Linear Search : Correctness

Let us rewrite the linear search algorithm of chapter 2 as follows:

input : Array: $A[1 \ldots n], Key$
output: index of first occurrence of key if it is found, 0 otherwise

```
i = 1
while i ≤ n do
    if A[i] = key then
        return i
    end
    else i++
end
return 0
```

Steps to Prove

- Loop Invariance
- Prove the Loop Invariance using Mathematical Induction
- Use the Loop Invariance to prove the correctness

Loop Invariance

```
i = 1
while i ≤ n do
    if A[i] = key then
        return i
    end
    else i++
end
return 0
```

Hypothesis $H(r)$: When the control reaches the "While" statement for the rth time,

1. $i = r$
2. key $\notin A[1 \ldots r-1]$

$\forall\ 1 \leq r \leq n+1$

Proof Of Loop Invariance

Hypothesis $H(r)$: When the control reaches the "While" statement for the rth time,

1. $i = r$
2. key $\notin A[1 \ldots r-1]$

Proof by induction on r.

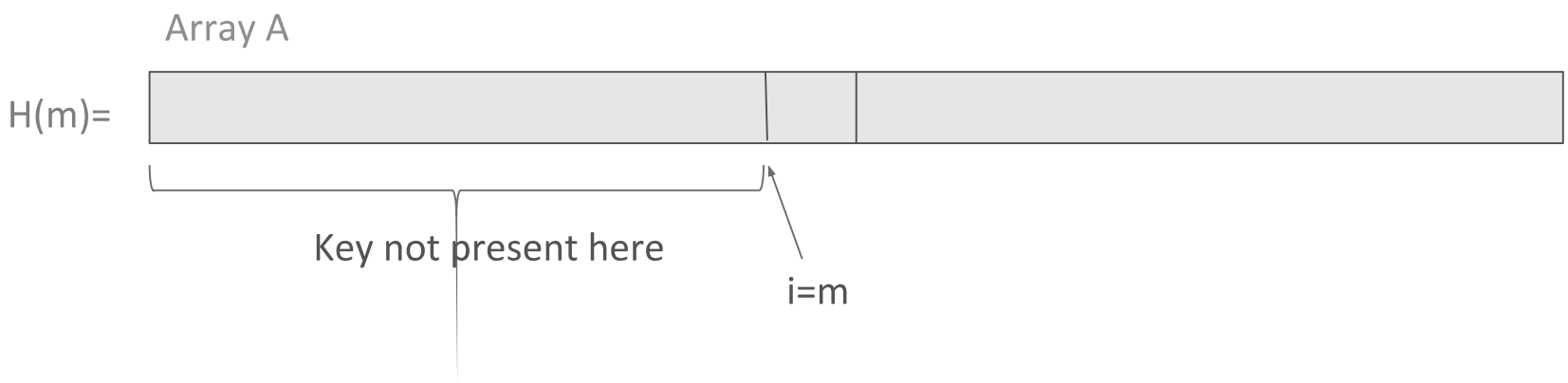
1. Base Case: when $r = 1$, the claim holds vacuously.
2. Induction Hypothesis: $H(m) \Rightarrow H(m+1)$.

Induction Hypothesis

```
i = 1
while i ≤ n do
    if A[i] = key then
        return i
    end
    else i++
end
return 0
```

Hypothesis $H(r)$: When the control reaches the "While" statement for the rth time,

1. $i = r$
2. key $\notin A[1 \ldots r-1]$

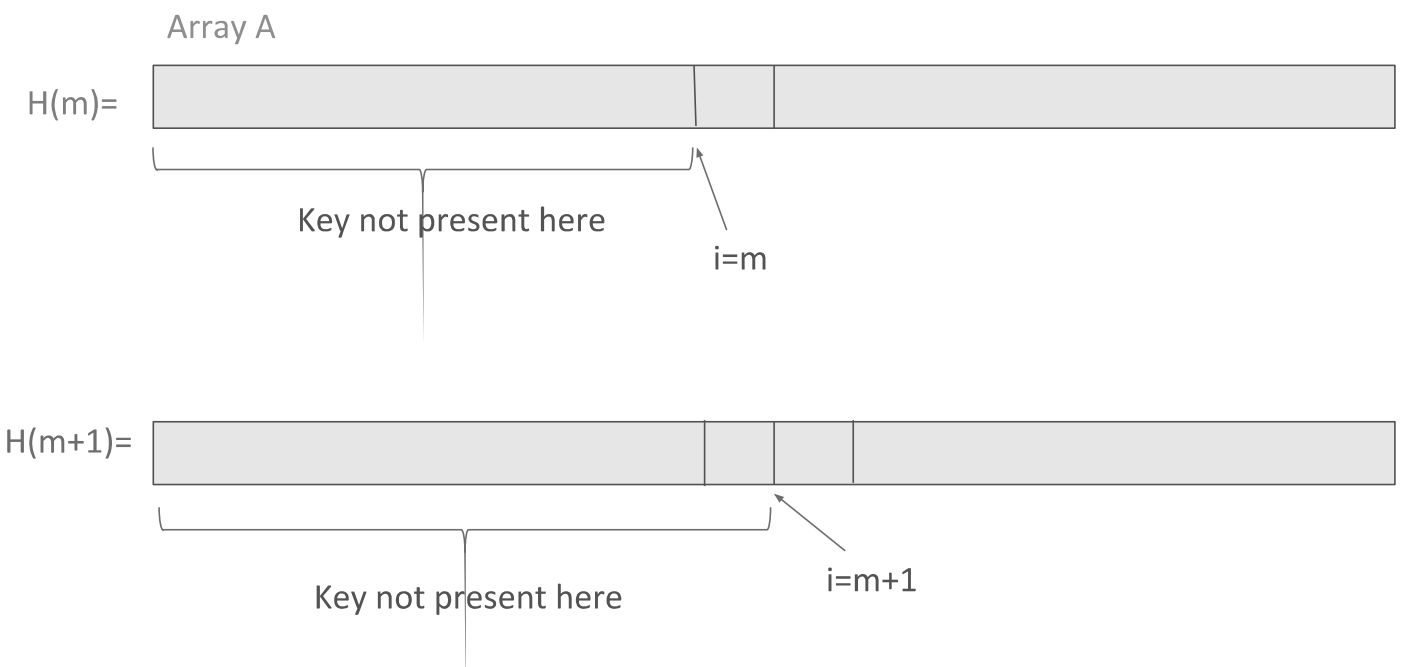


Induction Hypothesis

```
i = 1
while i ≤ n do
    if A[i] = key then
        return i
    end
    else i++
end
return 0
```

Hypothesis $H(r)$: When the control reaches the "While" statement for the rth time,

