



```
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define MAXPAROLA 30
#define MAXRIGA 80

int main(int argc, char *argv[])
{
    int freq[MAXPAROLA]; /* vettore di contatori delle frequenze delle lunghezze delle parole */
    char riga[MAXRIGA];
    int i, inicio, lunghezza;
    FILE *f;

    for(i=0; i<MAXPAROLA; i++)
        freq[i]=0;

    if(argc != 2)
    {
        fprintf(stderr, "ERRORE: serve un parametro con il nome del file\n");
        exit(1);
    }
    f = fopen(argv[1], "r");
    if(f==NULL)
    {
        fprintf(stderr, "ERRORE: impossibile aprire il file %s\n", argv[1]);
        exit(1);
    }

    while( fgets( riga, MAXRIGA, f )!= NULL )
    {
        inicio = 0;
        lunghezza = strlen(riga);
        for(i=0; i<lunghezza; i++)
            if( !isalpha( riga[i] ) )
                inicio = i+1;
        if( inicio < lunghezza )
            riga[inicio] = '\0';
        lunghezza = strlen(riga);
        if( lunghezza > MAXRIGA )
            lunghezza = MAXRIGA;
        riga[lunghezza] = '\0';

        for(i=0; i<lunghezza; i++)
            if( isupper( riga[i] ) )
                riga[i] = tolower( riga[i] );
    }
    fclose(f);

    for(i=0; i<lunghezza; i++)
        freq[riga[i]]++;

    for(i=0; i<MAXPAROLA; i++)
        if(freq[i]>0)
            printf("%c\t%d\n", i, freq[i]);
}
```

## Algorithms and Complexity

### Search Algorithms

Paolo Camurati and Stefano Quer

Dipartimento di Automatica e Informatica  
Politecnico di Torino

## Search on Unordered Arrays

### ❖ Problem definition

- Given an array storing **n** integer values
- Given a specific value, i.e., a key **k**
- Reply to the following question
  - Is key k present in the array v ?

