# COMP108
# Data Structures and Algorithms

## Data structures - Arrays (Part II Binary Search)

Professor Prudence Wong

pwong@liverpool.ac.uk

2020-21

## Binary search

- more efficient way of searching when the sequence of numbers is pre-sorted
- Input: a sequence of $n$ sorted numbers $A[1], A[2], \cdots, A[n]$ in ascending order and a target number *key*
- Idea of algorithm:
  1. compare *key* with number in the middle
  2. then focus on only the first half or the second half (depend on whether *key* is smaller or greater than the middle number)
  3. reduce the amount of numbers to be searched by half

## Binary search - Example - To find 24

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 7 | 11 | 12 | **15** | 19 | 24 | 33 | 41 | 55 | ← 10 numbers |
| | | | | **24** | | | | | | ← key |
| | | | | | 19 | 24 | **33** | 41 | 55 | ← 5 numbers left |
| | | | | | | | **24** | | | ← key |
| | | | | | **19** | 24 | | | | ← 2 numbers left |
| | | | | | **24** | | | | | ← key |
| | | | | | | **24** | | | | ← 1 number left |
| | | | | | | **24** | | | | **FOUND!** |

## Binary search - Example 2 - To find 30

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 7 | 11 | 12 | **15** | 19 | 24 | 33 | 41 | 55 | ← 10 numbers |
| | | | | **30** | | | | | | ← key |
| | | | | | 19 | 24 | **33** | 41 | 55 | ← 5 numbers left |
| | | | | | | | **30** | | | ← key |
| | | | | | **19** | 24 | | | | ← 2 numbers left |
| | | | | | **30** | | | | | ← key |
| | | | | | | **24** | | | | ← 1 number left |
| | | | | | | **30** | | | | **NOT FOUND!** |

## Binary search - Pseudo code

```
first ← 1
last ← n
found ← false
while first ≤ last AND found == false do
begin

      // check with number in the middle

end
if found == true then
      output ``Found!''
else
      output ``Not found!''
```

$$A[first], \cdots, A[mid], \cdots, A[last]$$

$$\begin{array}{rl} & key = A[mid] \\ \longleftarrow & key < A[mid] \\ & key > A[mid] \longrightarrow \end{array}$$

## Binary search - Pseudo code

```
first ← 1
last ← n
found ← false
while first ≤ last AND found == false do
begin

    // check with number in the middle

end
if found == true then
    output ``Found!''
else
    output ``Not found!''
```

$\lfloor \rfloor$ is the floor function
truncates the decimal part

```
mid ← ⌊(first + last) / 2⌋
if key == A[mid] then
  found ← true
else
  if key < A[mid] then
    last ← mid − 1
  else
    first ← mid + 1
```

## Binary search - Pseudo code

```
first ← 1, last ← n, found ← false
while first ≤ last AND found == false do
begin
   mid ← ⌊(first+last)/2⌋
   if key == A[mid] then
      found ← true
   else if key < A[mid] then
         last ← mid − 1
      else first ← mid + 1
end
if found == true then
      output ``Found!''
else
      output ``Not found!''
```

Why *first* $\leq$ *last*?

- When there is one number left, both *first* and *last* (and *mid*) point to the same location
- If this number isn't the key, then either *first* becomes *mid* $+ 1$ or *last* becomes *mid* $- 1$.
- In both cases, *first* becomes larger than *last* and the while condition becomes false; hence the loop terminates.

## Binary search - Time complexity

Best case:

- *key* is the number in the middle

$\Rightarrow$ 1 comparison

$\Rightarrow$ $O(1)$

Worst case:

- at most $\lceil \log_2 n \rceil + 1$ comparisons

$\Rightarrow$ $O(\log n)$

Why?

Every comparison reduces the amount of numbers by at least half

E.g., $16 \Rightarrow 8 \Rightarrow 4 \Rightarrow 2 \Rightarrow 1$

```
first ← 1, last ← n, found ← false
while first ≤ last AND found == false do
begin
   mid ← ⌊(first+last)/2⌋
   if key == A[mid] then
      found ← true
   else if key < A[mid] then
         last ← mid − 1
      else first ← mid + 1
end
```

Summary: Arrays - binary search

Next: Arrays - finding minimum/maximum

## For note taking