Answer Key

1. **E**
2. **C**
3. **E**
4. **A**
5. **C**
6. **C**
7. **D**
8. **A**
9. **D**
10. **B**
11. **C**
12. **E**
13. **B**
14. **C**
15. **A**
16. **B**
17. **A**
18. **B**
19. **D**
20. **C**
21. **D**
22. **B**
23. **E**
24. **C**
25. **D**
26. **A**
27. **E**
28. **D**
29. **D**
30. **E**
31. **A**
32. **A**
33. **B**
34. **D**
35. **B**
36. **D**
37. **D**
38. **E**
39. **E**
40. **D**
41. **C**
42. **A**
43. **E**

Answer Explanations

1. **(E)** Segment I is an initializer list which is equivalent to

```
int[] arr = new int[4];
arr[0] = 0;
arr[1] = 0;
arr[2] = 0;
arr[3] = 0;
```

Segment II creates four slots for integers, which by default are initialized to `0`. The `for` loop in segment III is therefore unnecessary. It is not, however, incorrect.

2. **(C)** If `arr` contains no negative integers, the value of `i` will eventually exceed `N-1`, and `arr[i]` will cause an `ArrayIndexOutOfBoundsException` to be thrown.

3. **(E)** The intent is to sum elements `arr[0]`, `arr[1]`, ..., `arr[arr.length-1]`. Notice, however, that when `i` has the value `arr.length-1`, it is incremented to `arr.length` in the loop, so the statement `sum += arr[i]` uses `arr[arr.length]`, which is out of range.

4. **(A)** The code segment has the effect of removing all occurrences of 0 from array `arr1`. The algorithm copies the nonzero elements to the front of `arr1`. Then it transfers them to array `arr2`.

5. **(C)** If `arr[i] < someValue` for all `i` from `2` to `k`, `SMALL` will be printed on each iteration of the `for` loop. Since there are `k - 1` iterations, the maximum number of times that `SMALL` can be printed is `k - 1`.