### ❖ Input

- $v[n]$ ,  $n$ ,  $k$

### ❖ Output

- Yes/No
- If Yes in which position (index) k is in the array

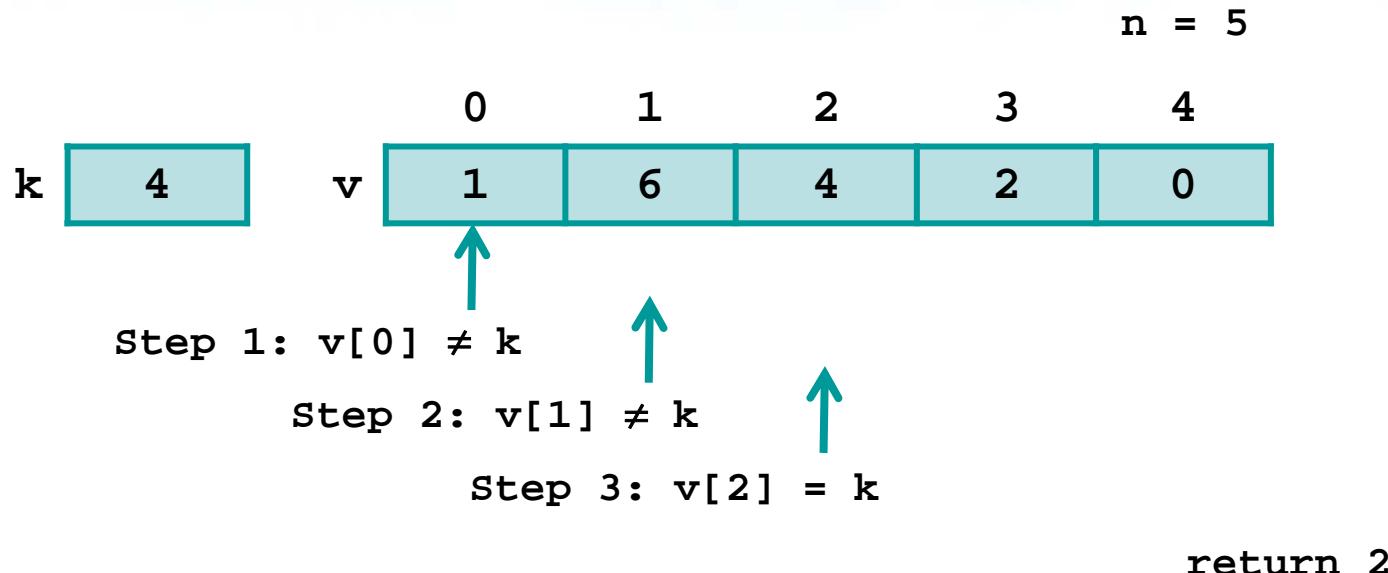
## Algorithm 1: Sequential Search

### ❖ Sequential search

- Scan the array from the first element to potentially the last one
- Compare key  $k$  and current value
  - Return index if the comparison is successful
- When all comparisons have been unsuccessful, i.e., at the end of the scan, return -1

