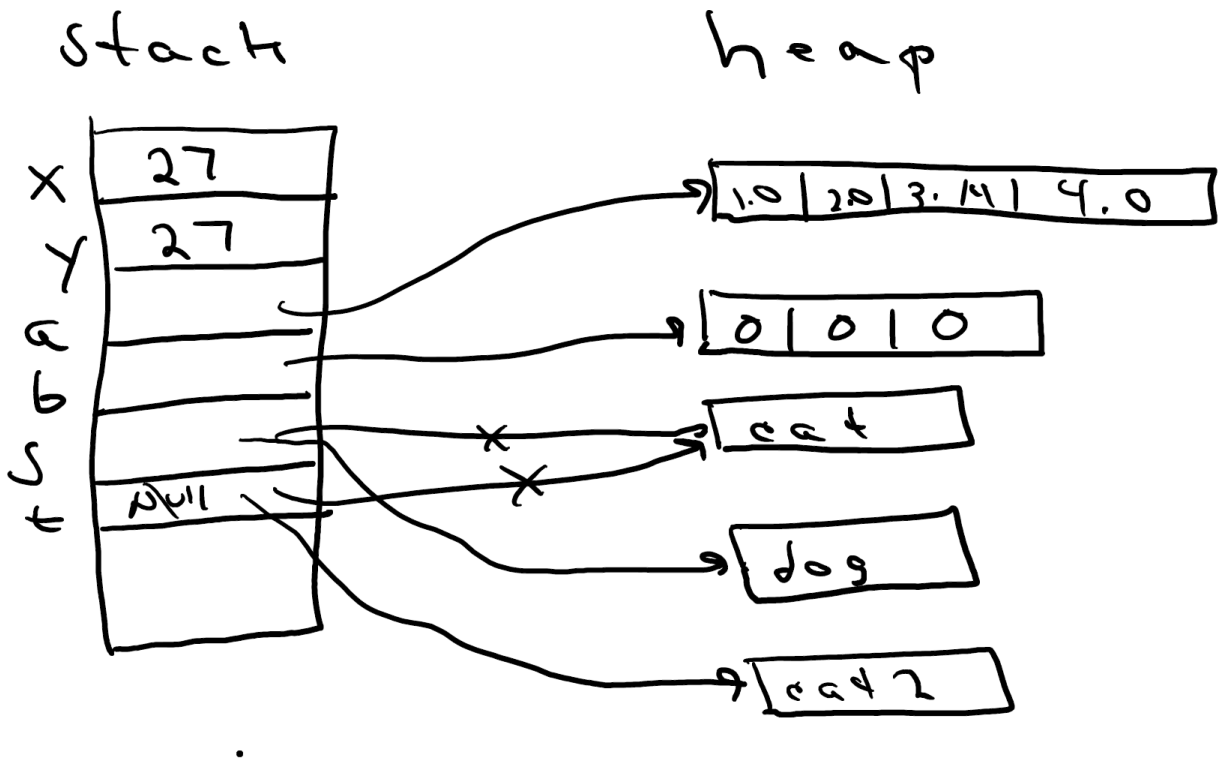
1. (10 points, 1 point off for each incorrect item, no negative scores)



2. (6 points, 3 points each, -1 for each wrong answer, no negative scores)

a) a, d, c, b
b) b, a
c) b, c, a

3. (6 points, 2 points each)

a) Not valid (x incremented, it can't still be 1}
b) Not valid (if x == -1, x will equal 0 after execution)
c) Valid

4. (6 points, 2 points each)

a) Valid
b) Not necessarily valid
c) valid

5. (12 points, 2 points each condition

a)

P: {((x>=0) && (x!=0)) || ((x!=-1) && (x<0)} = (x>0) || (x < -1)
if (x >=0) {
{wp(z=x, z!=0)=(x!=0)}
z = x;
}
else {
{wp(z=x+1, z!=0)=(x+1 !=0)=(x!=-1)}
z = x+1;
}
Q: z != 0

b)
P: {wp(b=b+1, b>0) = {b+1 > 0} = (b>-1) equivalent (b >= 0)
b = b + 1
{wp(a = b+1, a>0 && b>0) = (b+1>0 && b >0) = ((b>-1) && (b>0))=(b>0)
a = b + 1
{a > 0 && b > 0}

c)

P:wp(t=x, y>t) = (y>x)
t = x;
wp(x=y, x>t) = (y>t)
x = y;
wp(y=t, x>y) = (x>t)
y = t;
{x > y}

6. (12 points, 2 points for each condition.

a)

```
        { x < 0 && y > 0 }
  y = 2;
        { x < 0 && y = 2 }
  x = x + y;
```
{x_old < 0 && x == x_old + y && y == 2} = { x < 2 & y == 2 }

b)

```
        { |x| > 2 } = {x>= 3 || x <= -3}
  x_1 = x * 2;
```

```
        { x_1 >= 6 || x_1 <= -6}
             or you could also say { |x| > 4 }
        // should be { x_1 >= 6 || x_1 <= -6 && x_1 is even }
        // or { |x| > 4 && x is even }
    x_2 = x_1 - 1;
        { x >= 5 || x <= -7  } or also { x > 3 && x < -5 }
        // should be { x >= 5 || x <= -7 && x is odd }
        // or { x > 3 && x < -5 && x is odd }
```

In a case like this I would give credit for the answer { x >= 5 || x <= -7 } or {x > 3 || x < -5}.

