Big-O: Best, Average, and Worst Case

	Best case	Average case	Worst case
Sequential Search	$O(1)$ — found right away	$O(n)$ — found on average in the middle	$O(n)$
Binary Search	$O(1)$ — found right away	$O(\log n)$	$O(\log n)$
Hash table search	$O(1)$ — found right away	$O(1)$ — small fixed-length buckets	$O(n)$ — table degenerated into one or two buckets
Search in a binary search tree	$O(1)$ — found right away	$O(\log n)$	$O(n)$ — tree degenerated into nearly a list
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n)$ — array already sorted	$O(n^2)$	$O(n^2)$
Mergesort	$O(n \log n)$, or $O(n)$ in a slightly modified version when the array is sorted	$O(n \log n)$	$O(n \log n)$
Quicksort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$ — pivot is consistently chosen far from the median value, e.g., the array is already sorted and the first element is chosen as pivot
Heapsort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Insert a value into a heap	$O(1)$ — the value is the largest in the heap	$O(\log n)$	$O(\log n)$

Example:

The following method eliminates consecutive nodes with duplicate values from a linked list. For example, A→B→B→B→C→A→A becomes A→B→C→A.

```
public void skipDuplicates(ListNode head)
{
  ListNode p = head;
  ListNode p2;

  while (p != null)
  {
    p2 = p.getNext();
    while (p2 != null && p2.getValue().equals(p.getValue()))
    {
      p2 = p2.getNext();
    }
    p.setNext(p2);
    p = p.getNext();
  }
}
```

Which of the following best describes the best-case and worst-case running time for `skipDuplicates` on a list with *n* nodes?

	Best case:	Worst case:
(A)	$O(1)$	$O(n)$
(B)	$O(n)$	$O(n)$
(C)	$O(1)$	$O(n^2)$
(D)	$O(n)$	$O(n^2)$
(E)	$O(n^2)$	$O(n^2)$

No matter what exactly the "best case" is, the method has to examine every node of the list. The running time cannot possibly be constant; therefore, A and C are wrong answers. There is a case when all the nodes are different. In that case the inner loop is simply skipped, and we end up with a simple traversal of the list in one sequential loop. The running time in that case is $O(n)$, so E is not the right answer, either. We are left with B and D. This example is an exception to the rule of thumb: it has nested loops, so you might think the worst time might be $O(n^2)$. But this method eliminates consecutive duplicate values, not all duplicate values. The trick is that the outer loop doesn't necessarily go to the next node, but can jump over all the removed nodes. Even when all the nodes of the list are the same, each node is visited only once. We did warn you to use the rules of thumb with caution, didn't we? The answer is B.