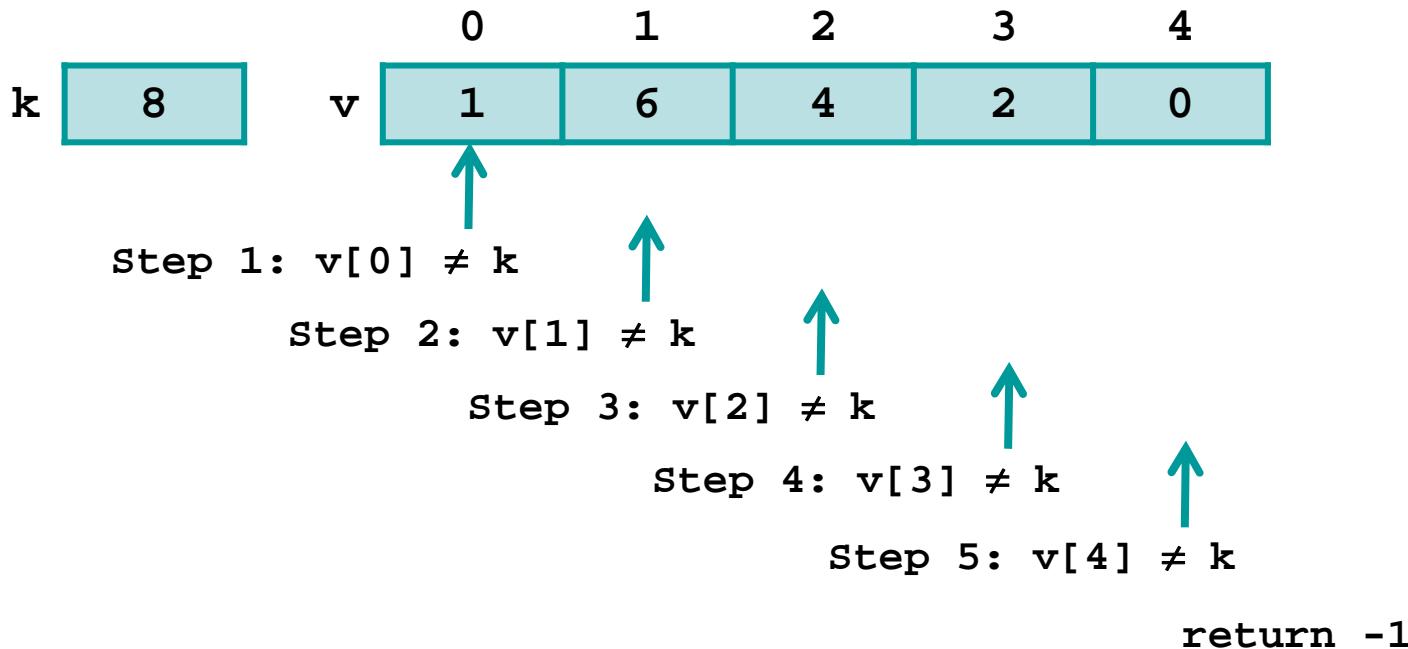
## Algorithm 1: Sequential Search

- ❖ Successful search



## Algorithm 1: Sequential Search

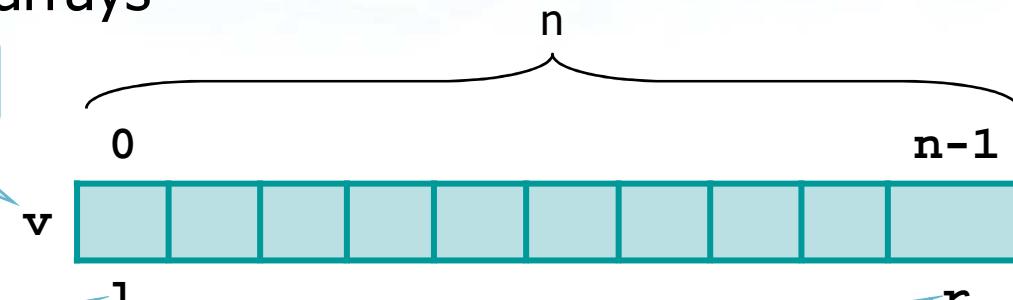
- ❖ Unsuccessful search



## Algorithm 1: Sequential Search

### ❖ Notation for arrays

Array  $v$  of  $n$  elements

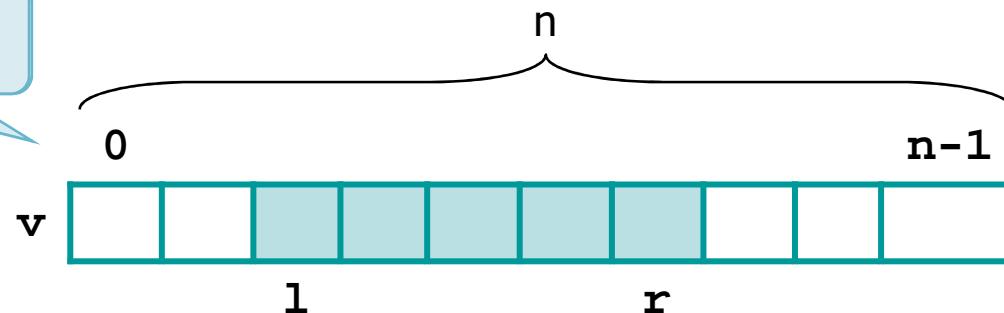


Leftmost element  
left or  $l$  index

Rightmost element  
right or  $r$  index

Entire array:  $l=0, r=n-1$   
Subarray:  $l, r$

We will use  
 $l$  and  $r$  in  
this section



## Algorithm 1: Sequential Search

Leftmost array index

Rightmost array index

```
int LinearSearch (int v[], int l, int r, int k) {  
    int i = l;  
    int found = 0;  
  
    while (i<=r && found==0) {  
        if (k == v[i]) {  
            found = 1;  
        } else {  
            i++;  
        }  
    }  
    if (found==0)  
        return -1;  
    else  
        return i;  
}
```

## Complexity Analysis

### ❖ Analytic analysis

```
int LinearSearch (...) {  
    int i = 1;  
    int found = 0;  
  
    while (i<=r && found==0) {  
        if (k == v[i]) {  
            found = 1;  
        } else {  
            i++;  
        }  
    }  
    if (found==0)  
        return -1;  
    else  
        return i;  
}
```

Number of operations  
(with unit cost)  
for the worst case

1  
1

$r-1+1+1$   
 $r-1+1$

$r-1+1$

1  
1

## Complexity Analysis

$$\begin{aligned}T(n) &= 1 + 1 + (r-l+1+1) + 2(r-l+1) + 1 + 1 \\&= 1 + 1 + (n + 1) + 2n + 1 + 1 \\&= 3n + 5 \\&= \Theta(n)\end{aligned}$$