1. $i = r$
2. key $\notin A[1 \ldots r-1]$

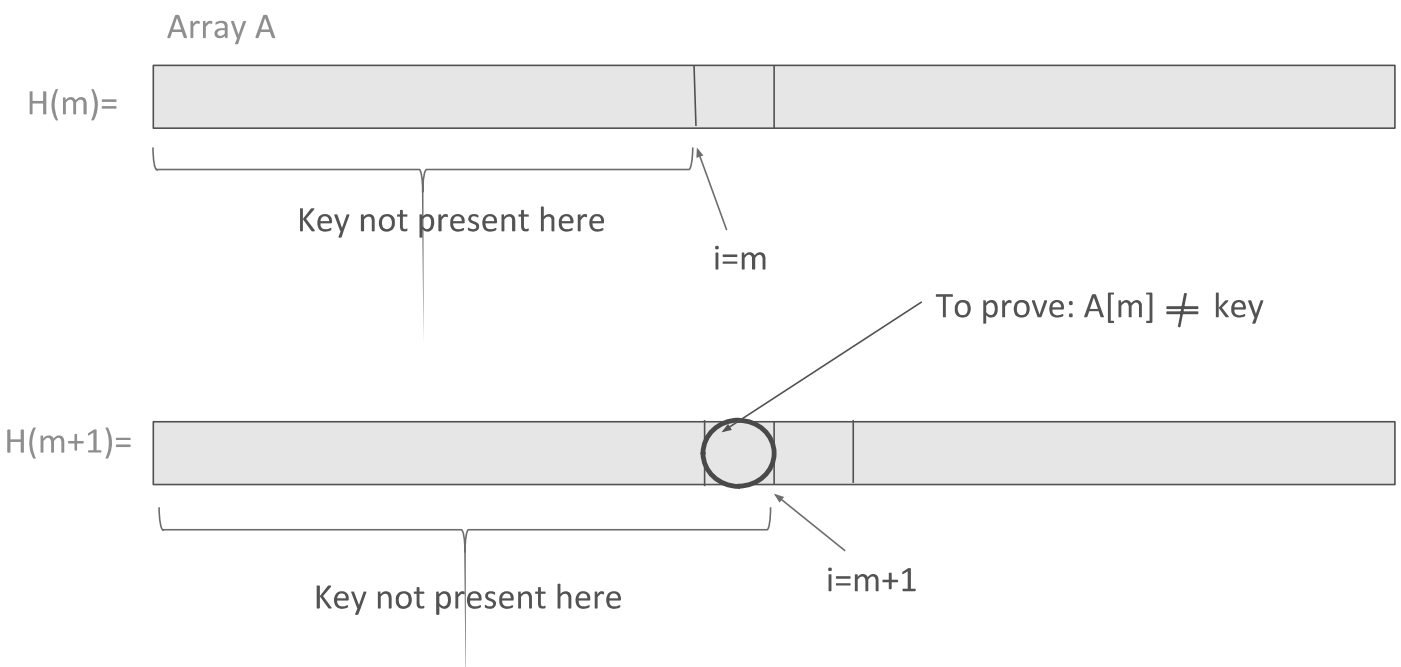


Induction Hypothesis

```
i = 1
while i ≤ n do
    if A[i] = key then
        return i
    end
    else i++
end
return 0
```

Hypothesis $H(r)$: When the control reaches the "While" statement for the rth time,

1. $i = r$
2. key $\notin A[1 \ldots r-1]$



Proof of Correctness

```
i = 1
while i ≤ n do
    if A[i] = key then
        return i
    end
    else i++
end
return 0
```

Hypothesis $H(r)$: When the control reaches the "While" statement for the rth time,

1. $i = r$
2. key $\notin A[1 \ldots r-1]$

Suppose that the test condition in "While" statement is executed exactly k times. i.e. body of the loop is executed $k-1$ times.

1. Case 1: $k \leq n$. In this case, $i = k$ and $A[1 \ldots k-1]$ does not contain the key. Also, the while loop terminated because $A[i](i.eA[k]) = key$. Thus the value returned in statement 4 is the position of the first occurrence of the key in the array.
2. Case 2: $k = n+1$. By loop invariant hypothesis, $i = n+1$ and $A[1 \ldots n]$ does not contain the key. Since $i = n+1$, control goes to statement 8 and the algorithm returns 0.

MCAC 301

Design and Analysis of Algorithms

Neelima Gupta

ngupta@cs.du.ac.in

PseudoCode
Eg. Linear Search

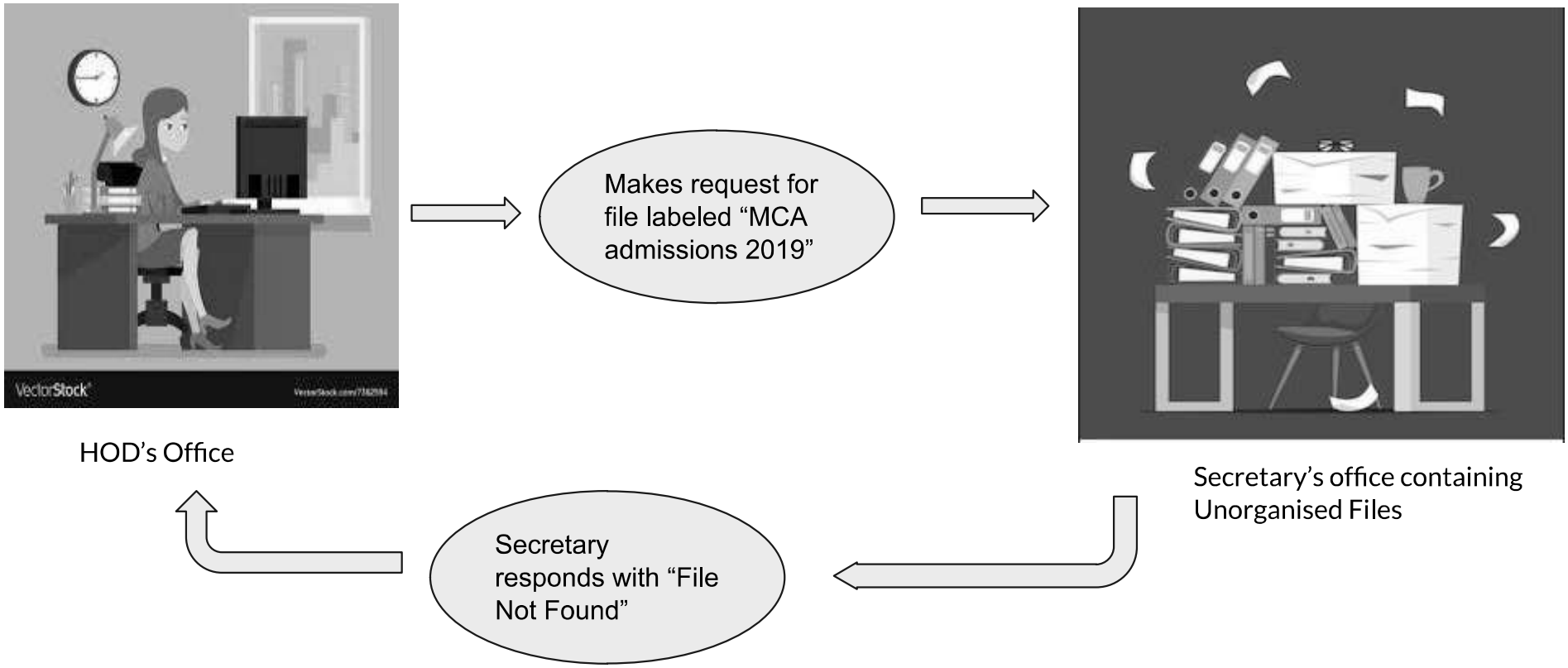


Figure A: Secretary at an office responding to the requests of the Head

- If there were n files and the requested file was not found, how many comparisons did the secretary have to perform?

