$\boxed{\text{Base Case:}}$ $n=1$. $\therefore$ left $=$ right $=0$. middle $=0$. If the list element is same as searchnum, middle $=0$ is returned (Correct). Otherwise, it's easy to see that binsearch returns -1.

$\boxed{\text{Inductive hypothesis:}}$ Let, $\forall K \leq n$, $K \in \mathbb{N}$, if K is the no. of elements in the list, then if searchnum is present in the list, the binsearch function returns a position pos s.t. list[pos] $=$ searchnum, else the binsearch function returns -1.

$\boxed{\text{Induction Step:}}$ Let, $K=n+1$, where K is the no. of elements in the list.
$\therefore$ left $=0$, right $=n$. In line 5, we compute middle $=\lfloor n/2 \rfloor$.

- If list[middle] $<$ searchnum, the COMPARE macro returns -1, and in line 7, we call binsearch function from the posns. middle$+1 = \lfloor n/2 \rfloor + 1$ to n.
- The no. of elements: $n - (\lfloor n/2 \rfloor + 1) + 1 = n - \lfloor n/2 \rfloor = \lceil n/2 \rceil \leq n$. We apply the inductive hypothesis to show that the binsearch function runs correctly.
- If list[middle] $=$ searchnum, we return middle
- Else, we again apply the inductive hypothesis similarly to show that the binsearch function runs correctly. $\square$

<u>Iterative implementation of binary search</u>

```
1. int binsearch(int list[], int searchnum, int left, int right)
2. {
3. int middle;
4. while(left<=right) {
5.   middle=(left+right)/2;
6.   switch(COMPARE(list[middle], searchnum)) {
7.     case -1: left= middle+1;
8.              break;
9.     case 0: return middle;
10.    case 1: right= middle-1;
11.  }
12. }
13. return -1;
14. }
```