3. c) Why might this modification sometimes improve the efficiency of a sequential search?

If you were to frequently search for the same numbers, they would appear closer to the beginning of the array, making it easier to search them for the next time they're called on. This is under the assumption that the code is written so that the search stops once the value is found (a few extra lines of code with booleans).

I.e.

```
    public static int seqSearch (String[] list, String item)
{
  int location = -1;
  boolean found = false;
  for (int i = 0; i < list.length && !found; i++)
    if (list[i].equals(item))
    {
      location = i;
      found = true;
    }
  return location;
}
```

4. a) 72

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Data	23	27	30	34	41	49	51	55	57	60	67	72	78	83	96
1st	F							M							L
2nd									F			M			L

b) 41

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Data	23	27	30	34	41	49	51	55	57	60	67	72	78	83	96
1st	F							M							
2nd	F			M			L								
3rd					F	M	L								
4th					FM	L									

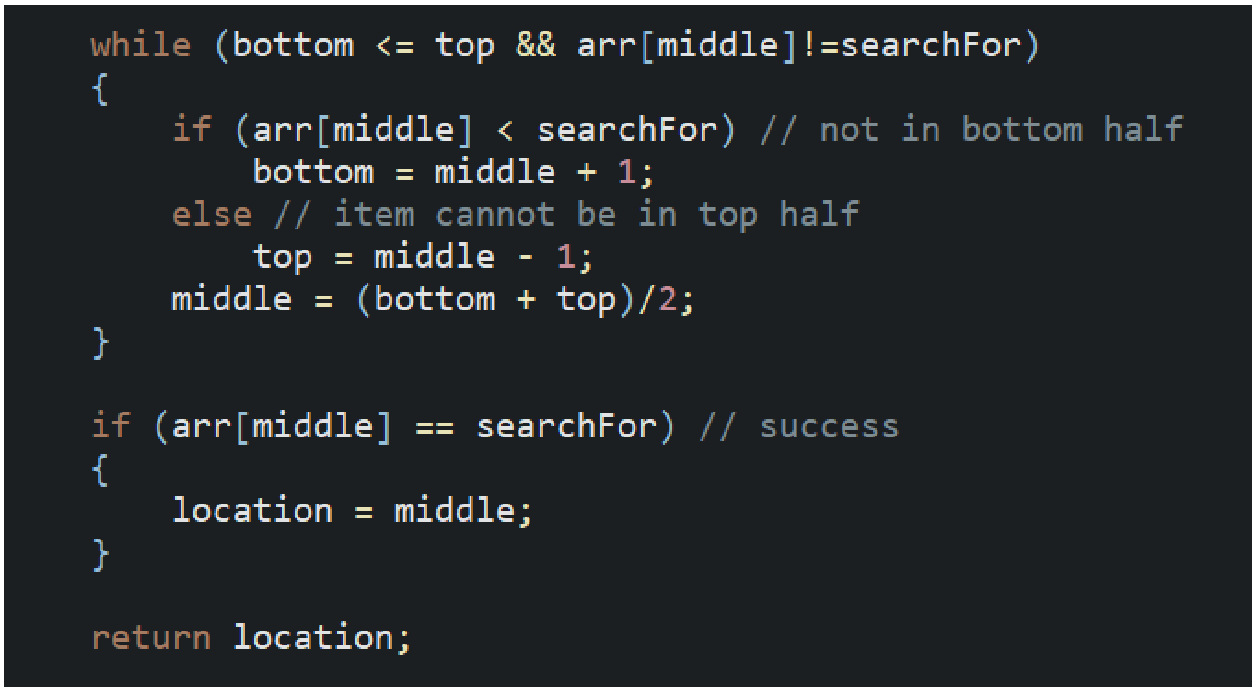
c) 62

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Data	23	27	30	34	41	49	51	55	57	60	67	72	78	83	96
1st	F							M							L
2nd									F			M			L
3rd									F	M	L				
4th									FM	L					

RESULT = -1

6. What change would have to be made so that it will search an array that is sorted in descending order?
In the if statement within the while loop, you would need to reverse the comparisons so that you're checking if the middle value in the array is **greater** than the value being searched.

Original code reference:



```
while (bottom <= top && arr[middle]!=searchFor)
{
    if (arr[middle] < searchFor) // not in bottom half
        bottom = middle + 1;
    else // item cannot be in top half
        top = middle - 1;
    middle = (bottom + top)/2;
}

if (arr[middle] == searchFor) // success
{
    location = middle;
}

return location;
```

8. (round down) a) 2 b) 1 c) 3 d) 4 e) 5 f) 6 g) 8 h) 9 i) 13