Searching

Definition 1. ***Searching***: *Given a set of elements* $\{x_1, x_2 .... x_n\}$ *stored in an array and a target element* x*, problem is to find the location/index where* x *is found in the array or return* 0 *otherwise.*

Consider the following algorithm that examines each element sequentially to determine if it is x or not. It is called **Linear/Sequential Search**.

Algorithm 1: Linear search

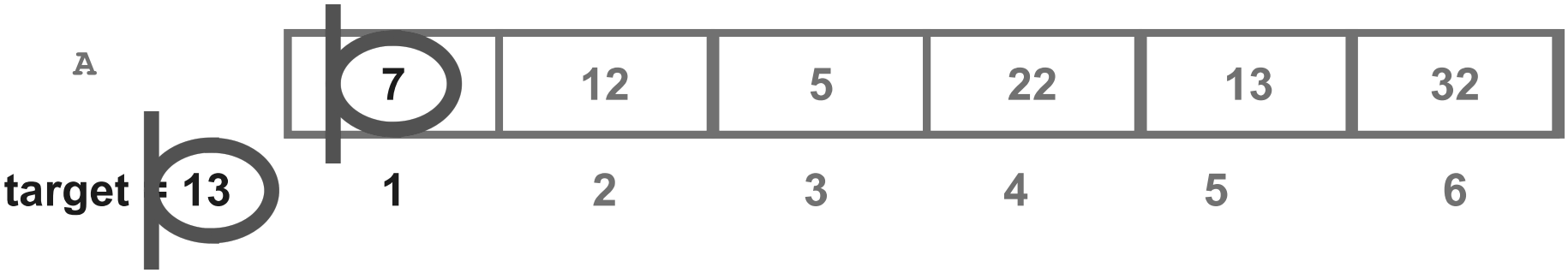
```
input  : Array: A[1...n], Key
output: index of first occurrence of key if it is found, 0 otherwise
for i ← 1 to n do
    if A[i] = key then
        return i
    end
end
return 0
```

Note: We use indices starting from 1 to n in algorithms (pseudocode) and not 0 to n-1

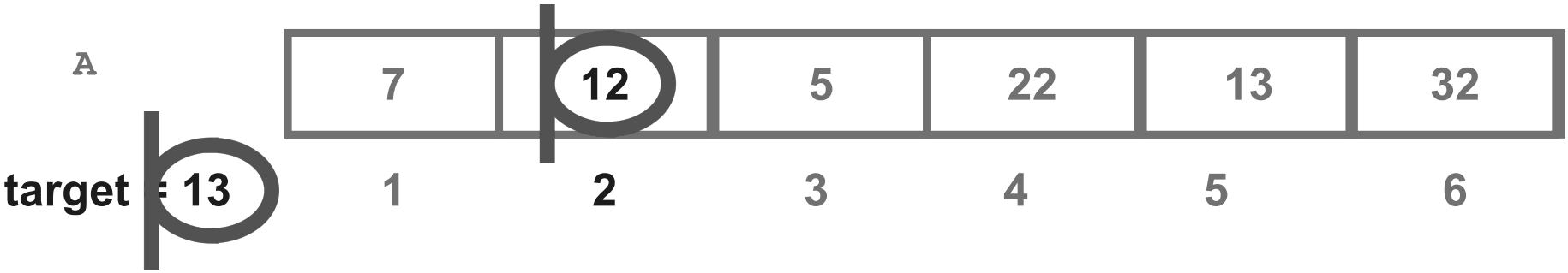
Example of Linear Search

A	7	12	5	22	13	32
target = 13	1	2	3	4	5	6

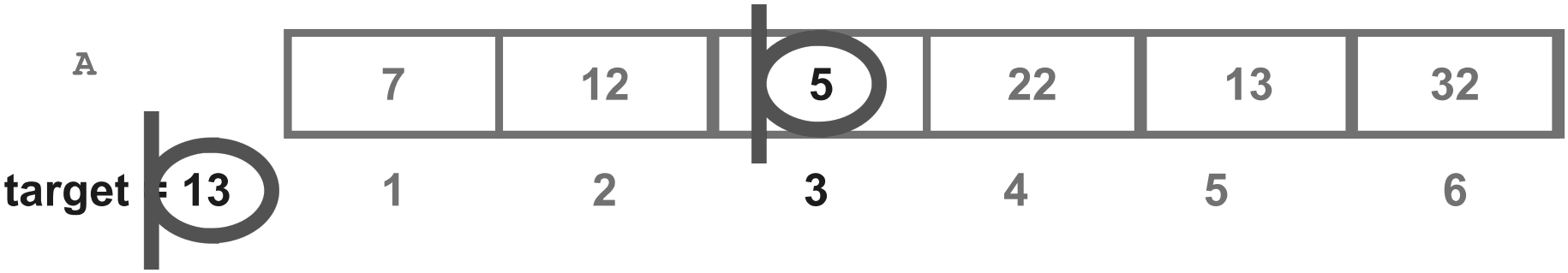
https://www.cc.gatech.edu/home/bleahy/cs1311/cs1311lecture15wdl.



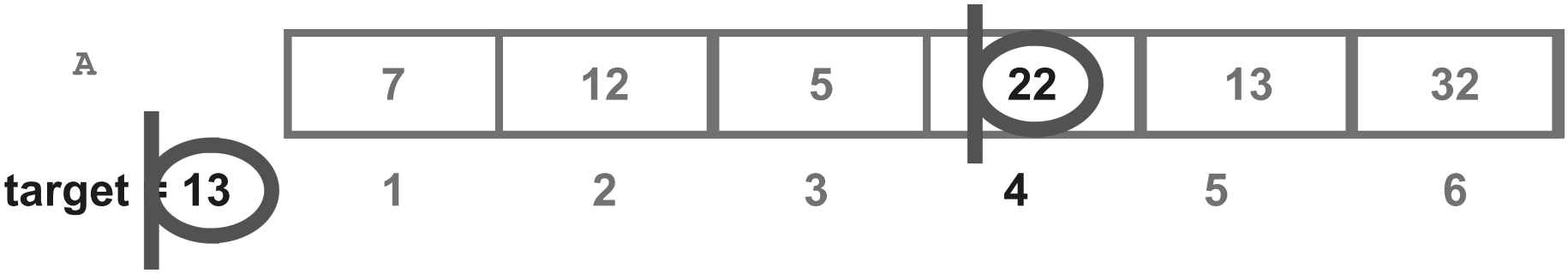
https://www.cc.gatech.edu/home/bleahy/cs1311/cs1311lecture15wdl.



https://www.cc.gatech.edu/home/bleahy/cs1311/cs1311lecture15wdl.