➤  $T(n)$  grows linearly

Worst case  
 $O(n)$  overall

1  
1

$r-l+1+1$   
 $r-l+1$

$r-l+1$

1

1

## Complexity Analysis

### ❖ Intuitive analysis

- The worst case scenario is the one of an unsuccessful search
  - We have  $n$  steps for a search miss
  - In average  $n/2$  steps for a search hit

- $T(n)$  grows linearly with  $n$

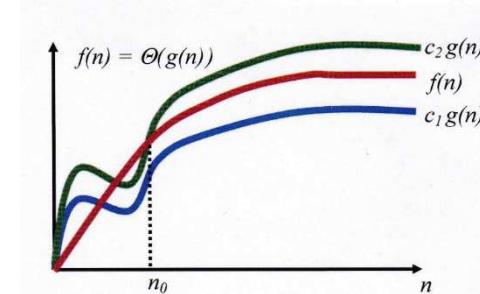
- $T(n) = \Theta(n)$

### ❖ Analytic analysis

- Worst case

- Unsuccessful search

- We assume unit cost for all operations



## Search on Ordered Arrays

### ❖ Problem definition

- Given an array storing **n** integer values **in ascending order**
- Given a specific value, i.e., a key **k**
- Reply to the following question
  - Is key k present in array v ?

### ❖ Input

- $v[n]$ ,  $n$ ,  $k$

### ❖ Output

- Yes/No
- If Yes in which position (index) k is in the array

## Algorithm 2: Sequential Search

Leftmost array index

Rightmost array index

```
int LinearSearch (int v[], int l, int r, int k) {  
    int i = l;  
  
    while (i<=r && k>v[i]) {  
        i++;  
    }  
    if (k == v[i]) {  
        return (i);  
    } else {  
        return (-1);  
    }  
}
```

If  $k < v[i]$  it is not  
possible to find  $k$  any  
more

## Complexity Analysis

- ❖ Has complexity been improved?
- ❖ Intuitive analysis
  - The worst case scenario is the one of an unsuccessful search with an element larger than all array values
    - We have  $n$  steps
  - $T(n)$  still grows linearly with  $n$ 
    - $T(n) = \Theta(n)$
- ❖ Can we do better?

## Algorithm 3: Binary Search

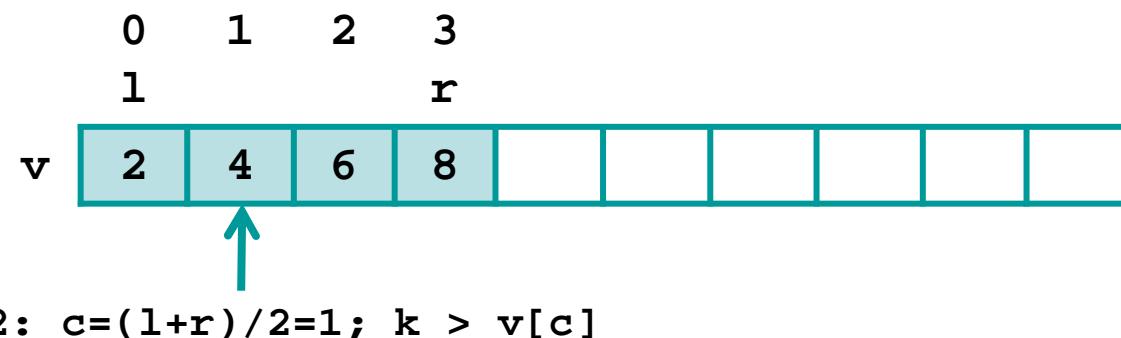
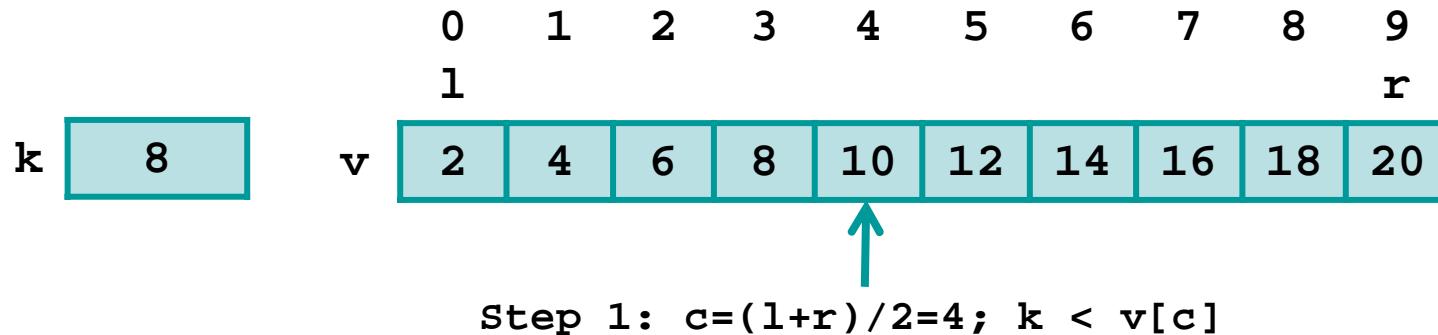
- ❖ Binary search in a sorted array
  - First version: 1946
  - First bug-free version: 1962
- ❖ Approach
  - Start with (sub-)array of extremes l and r
  - At each step
    - Find middle element  $c=(int)((l+r)/2)$
    - Compare k with middle element in the array
      - =: termination with success
      - <: search continues on left subarray
      - >: search continues on right subarray

