size== maxsize;}
15
16 public int length(){
17 return size;}
18
19 public PQElement<T> serve(){
20 PQElement<T> temp= data[head];
21 head++;
22 size--;
23 return temp;}
24
25 public void enqueue(T e, int pty){
26 PQElement<T> temp= new PQElement<T>(e, pty);
27 if(size== 0)
28 data[head] = temp;
29 else if(pty> data[head].p)
30 if(head!= 0){
31 data[--head] = temp;
32 size++;
33 return;}
34
35 if(head!= 0) {
36 for(int x= 0; x< data.length; x++)
37 if(head+ x< data.length)
38 data[x] = data[head+ x];
39 else
40 data[x] = null;
41 head= 0;}
42
43 int index= head;
44 int i= 0;
45 if(!(data[head].p< pty)){
46 for(i= 0; i< size; i++){
47 if(data[index+ 1] != null)
48 if(data[index].p>= pty && pty> data[index+ 1].p){
49 index++;
50 break;}
51 index++;}}
52 for(intj= size-1; j>= i; j--) {
```

```
53 data[j+ 1] = data[j];
54 if(j== index)
55 break;}
56 data[index] = temp;size++;}}
```

Problem 4:

4.1:

```
 1 LinkedPQ<Item> temp= new LinkedPQ<Item>();
 2 LinkedList<ItemPair> list= new LinkedList<ItemPair>();
 3 ItemPair b;
 4 int size;
 5 items.findfirst();
 6 while(!items.last()) {
 7 temp.enqueue(items.retrieve(), items.retrieve().getPrice());
 8 items.findnext();}
 9 temp.enqueue(items.retrieve(), items.retrieve().getPrice());
10 size= temp.length();
11 for(int i= 0; i< size; i= i+ 2) {
12 b= new ItemPair(temp.serve().data, temp.serve().data);
13 list.insert(b);}
14 if(size% 2 != 0) {
15 b= newItemPair(temp.serve().data, null);
16 list.insert(b);}
17 return list;}
```

4.2:

```
 1 public static LinkedList<ItemPair>
maxPairing(LinkedList<Item> items){
 2 LinkedPQ<Item> temp= new LinkedPQ<Item>();
 3 LinkedList<ItemPair> list= new LinkedList<ItemPair>();
 4 ItemPair b;
 5 int size;
 6 items.findfirst();
```

```
while(!items.last()){
temp.enqueue(items.retrieve(), items.retrieve().getPrice());
items.findnext();}
temp.enqueue(items.retrieve(), items.retrieve().getPrice());
size= temp.length();
LinkedList<Item> tempList= new LinkedList<Item>();
for(int j= 0; j< size; j++)
tempList.insert(temp.serve().data);

Item item1, item2;

while(!tempList.empty()){
tempList.findfirst();
item1= tempList.retrieve();
tempList.remove();
if(!tempList.empty()){
while(!tempList.last())
tempList.findnext();
item2= tempList.retrieve();
tempList.remove();}
else
item2= null;

b= new ItemPair(item1, item2);
list.insert(b);}
return list;}
```

4.3:

**When I use min pairing method:**
**The first pair = 600 SR and 400 SR**
**The second pair = 200 SR and 100 SR**
**The third pair = 80 SR and 60 SR**

**Total: 1160 SR instead of 1320 SR (When I use the max pairing method)**
**I will gain 160 SR**