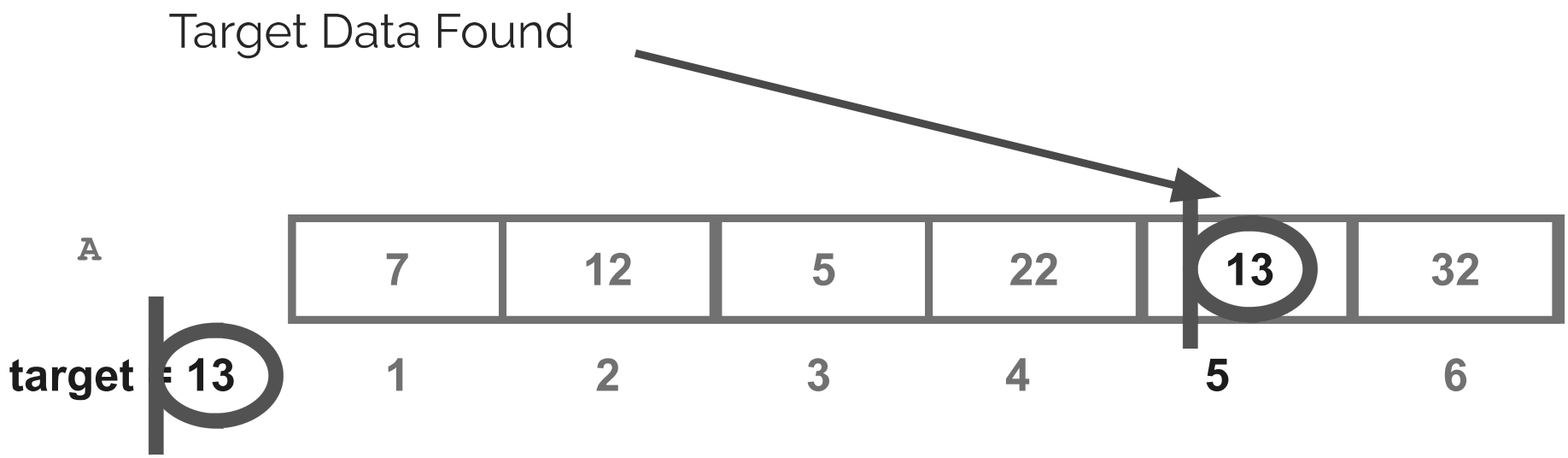


https://www.cc.gatech.edu/home/bleahy/cs1311/cs1311lecture15wdl.



https://www.cc.gatech.edu/home/bleahy/cs1311/cs1311lecture15wdl.

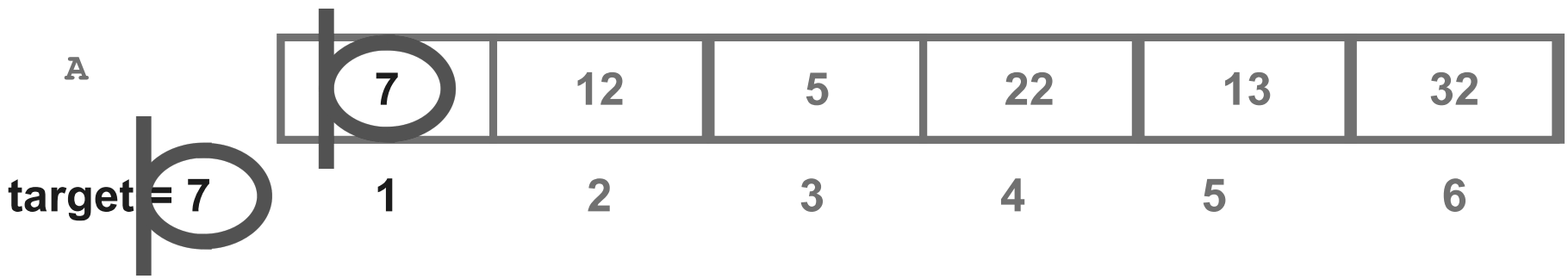
Target Data Found



https://www.cc.gatech.edu/home/bleahy/cs1311/cs1311lecture15wdl.

Linear Search Analysis: Best Case

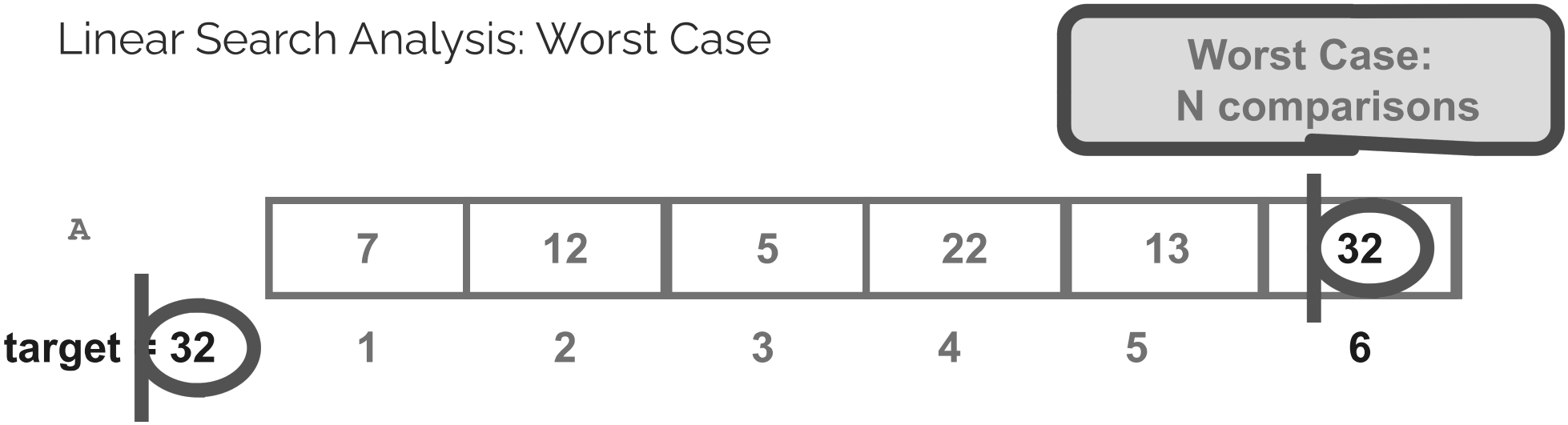
Best Case:
1 comparison



Best Case: **match with the first item**

https://www.cc.gatech.edu/home/bleahy/cs1311/cs1311lecture15wdl.