Found bug in java  
Arrays.binarySearch():  
2006

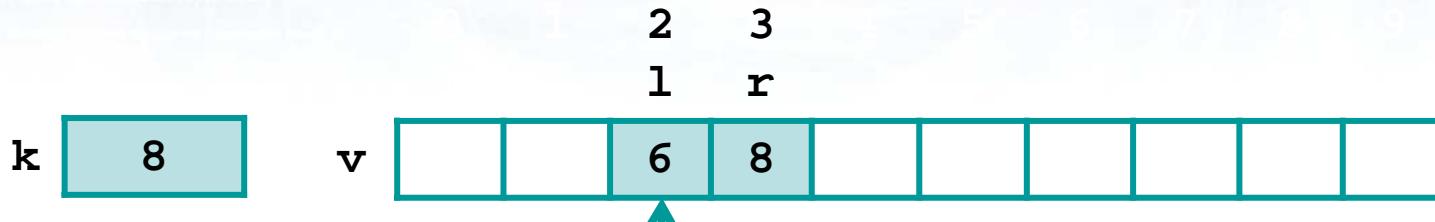
## Algorithm 3: Binary Search

- ❖ Successful search

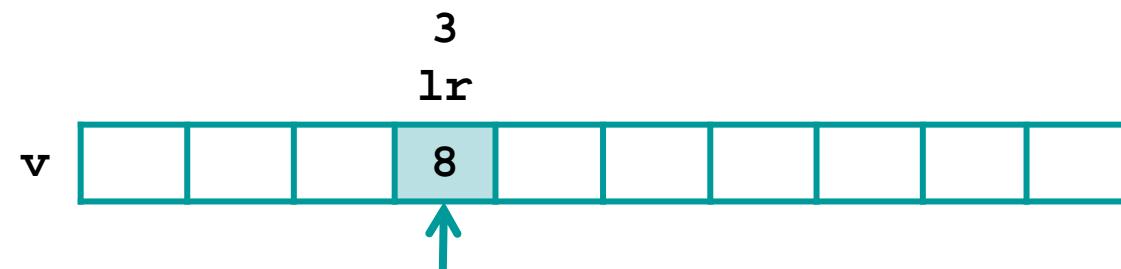
$l$  = leftmost array index  
 $r$  = rightmost array index  
 $c$  = index of middle element



