98

A(a) first blank: int[] ✓

second blank: {2,0,1,3,0} ✓

A(b) list[2]+list[3]

= 1+3

= $\boxed{4}$ ✓

A(c) list[list[0]+list[1]]

= list[2+0]

= list[2]

= $\boxed{1}$ ✓

A(d) list[list[3]+1]

= list[3+1]

= list[4]

= $\boxed{0}$ ✓

A(e) list[list[list[4]+list[2]]]

= list[list[0+1]]

= list[list[1]]

= list[0]

= $\boxed{2}$ ✓

B. 7777777

55555

333

1 ✓

```
if (n < 1)
  return 0;   } ⟹ To handle n≤0.
```

C (a)

```
double sum = 0;
for (int i= 1; i<=n; i++)
{
  sum += (1/i);
}
return sum;
```

~~2~~

1.0/i

1.1

C (b)

```
System.out.print (harmonicSeries(50));
```

D(a) 5 iterations ✓

output:

3 8 13 18 23 ✓

D(c) blank on line 1: < 32 ✓

blank on line 3: i-4 ✓

5 iterations ✓

{ correct, but $i \leq 27$ or $i < 28$ is the tight bound, else, $i < 29$, $i < 30$ . . . $i < 32$ are all correct }

10 0 6

D(d) The code segment in listing 4 is more efficient because although it has the same number of comparisons and add/subtract-type operations, it uses 5 less multiply/divide/modulus-type operations, and is thus more overall efficient