Linear Search Analysis: Worst Case

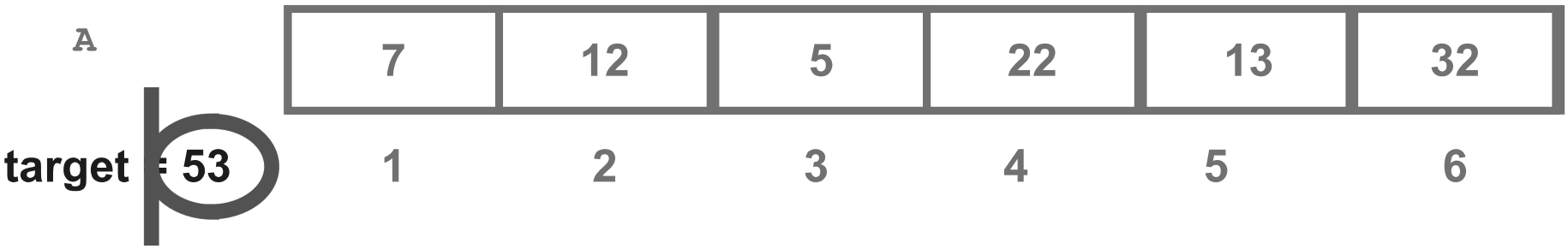


Worst Case: **match with the last item**

https://www.cc.gatech.edu/home/bleahy/cs1311/cs1311lecture15wdl.

Linear Search Analysis: Unsuccessful Search Case

Unsuccessful Search: N comparisons



Unsuccessful Search: no match

https://www.cc.gatech.edu/home/bleahy/cs1311/cs1311lecture15wdl.

Analysis of Linear Search : Counting the number of primitive steps

Algorithm 1: Linear search

```
input  : Array: A[1...n], Key
output: index of first occurrence of key if it is found, 0 otherwise
for i ← 1 to n do
    if A[i] = key then
        return i
    end
end
return 0
```

T1

T2

1. What is the relationship between T1 and T2?
   Ans: T1 = T2 for a successful search, T1 = T2 + 1 = n + 1 for an unsuccessful search
2. In the worst case successful search: T1 is ? T2 is?
   Ans: T1 = T2 =n
3. In the best case successful search: T1 is ? T2 is?
   Ans: T1 = T2 =1
4. In case of an unsuccessful search: T1 is ? T2 is?
   Ans: T1 = T2 + 1 = n + 1

Analysis of Linear Search : Counting the number of primitive steps

Total Number of Primitive Steps: (counting “assignment and comparison” done in statement 1 as one primitive step)

- T1+ T2 + 1(for the “return” statement)

this is =

- n + n + 1 = 2n + 1 in the worst case successful search
- 1 + 1 + 1 = 3 in the best case successful search
- (n + 1) + n + 1 = 2n + 2 in an unsuccessful search

Ignoring the constants

When we compare running times of two algorithms, constants become insignificant for large n.

For example

$\lim_{n \to \infty} \frac{(c1 \log n + c2)}{c3\, n + c4} = 0$, irrespective of the constants c1, c2, c3 and c4.

We will define the notion of asymptotic functions to capture this.

Programming Assignment:

- Implement Linear Search
  - Count the number of key comparisons for various inputs and plot the graph.
  - For every input size n , run it with n+ 1 different keys - n successful plus 1 unsuccessful
  - Compute the minimum, maximum and average of the number of key comparisons for each input size.
  - Plot the graph for each case - best, worst and average number of comparisons.
  - n varies from 10 to 100 in steps of 5

MCAC 301: Design and Analysis of Algorithms

Neelima Gupta

ngupta@cs.du.ac.in

April 27, 2023

Can we search faster?

