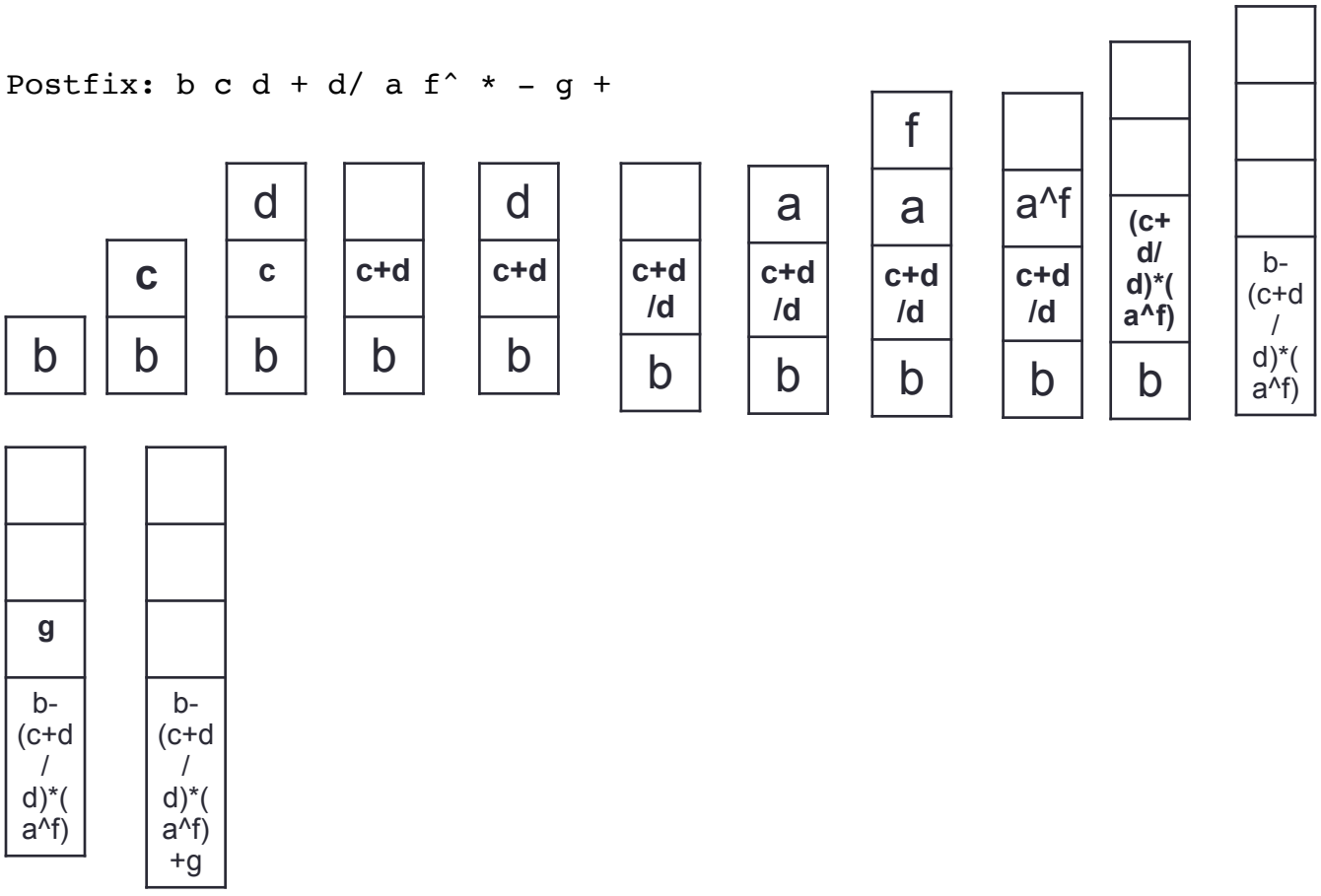
SHAHAD ALMUHIZI
