Saeleh Babapour (P.1)

```
int binarySearch(int[] arr, int key)
{
    int lo = 0;                          (1)
    int mid = arr.length - 1;            (3)
    int hi = arr.length - 1;             (3)
                                         (1)   } => 8
    while( lo <= hi)
    {
        mid = (lo + hi)/2;
        if (key < arr[mid])              (2)
            hi = mid - 1;                (2)
        else if (arr[mid] < key)         2
            lo = mid + 1;                (2)
    else
        return mid;                      (1)
}
return -1                                (1)
}
```

In each step, the array gets cut by half n/2 and key is searched inside the first or second half

It is log(n) because number of operation is less than (n) by 1/2

best case: O(1)

worst case: n/2 + (8+4) → O(log(n))

Part 2

Worst Case:

n : length of arrg

binary : divide by 2 until we get 1

$\frac{n}{2^x} = 1$ x: number of half-ing

$2^x = n$

$\log_2(2^x) = \log_2 n$

$x = \log_2 n$ → we need to divide by $\log(n)$ times until we have everything divided.

best case → $O(1)$

This happens when key is equal to arr[mid]

suppose we have an array of odd number of element

$2k+1 : k \in \mathbb{N}$

if key $= \left\lfloor \frac{2k+1}{2} \right\rfloor + 1 = mid$ → key found.

→ $O(1)$