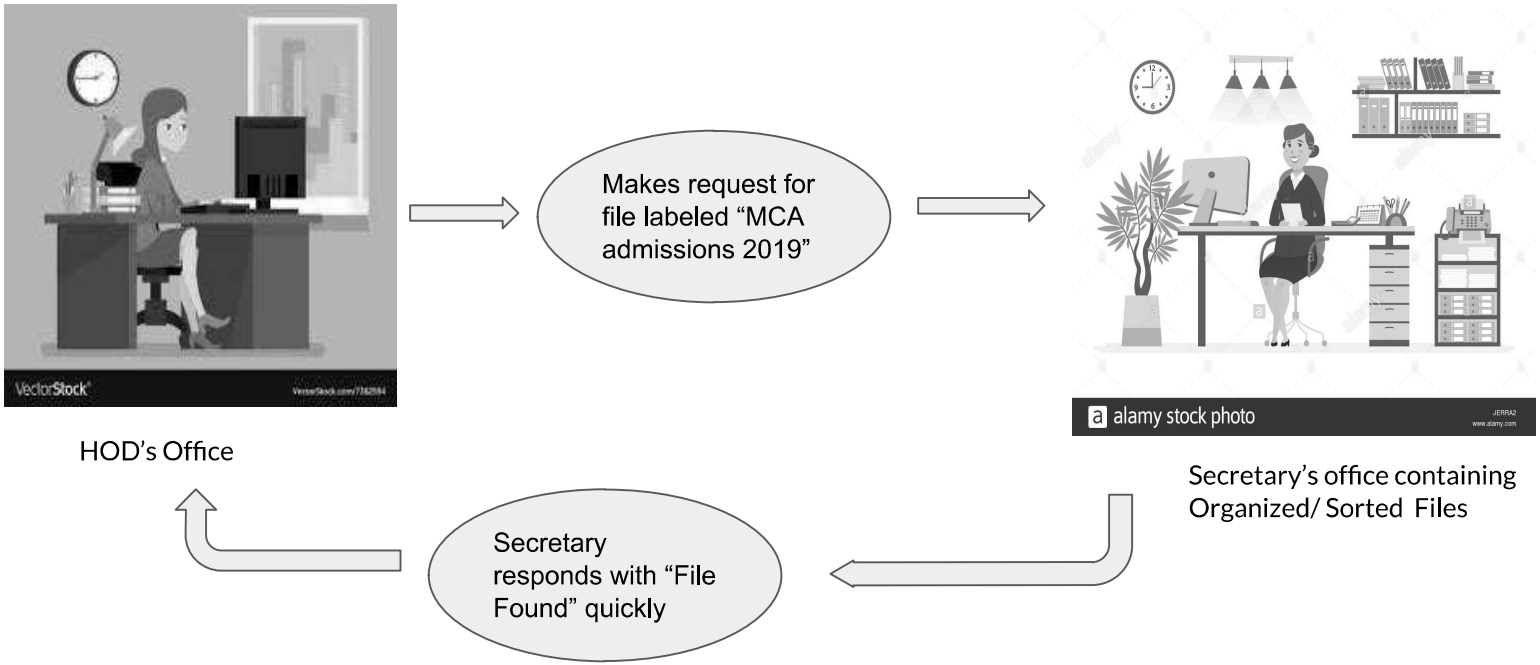


Figure A: Secretary at an office responding to the requests of the Head

1

Searching for a word in a dictionary

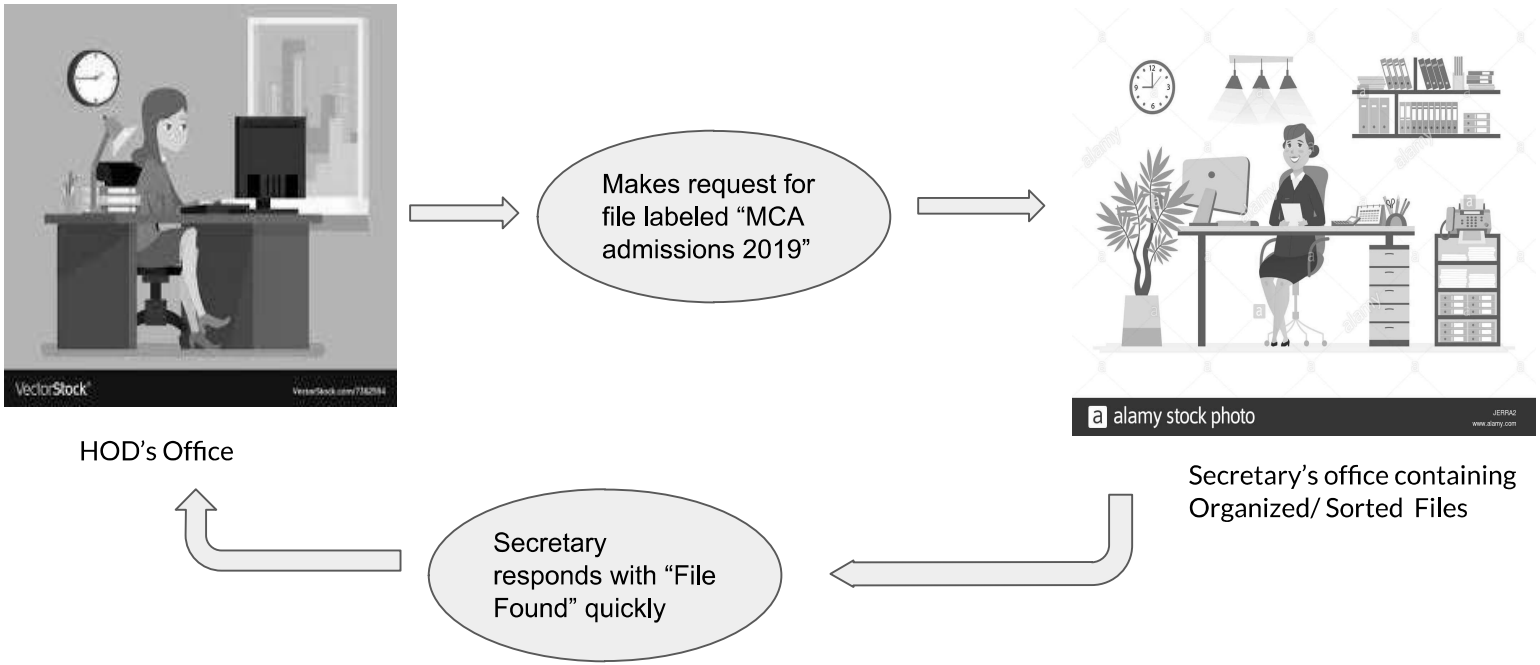
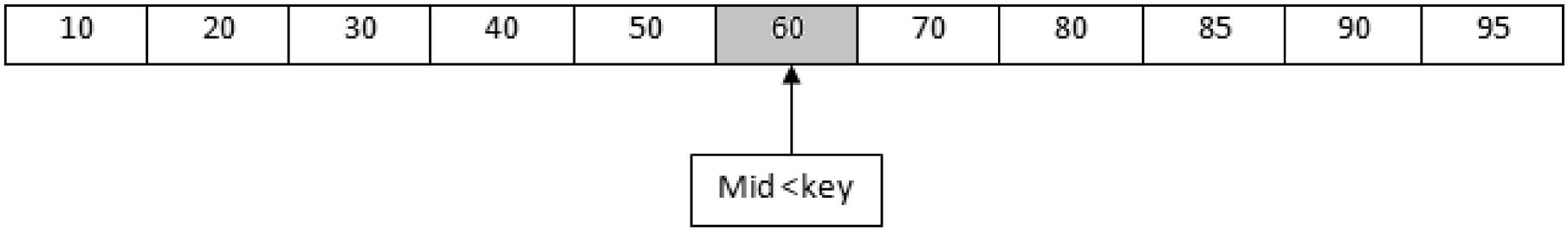


Figure A: Secretary at an office responding to the requests of the Head

1

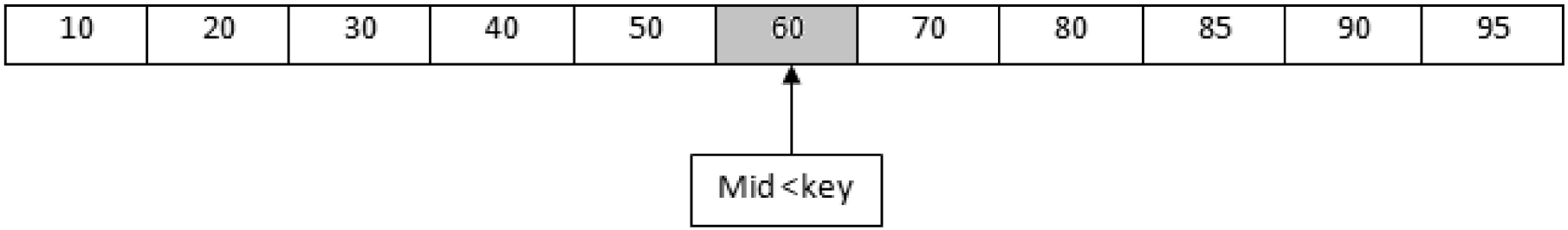
Binary Search

Key: 65



Binary Search

Key: 65



Divide and Conquer Paradigm

1. Divide : Divide the problem into a number of subproblems of smaller Size.
2. Conquer : Solve each sub-problem.
3. Combine : Combine the solution of the subproblems to obtain the solution of the original (bigger) problem.

A subproblem is solved recursively (i.e. by applying the same algorithm on the smaller problem) so long as it's size $>$ some threshold (called the "terminating condition"). Thereafter, it is solved directly.