436201525

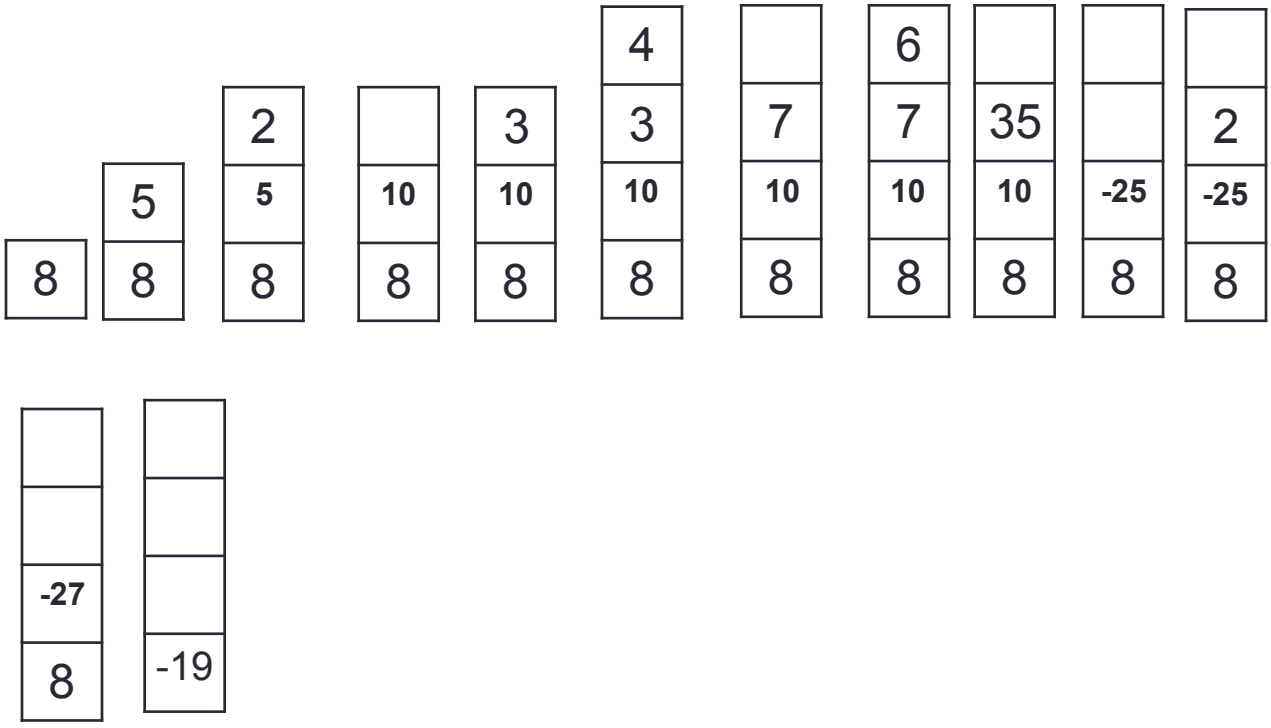
Problem 1:
1.