## Algorithm 3: Binary Search



Step 3:  $c = (l+r)/2 = 2$ ;  $k > v[c]$

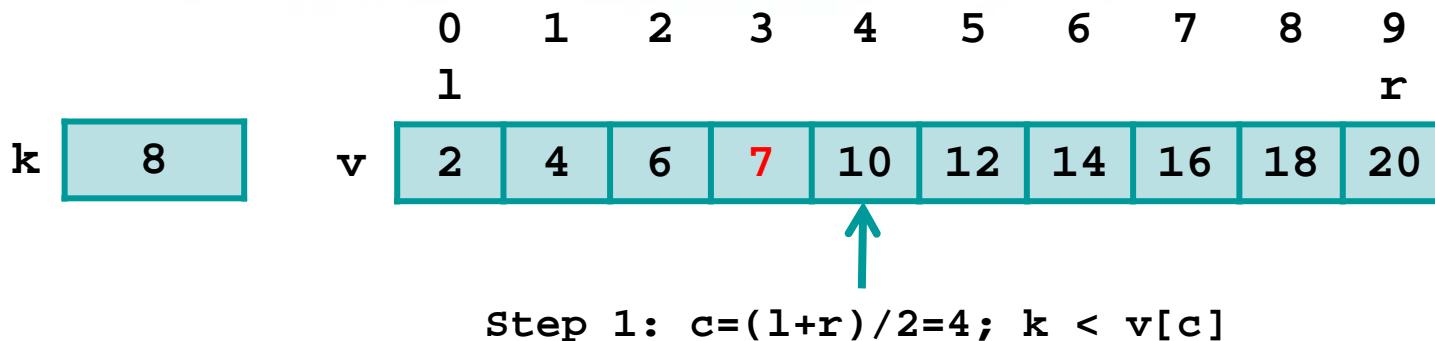


Step 4:  $c = (l+r)/2 = 3$ ;  $k = v[c]$

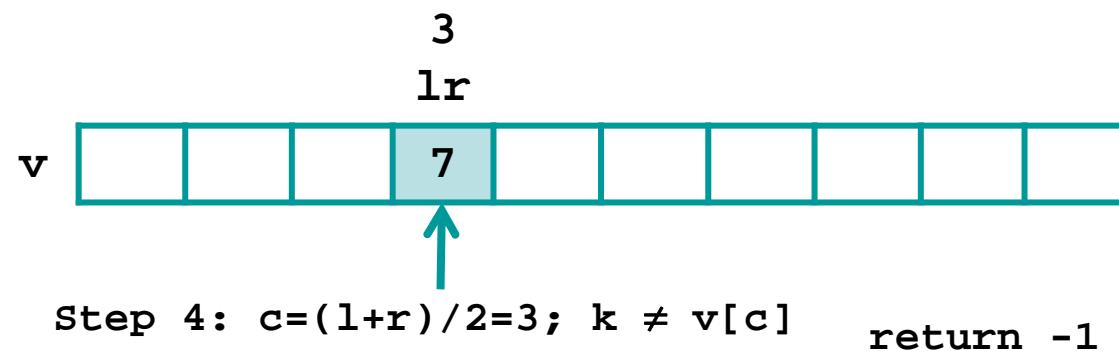
**return 3**

## Algorithm 3: Binary Search

### ❖ Unsuccessful search



... like previous case but last step ...



## Algorithm 3: Binary Search

Leftmost array index

Rightmost array index

```
int BinarySearch (int v[], int l, int r, int k) {  
    int c;  
  
    while (l<=r){  
        c = (int) ((l+r) / 2);  
        if (k == v[c]) {  
            return(c);  
        }  
        if (k < v[c]) {  
            r = c-1;  
        } else {  
            l = c+1;  
        }  
    }  
    return(-1);  
}
```

Or just  
 $c = (l+r)/2;$   
as c is an integer variable

## Complexity Analysis

### ❖ Analytic analysis

➤ The worst case scenario is the one of an unsuccessful search or a successful search with a hit at the final step

- We have  $n$  elements
- At each step, we divide  $n$  by 2
- In  $i$  steps, we divide  $n$  by  $2^i$ , i.e.,  $n/2^i$
- We stop after  $i$  steps, when
  - $n/2^i = 1$
  - $n = 2^i$
  - $i = \log_2 n$

➤  $T(n)$  grows logarithmically with  $n$

- $T(n) = O(\log_2 n)$