Binary Search: Split in the middle

input : Array: $A[1 \ldots n], Key$
output: Index of key if key found, -1 otherwise

```
Binary-Search-Recursive(A, first, last, key)
/* "first" and "last" are the first and the last indices of the
  array
if first ≤ last then
    mid = (first + last)/2
    if A[mid]=key then
        return mid
    end
    if A[mid] < key then
        Binary-Search-Recursive(A, mid + 1, last, key)
    end
    else
        Binary-Search-Recursive(A, first, mid -1, key)
    end
end
```

Frame Title

Quick Demo

Time Complexity

Let $W(n)$ be the number of comparisons performed by the binary search algorithm in worst case. Then,
$W(n) = ?$

Time Complexity

Let $W(n)$ be the number of comparisons performed by the binary search algorithm in worst case. Then,
$W(n) = ?$
$W(n) = W(n/2) + ?$

Time Complexity

Let $W(n)$ be the number of comparisons performed by the binary search algorithm in worst case. Then,
$W(n) = ?$
$W(n) = W(n/2) + ?$
$W(n) = W(n/2) + 2$

Time Complexity

Let $W(n)$ be the number of comparisons performed by the binary search algorithm in worst case. Then,
$W(n) = ?$
$W(n) = W(n/2) + ?$
$W(n) = W(n/2) + 2$
$W(n) = ?$

Time Complexity

Let $W(n)$ be the number of comparisons performed by the binary search algorithm in worst case. Then,
$W(n) =?$
$W(n) = W(n/2)+?$
$W(n) = W(n/2) + 2$
$W(n) =?$ $W(n) = \log n$
Let $B(n)$ be the number of comparisons performed by the binary search algorithm in best case. Then,

Time Complexity

Let $W(n)$ be the number of comparisons performed by the binary search algorithm in worst case. Then,
$W(n) = ?$
$W(n) = W(n/2) + ?$
$W(n) = W(n/2) + 2$
$W(n) = ?$ $\;\; W(n) = \log n$
Let $B(n)$ be the number of comparisons performed by the binary search algorithm in best case. Then,
$B(n) = ?$

Time Complexity

Let $W(n)$ be the number of comparisons performed by the binary search algorithm in worst case. Then,
$W(n) = ?$
$W(n) = W(n/2) + ?$
$W(n) = W(n/2) + 2$
$W(n) = ?$ $\;\; W(n) = \log n$
Let $B(n)$ be the number of comparisons performed by the binary search algorithm in best case. Then,
$B(n) = ?$
$B(n) = 1$

MCSC 101: Design and Analysis of Algorithms

Neelima Gupta

ngupta@cs.du.ac.in

January 4, 2021

Binary Search

```
Binary-Search(A[], n, key)
left ← 1, right ← n
while left ≤ right do
    mid ← (left+right)/2
    if (A[mid]=key) then
        return mid
    end
    if (A[mid] < key) then
        left ← mid+1
    end
    else
        right ← mid-1
    end
end
return 0
```

Defining the I/O of Binary Search

- Input : Given an array L containing n items (n$\geq$ 0) ordered such that A[1] $\leq$ A[2] $\leq$ A[3] $\leq \ldots \leq$ A[n] and given x
- Output : The binary search algorithm terminates with index_mid = an occurrence of x in A, if found and, index_mid= 0 otherwise.

Binary search can only be applied if the array to be searched is already sorted

Search for the no. 15

Search for the no. 30

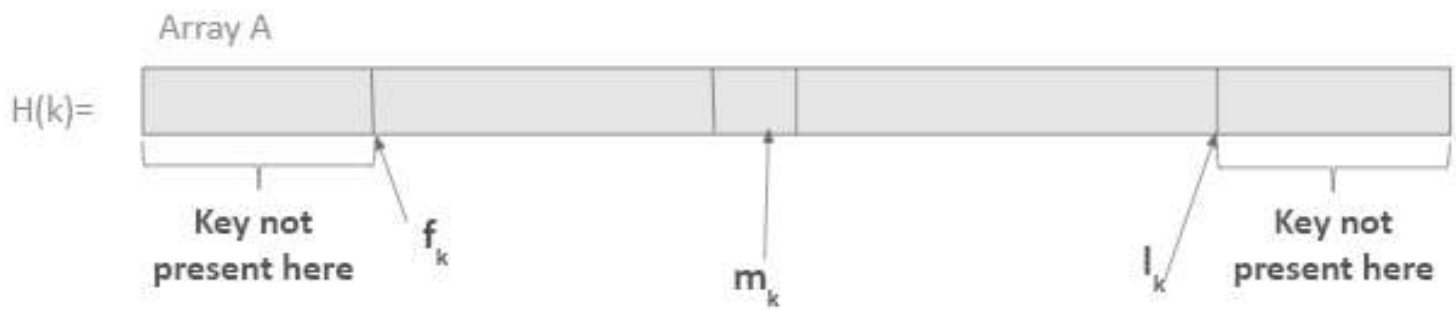
Proof of correctness: The Loop Invariant

Let r be the maximum number of times the loop starting from line 3 runs. For $1 \leq k \leq r$, let f_k and l_k be the values of first and last respectively when the control reaches the while statement for the kth time.

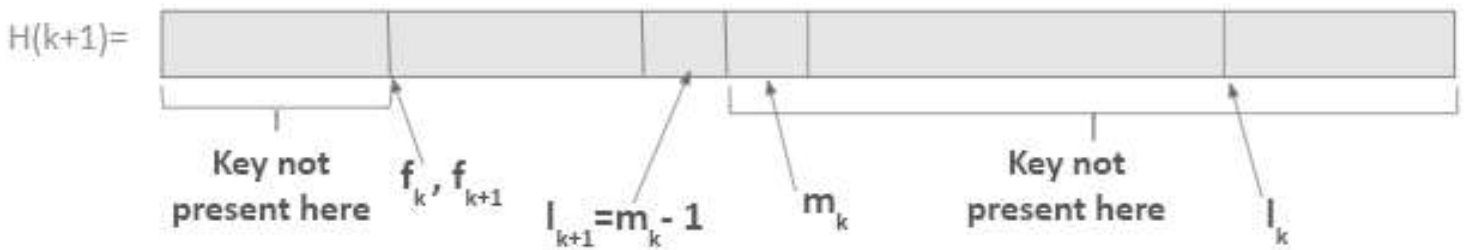
- Loop Invariant: For $1 \leq k \leq r$, $H(k)$: when the control reaches the while statement for the kth time $A[i] \neq x$ for every $i < f_k$ and for every $i > l_k$.
- We will prove the loop invariant by induction.
- Base Case: $k = 1$. $f_k = 1$ and $l_k = n$. Hence the claim is vacuously true.

Inductive Step: Suppose $H(k)$ is true. We will prove that $H(k+1)$ is true. In the kth iteration, we have three possibilities:

- Case 1: $x = A[m_k]$ . . . .not possible else we wouldn't enter the loop $(k+1)^{th}$ time.