Postfix: b c d + d/ a f^ * - g +



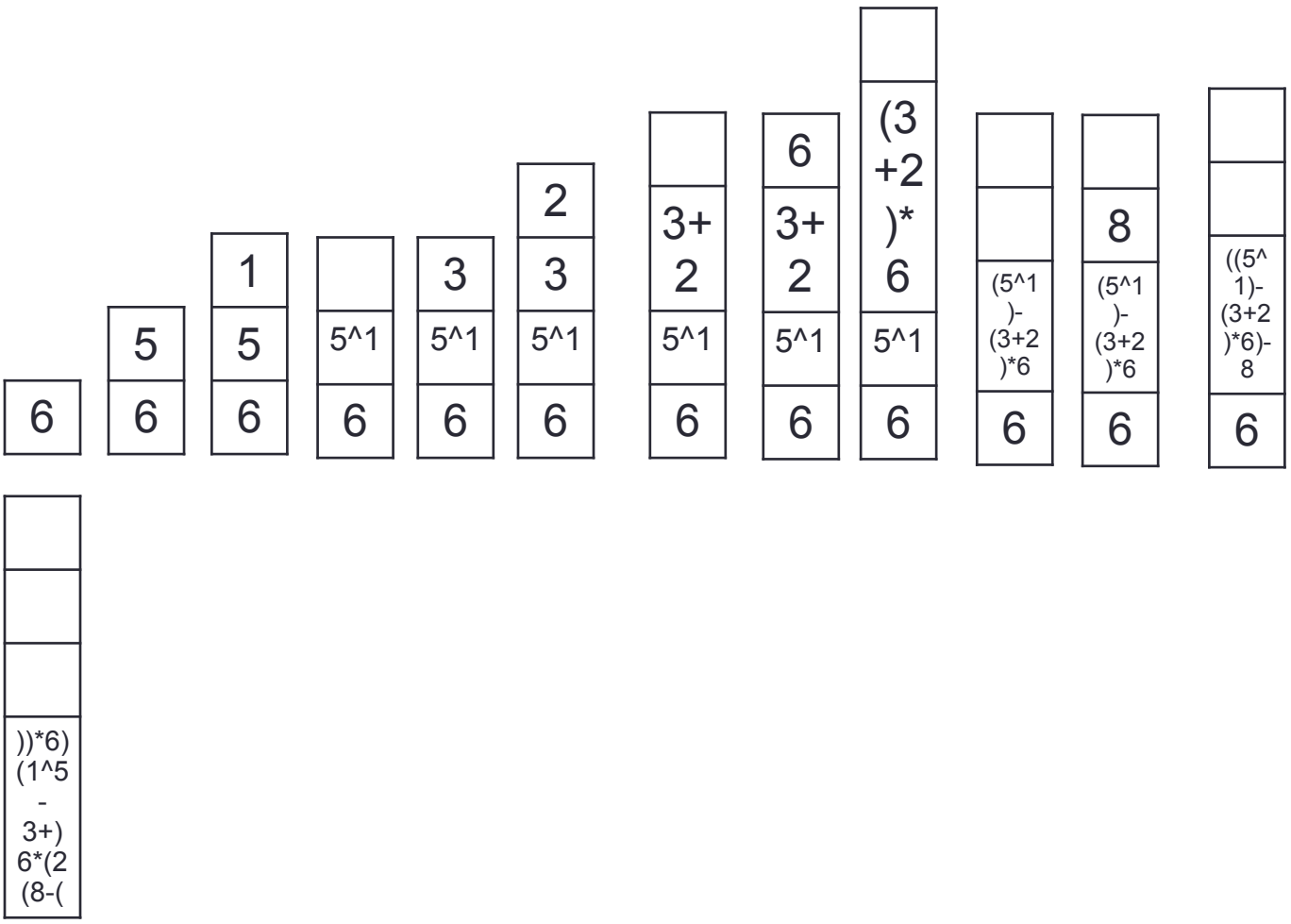
2.

