Workshop 4 Solutions

August 19, 2017

1. The number of operations in binary search is of $O(\log n)$. At any given search iteration i the size of the array for next search is $\frac{n}{2^i}$. On search completion, this value will be 1, meaning number of iterations i needed are $\log n$.

2. In short, no. The problem with linked lists is the cost associated with travel along the list. For instance, the cost of traversal to the center of the list is $n/2$ which is larger than the actual cost of the binary search in an array as discussed in the previous point. There are also two cases to consider: singly and doubly linked list. In a single linked list (with one pointer per node), it is not possible to go back from the center of the list if the value being searched is towards the front half of the array; you'd have to start from the the front of the list each time which adds to the complexity. For a doubly linked list (with two pointers per node), it is easier to move from the center position but the complexity in the optimal case is still $O(n)$ which is larger than the actual binary search time complexity. How is it $O(n)$? The cost of traversal to the center is $n/2$. If the element is in the next half or the previous half, you have to travel $n/4$ from this center. Following this, you will have to travel $n/8$ from the second center and so on. You can do that easily since you have pointers in both directions on each node. The total traversal cost then is:

$$\begin{aligned}\text{Cost} &= \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \frac{n}{16} + \dots \\ \text{Cost} &= n\Big(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \dots\Big) \\ \text{Cost} &= n\end{aligned}$$

   Since sum of the bracketed geometric progression above is 1. You still end up doing $O(\log n)$ comparisons as it is a binary search algorithm but the cost of traversal is the tight upper bound. In an array, the cost of accessing any element is $O(1)$.