1. Find $T(N)$ for the Binary Search Algorithm

```
class List {
    int size;
    int count;
    int* L;
    const int MsMin;
    int Index(int key);
public:
    …
```

Figure 1. **class** *List*.

Line	Code	Cost
1	**int** *List*::*Index*(**int** *key*)	0
2	{ **int** *hi*=*count*;	1
3	**int** *lo*=**1**;	1
4	**while** (*lo*<=*hi*) {	k
5	**int** *m*=(*lo*+*hi*)/**2**	3*k*
6	**if** (*key*==*L*[*m*]) **return** *m*;	*k*
7	**if** (*key*<*L*[*m*])	*k*
8	*hi*=*m*-**1**;	2*k*
9	**else**	0
10	*lo*=*m*+**1**;	2*k*
11	}	1
12	**return 0**;	0
13	}	0
Total		**8*k*+3**

1.1. Member *count* is the N in $T(N)$ because member *count* records the number of keys in the list.

1.2. Recall member *L* points to an array of keys having a sentinel in element zero. An element smaller than any key in the list is placed in element zero. The sentinel is not part of the list. The list is contained in elements zero through the element indexed by member *count*.

1.3. Line 1. The function prologue is discounted by convention.

1.4. Line 2. Assignment is given unit cost bringing the total for this line to 1. The value of integer parameter *N* is the number of integers over which the search is made.

1.5. Line 3. Assignment is given unit cost bringing the total for this line to 1.

1.6. Line 4. Comparison (<=) is given unit cost. The comparison is executed k+1 times. The value of the comparison is true k times and false one time. The closing curly brace on line 11 has no cost but is assigned a cost of one to account for the case when the comparison is false.

1.7. Line 5. Assignment, addition, and division are each given unit cost bringing the total for this line to 3 each time it is executed. Line 5 is executed as many times as the comparison on the previous line is true. Line 5 is executed k times.

1.8. Line 6. Comparison (==) is assigned unit cost. No cost is given to the **`return`** statement. The comparison is executed k times.

1.9. Line 7. Comparison (<) is assigned unit cost. The comparison is executed k times.

1.10. Line 8. Assignment and addition are each assigned unit cost bringing the total to 2 each time this line is executed. The more costly of the two alternatives to the test on line 7 is assumed to be executed for the purpose of finding an upper bound on the cost of this algorithm. Since both alternatives are equally costly either can be included in the total. Please note, however, that the cost of only one alternative can be included in the total.

1.11. Line 9. By convention, no cost is assigned to the reserve word **`else`**.

1.12. Line 10. Assignment and subtraction are each assigned unit cost bringing the total to 2 each time this line is executed. The more costly of the two alternatives to the test on line 7 is assumed to be executed for the purpose of finding an upper bound on the cost of this algorithm. Since both alternatives are equally costly either can be included in the total. Please note, however, that the cost of only one alternative can be included in the total.

1.13. Line 11. No cost is assigned to a line having only a curly brace. However, line 11 marks the end of the loop that began on line 4. The cost of comparison when the comparison is false is accounted here.

1.14. By convention, no cost is assigned to a **`return`** statement.

1.15. Computing *k*.

1.15.1. Recall that member *count* is equivalent to *N*.

1.15.2. In each iteration the value of *N* is halved. Call the remaining number of values to search $n_0, n_1, \cdots, n_k$ where $n_0 = N$.

1.15.3. A recurrence relation can be established for n_i. $n_{i+1} = \frac{n_i}{2}$

1.15.4. $n_1 = \frac{n_0}{2}, n_2 = \frac{n_1}{2}, \cdots, n_k = \frac{n_{k-1}}{2}$

1.15.5. We can solve the equations developed by the recurrence relation.

$$n_0 = N = \frac{N}{2^0}$$

$$n_1 = \frac{n_0}{2} = \frac{N}{2} = \frac{N}{2^1}$$

$$n_2 = \frac{n_1}{2} = \frac{N/2^1}{2} = \frac{N}{2^2}$$

$$\vdots$$

$$n_k = \frac{N}{2^k}$$

1.15.6. We know that in the worst case $n_k = 1$. In the worst case the algorithm will find a match when only one element remains.

1.15.7. $n_k = \frac{N}{2^k} = 1 \Rightarrow N = 2^k$

1.15.8. $k = \lceil \log_2 N \rceil$

1.16. $T(N) = 8k + 3 = 8\lceil \log_2 N \rceil + 3$