8 5 2 ^ 3 4 + 6 * – 2 – +



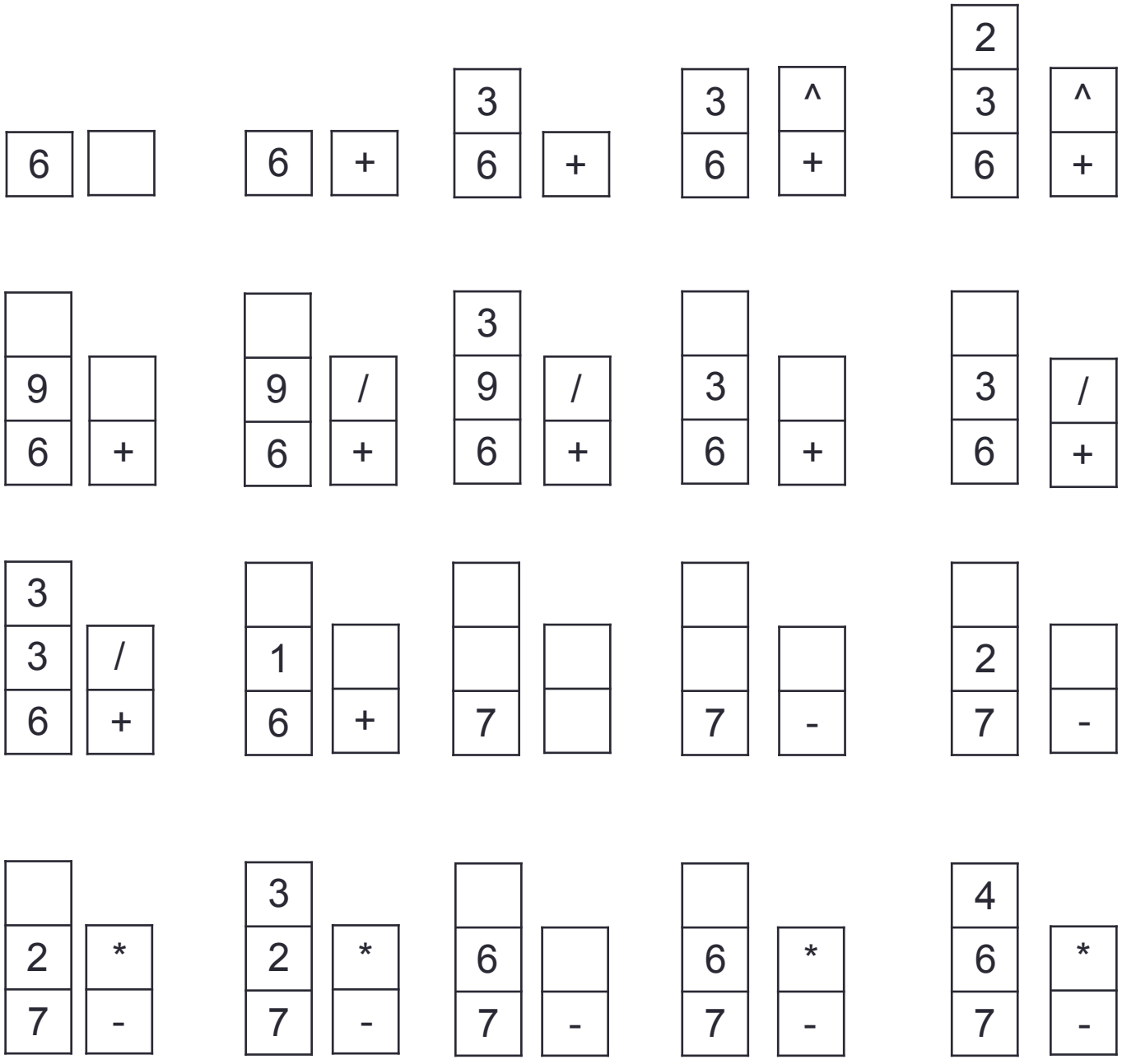
3.

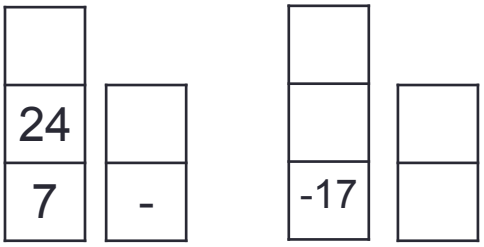
infix: (6 * (((5^ 1) - ((3+2) *6)) -8))



4.

6 + 3 ^ 2 / 3 / 3 – 2 * 3 * 4





Problem 2 :

1.

```
public static <T extends Comparable<T>> Stack<T>
mergeSortedStacks(Stack<T> s1, Stack<T> s2){

Stack <T> tmp = new Stack <T> ();

Stack <T> tmp1=new Stack<T> ();

T min, max, p ,q;

while (! s1.empty() ){
p=s1.pop();
q= s2.pop();

if (p.compareTo(q) > =0)

tmp.push( q );

else {

tmp.push( p );

while (! tmp.empty())

tmp1.push(tmp.pop());}
```

Text

else
tmp.push(p);

```
public static <T extends Comparable<T>> Stack<T> mergeSortedStacks(Stack<T> s1, Stack<T> s2){
Stack <T> tmp = new Stack <T> ();
Stack <T> tmp1=new Stack<T> ();
T min, max, p ,q;
while (! s1.empty() ){
p=s1.pop();
if (!s2.empty()){
q= s2.pop();
if (p.compareTo(q) > =0){
tmp.push(q);
s1.push(p);}
else{
tmp.push(p);
s2.push(q);}}
else{
tmp.push(p);}}
while (!s2.empty())
tmp.push(s2.pop());

while (! tmp.empty())
tmp1.push(tmp.pop());}
```