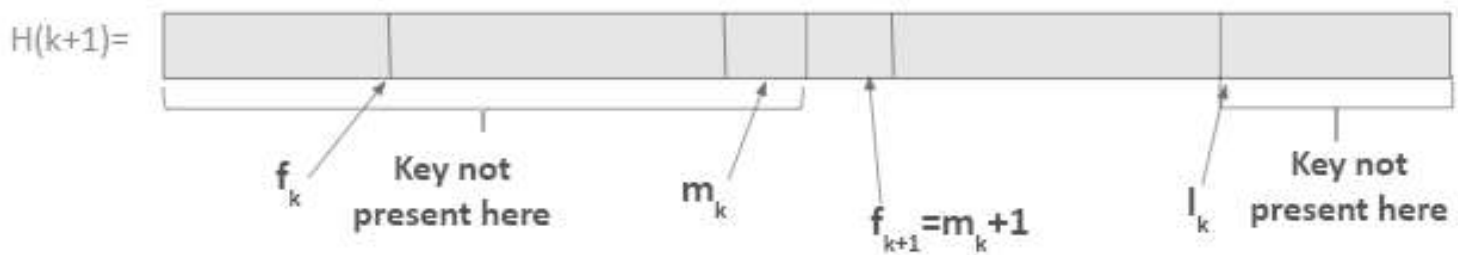


- Case 2. $x < A[m_k]$.



Since the array is sorted, we have
$x < A[m_k] \leq A[i]\ \forall i = m_k + 1 \ldots l_k$.
i.e. $x \neq A[i]\ \forall i = m_k \ldots l_k$.

- Case 3. $x > A[m_k]$.



Since the array is sorted, we have
$x > A[m_k] \geq A[i]\ \forall i = f_k \ldots m_k - 1$.
i.e. $x \neq A[i]\ \forall i = f_k \ldots m_k$.

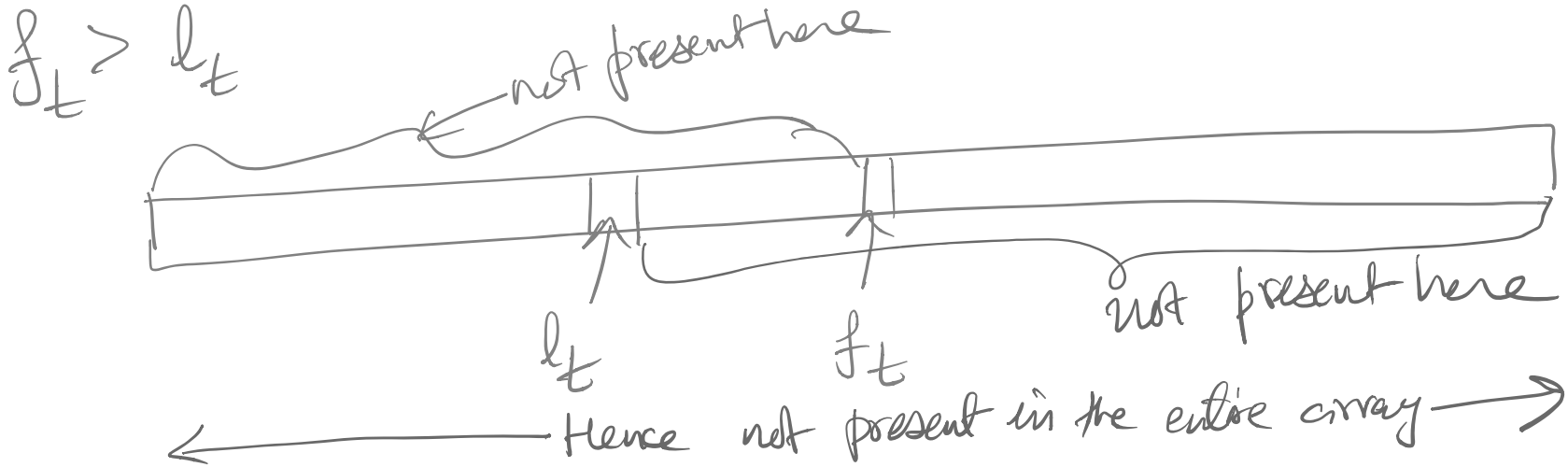
Correctness of the algorithm assuming the loop invariant

Suppose the test condition is executed t times.

- Case 1: $f_t \leq l_t$. We exit from the loop because $A[m_t] = x$. Thus, index_mid is a position of occurrence of x as index_mid $= m_t$
- Case 2: If $f_t > l_t$, we exit the while statement and return 0. We will next argue that the element is not present in the array in this case:
  By loop invariant hypothesis, $A[i] \neq x$ for $i < f_t$ and for $i > l_t$. the claim follows as $f_t > l_t$.

Hence assuming that the statement H(k) is correct the algorithm works correctly.

Correctness of the algorithm assuming the loop invariant



By loop invariant hypothesis, $A[i] \neq x$ for $i < f_t$ and for $i > l_t$. the claim follows as $f_t > l_t$.

Time Complexity

Let $W(n)$ be the number of comparisons performed by the binary search algorithm in worst case. Then,

Time Complexity

Let $W(n)$ be the number of comparisons performed by the binary search algorithm in worst case. Then,
$W(n) = ?$

Time Complexity

Let $W(n)$ be the number of comparisons performed by the binary search algorithm in worst case. Then,
$W(n) = ?$
$W(n) = W(n/2) + ?$

Time Complexity

Let $W(n)$ be the number of comparisons performed by the binary search algorithm in worst case. Then,
$W(n) = ?$
$W(n) = W(n/2) + ?$
$W(n) = W(n/2) + 2$

Time Complexity

Let $W(n)$ be the number of comparisons performed by the binary search algorithm in worst case. Then,
$W(n) = ?$
$W(n) = W(n/2) + ?$
$W(n) = W(n/2) + 2$
$W(n) = ?$

Time Complexity

Let $W(n)$ be the number of comparisons performed by the binary search algorithm in worst case. Then,
$W(n) =?$
$W(n) = W(n/2)+?$
$W(n) = W(n/2) + 2$
$W(n) =?$ $W(n) = \log n$
Let $B(n)$ be the number of comparisons performed by the binary search algorithm in best case. Then,

Time Complexity

Let $W(n)$ be the number of comparisons performed by the binary search algorithm in worst case. Then,
$W(n) = ?$
$W(n) = W(n/2) + ?$
$W(n) = W(n/2) + 2$
$W(n) = ?$ $W(n) = \log n$
Let $B(n)$ be the number of comparisons performed by the binary search algorithm in best case. Then,
$B(n) = ?$

Time Complexity

Let $W(n)$ be the number of comparisons performed by the binary search algorithm in worst case. Then,
$W(n) = ?$
$W(n) = W(n/2) + ?$
$W(n) = W(n/2) + 2$
$W(n) = ?$ $\;\; W(n) = \log n$
Let $B(n)$ be the number of comparisons performed by the binary search algorithm in best case. Then,
$B(n) = ?$
$B(n) = 1$