Graded homework - Binary search

Team-Berger-Nussbaum

1) Comment the code on the previous page with your understanding of it.

Line	Code	Comment	
1)	int binarySearch(int left , int right , E target) {	Function declaration	
2)	int mid = (left + right)/2;	Compute middle index	
3)	int comparison = comp.compare(target, a.get(mid));	Compare target with element at middle index And save the comparison result. (-1 = target < elem@middle) (0 = target = elem@middle) (1 = target > elem@middle)	
4)	if (right == left)	Check if right index is the same as left index	
5)	return (comparison <=0) ? left : right+1;	When comparison I smaller or equal to 0 (target < elem@middle) return left index, otherwise return right index +1	Base case index overlap → Decide whether left or right is correct position
6)	else if (comparison == 0 || (comparison < 0 && left == mid))	When comparison is 0 (target = elem@middle) OR comparison is smaller than 0 (target < elem@middle) AND left index is the same as middle index	Base case (same element is found OR left index is middle index) → mid is the correct position
7)	return mid ;	Return middle index	
8)	else if (comparison < 0)	When comparison is smaller than 0 (target < elem@middle)	
9)	return binarySearch (left , mid -1, target);	Search in left part	Recursion in left part
10)	else		
11)	return binarySearch (mid +1, right , target);	Search in right part	Recursion in right part
12)	}		

Graded homework — Binary Search

Team - Berger - Nussbaum

2) Worst-time complexity of binary search

Worst case → Wenn das element im letzten Split gefunden wird.

$$T_A(n) \begin{cases} \Theta(1) & n \leq 1 \\ 1\, T_A(n/2) + \Theta(1) & \end{cases}$$

- Nur 1 Rekursion
- Problembereich wird halbiert
- Knotenarbeit

$a = 1$
$b = 2$
$c = 1$

$f(n) = \Theta(1) = \Theta(n^l \cdot (\log(n))^k)$
$\Rightarrow l = 0$
$k = 0$

$\Rightarrow \quad l = 0 \qquad \log_2(1)$
$\qquad\quad 0 = 0$

$\Rightarrow \Theta(f(n) \cdot \log(n))$
$\Rightarrow \Theta(1 \cdot \log(n))$
$\Rightarrow \Theta(\log(n))$

3) Die Zeitkomplexität steigt, wenn SinglyLinkedList verwendet werden.
→ Dies passiert deshalb, weil der Zugriff auf bestimmte Elemente (an bestimmten Positionen) nicht mehr konstant ist.
z.B. wenn die gewünschte Position = n/4 ist
→ Dann müssen vom 1. Element (n/4) - Schritte passieren um das Element zu betrachten

Bei DoublyLinked Lists ist das ganze gleich.
→ Bei DobblyLinkedLists können wir im Unterschied zur SinglyLinkedList auch von rechts her navigieren.
Bringt uns beim BinarySearch jedoch nicht viel, da das nächste gefragte Element wieder in der Mitte der beiden vorherigen Grenzen liegt.
→ Ob von links (Start) oder von rechts (Ende) navigiert wird, spielt keine Rolle.

4) Siehe README.MD & ../code ☺

5) Halbiere den Problembereich, bis das Problem trivial wird.