Matthew Vollkommer 8102494122 HW4

1. O(n)
2. F(x) = 60x; O(n); O(5) = 5
   Time(n) = 15n^2 +45n; O(n^2); O(5)= 25
   F(x) = 60x; O(n); O(1,000,000) = 1,000,000
   Time(n) = 15n^2 +45n; O(n^2); O(1,000,000) = (1,000,000)^2
3. O(n^2)
4.

	selection sort	insertion sort	bubble sort	quick sort	merge sort
Array at Begin:	11 44 10 5 21 110 121	11 44 10 5 21 110 121	11 44 10 5 21 110 121	11 44 10 5 21 110 121	11 44 10 5 21 110 121
After Pass #1:	11 44 10 5 21 110 121	11 44 10 5 21 110 121	11 10 5 21 44 110 121	5 10 11 44 21 110 121	11 44 5 10 21 110 121
After Pass #2:	11 44 10 5 21 110 121	10 11 44 5 21 110 121	10 5 11 21 44 110 121	5 10 11 21 44 110 121	5 10 11 44 110 121
After Pass #3:	11 21 10 5 44 110 121	5 10 11 44 21 110 121	5 10 11 21 44 110 121	5 10 11 21 44 110 121	5 10 11 44 110 121
After Pass #4:	11 5 10 21 44 110 121	5 10 11 21 44 110 121	5 10 11 21 44 110 121	5 10 11 21 44 110 121	
After Pass #5:	10 5 11 21 44 110 121	5 10 11 21 44 110 121			
After Pass #6	5 10 11 21 44 110 121		O(n^2)	O(n^2)	O(n log(n))
Worst Case :	O(n)	O(n^2)			
Comparisons:	21	12	21	37	33

5. It will always need at least three assignment statements to swap two values. The third statement is necessary to temporarily hold a value in memory as they are being moved.
6. append() and pop() are both O(1). If the top of the stack is at position 0, the algorithms are still the same(same logic) but append() and pop() are now O(n). They become slower.

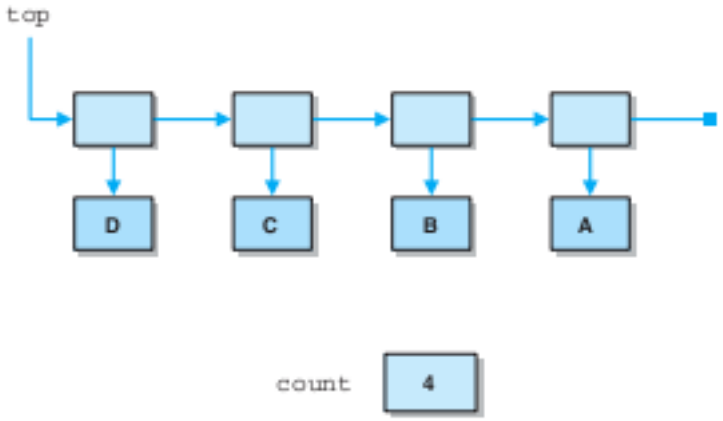


FIGURE 13.9 A linked implementation of a stack

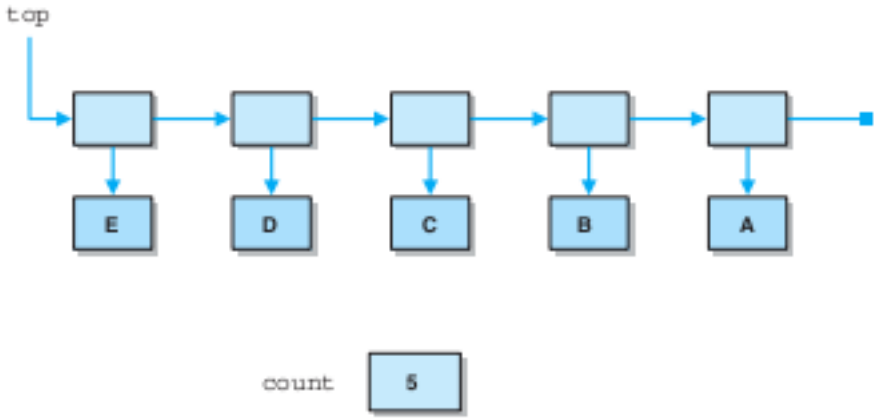


FIGURE 13.10 The stack after pushing element E

pg. 3

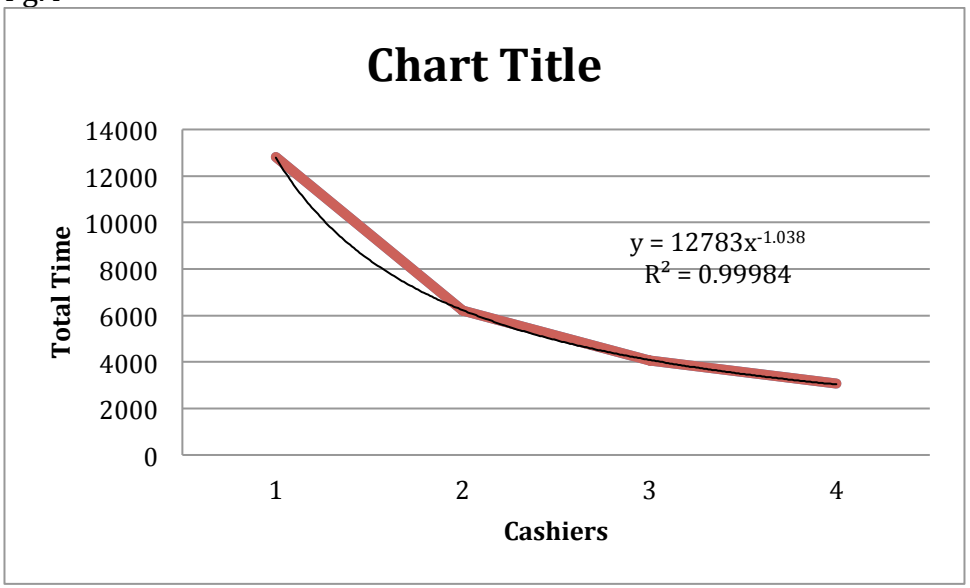
Instead of adding to the top from the beginning, as depicted n the book,
It will have to loop through the array, shifting each to the left one.

7. Time complexity is O(n)

```
public void push(String t) {
top++;
for(int counter = 0; counter < top; counter++){
        if(counter == 0){
                e [top] = t; }
}else {
        e[counter] = e[counter - 1]
}
}
```

8. It would be O(n)

9. The Time complexity for each algorithm would be O(n)
10.



With each additional cashier, total time changes by about 1/x;

11. Measuring these sorting algorithms with comparison makes the most since. They are all comparison sorting algorithms. An algorithm my compare elements without making a swap, sometimes even the whole list. Measuring with swaps would not be as accurate as measuring based on comparisons. Some of the algorithms may swap multiple times on a run through, even though they are more efficient than another sorting algorithm that compared all elements without making a swap.

12. 17 times