2.

```
public static <T> void pushElement(Stack<T> st, T e){

Stack <T> tmp=new Stack <T>();
  Stack <T> tmp1=new Stack<T>();
T val;

int count=0;

while (! st.empty()){

val=st.pop();

if (
     val.equals(e)==false)
tmp.push(val);

else

count ++ ;}

while (count!=0){

tmp.push(e);

count- -; }
 while (!tmp.empty())
tmp1.push(tmp.pop());}
```

Problem 3:

1.

```
public static <T extends Comparable<T>> sort(Stack <T> st){

if (st.empty() )

return;

T e, f, tmp;

e=st.pop();

val=st.pop();
```

```
if (f!=null){

if (e.compareTo(f) <0 ){

tmp= f;

f=e;

e=tmp;}

st.push(val);}

st.push(e);

else

sort(st);}}
```

another sol:

```
public static <T extends Comparable<T>> sort(Stack <T> st){

if (st.empty() )

return;

T x=st.pop();

recSort(st, x);

st.push(x);}

private static <T extends Comparable<T>> recSort(Stack <T> st, T
     pre){

if (st.empty())

return;
```

```
T x=st.pop();

if (x.compareTo(pre) <0)

st.push(x);

recSort(st, x);

st.push(x);}
```

2.

```
public boolean recSearch(T k){

return recSearch(k, head); }

private boolean recSearch(T k, Node<T> p){

if (p == null)

return false;

if (p.data.equals(k))

return true;

return recSearch(k, p.next);}
```

3.

```
public <T > void reverse(Queue <T > q){
if (q.length()==0)
return;
T tmp=q.serve();
reverse(q);
```

```
q.enqueue(tmp);}
```

4.

```
public <T> Queue<T> merge(Queue<T> q1, Queue<T> q2){
return recMerge(q1, q2, new Queue<T>());}

public <T> Queue<T> recMerge(Queue<T> q1, Queue<T> q2, Queue<T>
q){
if (q1.length() ==0 && q2.length() ==0 )
return q;
if (q1.length()!=0)
q.enqueue(q1.serve());

if (q2.length()!=0)
q.enqueue(q2.serve());

return recMerge(q1, q2, q);}
```

Problem 4:
1.

```
public LinkedList<T> T car(){
if (l==null)
return;

return head;}

public <T> LinkedList<T> list cdr(){
if (l==null)
return l;

else {
head=head.next;
return l;}}
```

2.

```
public static <T> List<T> list(T e){
List <T> l= new List<T> ();

return l.insert(e); }
```

3.

```
public static <T> List<T> concat(List<T> l1, List<T> l2){
List <T> l=new List<T>();
l.findFirst();

while (!l1.last()){
l.insert(l1.retrieve());
l1.findNext();}

l.insert(l1.retrieve());

while(!l2.last()){
l.insert(l2.retrieve());
l2.findNext();}
l.insert(l2.retrieve());
return l;}
```

4.
(a):

```
public static <T> void print(List<T> l){

if (l.empty())
return;


System.out.println(l.retrieve());
print(l.findNext());}
```

(b):

```
public static <T> List<T> inverse(List<T> l){
if (l.empty())
return;
List<T> list=new List<T>();
```

```
while(!l.last()){
list.insert(l.retrieve());
l.findNext();}
list.insert(l.retrieve());

return recInverse(list);}

private static <T> List<T> recInverse(List<T> list){
if (list.empty() || list.car().findNext().empty())
return list;
T q;
T p=list.car();
T tmp=p.findNext().retreive();
p.findNext().retreive()=q;
q=p;
p=tmp;

recInverse(list);
return list;}
```

(c):

```
public static <T> List<T> remove(List<T> l, T e){
if (l.empty())
return;
List<T> list=new List<T>();

while(!l.last()){
list.insert(l.retrieve());
l.findNext();}
list.insert(l.retrieve());

T val=list.car();
if (val.equals(e)){
list.retrieve()=list.findNext().retrieve();
return list;}

list.cdr.findFirst();
return recRemove(list, e);}

private static <T> List<T> remove(List<T> list, T e){
if (list.empty())
return;
```

```
if(list.car().findNext().empty())
return;



T val=list.sdr.retrieve();
if (val.equals(e){
list.retrieve()=list.findNext().retrieve();
return list;}

list.sdr.findNext();

return remove(list);}
```