Linear Search

Key = 2

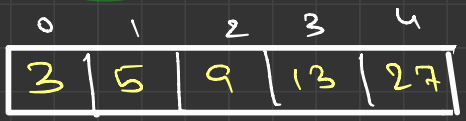
```
for (0 → Size of array)
{
    if (arr[i] == key)
    { return i; }
}
return -1;
```

Time complexity

O(n)

Binary Search

key = 13



* ensure it is sorted

Condition

Elements should be in monotonic function

Time complexity

$O(\log n)$

To optimize

$$mid = (start + end) / 2$$

max value of int = $2^{31} - 1$

According to this,

start = $2^{31} - 1$
end = $2^{31} - 1$

) + → out of int range

We can write this

$$mid = s + \left(\frac{e - s}{2}\right)$$

this will save us from long elements array

```
int binarySearch(int arr[], int size, int key)
{
    int start = 0;
    int end = size - 1;

    int mid = (start + end) / 2;

    while (start <= end)
    {
        if (arr[mid] == key)
        { return mid; }

        if (key > arr[mid])
        { start = mid + 1; }
        else
        { end = mid - 1; }

        mid = (start + end) / 2;
    }
    return -1;
}

int main(){
    int even[6] = { 2, 4, 5, 6, 9, 11 };
    int odd[5] = { 9, 12, 21, 35, 47 };

    int index = binarySearch(even, 6, 12);
    cout << " index of 12 is " << index;

    return 0;
}
```

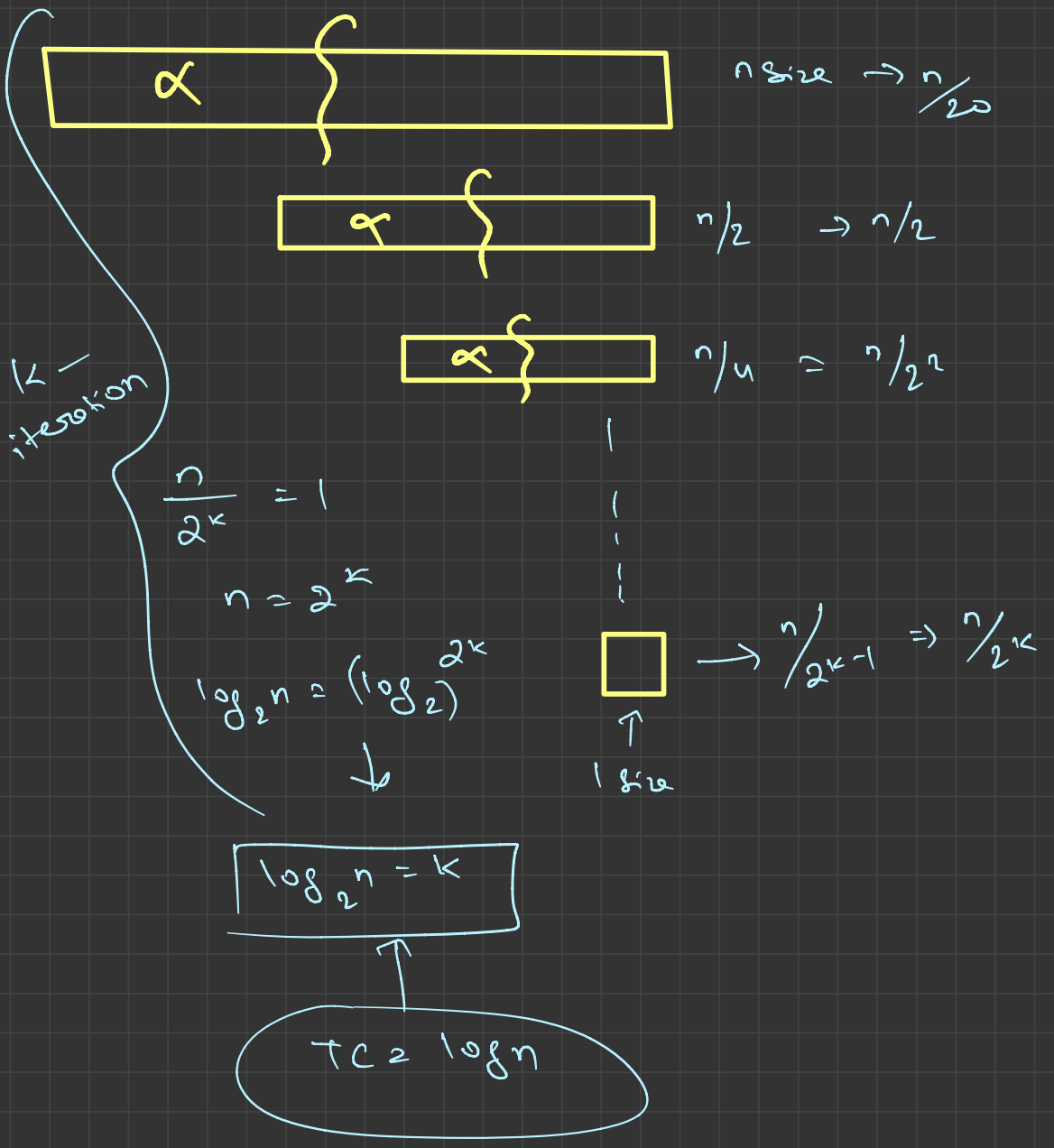
Check mid == key

Goes to right part

Goes to left part

New mid generate

Time Complexity



when we stuck into infine loop ?

So we have to make sure while we using such condition

$s < e$

while ($s <= e$)

If using this condition with the below change you goes into infinite loop

remove

$s = mid + 1$

change

$s = mid$

$e = mid - 1$

change

$e = mid$