7. (12 points, 2 points each)

a) false
b) false
c) false
d) true (if a is null, precondition that key is contained in a is violated)
e) true
f) false (fails if a is not sorted)
g) false
h) false

8 (12 points)

	(a)	(b)	(c)	(d)	(e)
(i)	X				
(ii)	X	X		x	X
(iii)					
(iv)	X	X			X
(v)	x	x			x

Why isn't iii, e checked?
If you assume Java, it returns a negative value for a negative argument.

For example,

System.out.println(-25 % 10) prints -5.

In C
x = 5 % (-3);
y = (-5) % (3);
z = (-5) % (-3);

printf("%d ,%d ,%d", x, y, z); prints (2, -2 , -2) in gcc.

See this https://stackoverflow.com/questions/11720656/modulo-operation-with-negative-numbers discussion for more details about the C case.

In Python

>>> -25 % 10
5
>>>
Given the Python result, on a test, I would have to give credit for specifying that it could return a non-negative number.

9 (17 points,)

LI: n <= arr.length && p = arr[0]*arr[1]*arr[2]…arr[n-1] (2 points)

Or LI: n <= arr.length && p = prod(arr[0]..arr[n-1])

D: arr.length – n (2 points)

Proof:

Base: n = 1; p = arr[0], only one element and it's equal to p, so base case holds (3 points)

p == arr[0] => prod(arr[0]..arr[n-1]) == arr[0]

n == arr.length == 1 => n <= arr.length (arr.length can't be 0)

Induction: step k, assume p(k) = arr[0]*arr[1]*arr[2]…arr[n(k)-1] is true (4 points)

n(k) <= arr.length

Step k + 1:

n(k+1) =n(k) + 1

p(k+1) = p(k) * arr[n(k)] = p(k)* arr[n(k+1) – 1]

p(k+1) = (arr[0]*arr[1]*arr[2]…arr[n(k) -1]) * arr[n(k+1) - 1]= arr[0]*arr[1]*arr[2]…arr[n(k+1) -1]

at step k, n(k) < arr.length or we would have exited.

at step k+1 , n(k+1) = n(k) + 1 => n(k+1) <= arr.length

Decrementing: (3 points)

D(k) = arr.length – n(k)

D(k+1) = arr.length – n(k+1) = arr.length – (n(k)+1)=D(k)-1

D_k+1 < D_k

D = 0 => arr.length == n => !(n <arr.length)

Partial correctness (3 points)

!(exit condition) && LI => postcondition

!(n < arr.length) + (n <= arr.length && p == prod(arr[0]..arr[n-1])

=> (n==arr.length) && (p == prod(arr[0]..arr[n-1])

=> p = prod(arr[0]…arr[arr.length-1])

10) 10 points

a) (5 points)
Requires: true or none
Modifies: nothing
Effects: none
Returns: this[index]; element at index
Throws: IndexOutOfBoundsException - if the index is out of range (index < 0 || index >= size())

b) (5 points)
E = if index < 0 || index >= size
Throws IndexOutOfBoundsException
Else
return this[index]
true ➔E &&(this_post[i] == this_pre[i] forall i :: 0 <= i < size)
or
true ➔ E && (nothing is modified)

11. (16 points)

a) (2 points)
LI: `(total + x*y == a*b) && (x >= 0)`

b) (2 points)
initially x=a, y=b, total=0
LI: (`total + x*y == a*b) && (x >= 0)`
Substituting values (0 + a*b == a*b) && (a >= 0)
The first term is true by definition, second term is true by the precondition

c) (8 points) (4 points each for even and odd cases)

assume: `total + x*y == a*b) && (x >= 0)` holds at iteration k.
Two cases:
x is even:
x_new = x / 2

y_new = y * 2
total_new = total
total_new + x_new*y_new = total + (x/2) * (y*2) = total + x*y = a*b

x is odd:
total_new = total + y
x_new = (x-1) / 2
y_new = y * 2
total_new + x_new*y_new
= (total+y) + ((x-1)/2) * (y*2)
= (total + y) + (x-1)*y
= total + x*y = a*b

At iteration k, x >= 0, if x == 0 at iteration k, we don't have iteration k+1. x_new is either x/2 or (x-1)/2 in either case x_new >= 0.

d) (2 points)
!(x > 0) && ((total + x*y == a*b) && (x >= 0))
= (x==0) && (total + x*y == a*b)
= total == a*b

e) (2 points)
D = x
Initailly x>=0 by the precondition. At each iteration x_new = x/2 or x_new=(x-1)/2 so it decreases: D_new < D
When D = 0 => x=0 => !(x>0);