

Peak Finding

COMS10017 - (Object-Oriented Programming and) Algorithms

Dr Christian Konrad

Let $A = a_0, a_1, \dots, a_{n-1}$ be an array of integers of length n

| | | | | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| a_0 | a_1 | a_2 | a_3 | a_4 | a_5 | a_6 | a_7 | a_8 | a_9 |

Let $A = a_0, a_1, \dots, a_{n-1}$ be an array of integers of length n

| | | | | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| a_0 | a_1 | a_2 | a_3 | a_4 | a_5 | a_6 | a_7 | a_8 | a_9 |

Definition: (Peak)

Integer a_i is a *peak* if adjacent integers are not larger than a_i

Let $A = a_0, a_1, \dots, a_{n-1}$ be an array of integers of length n

| | | | | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| a_0 | a_1 | a_2 | a_3 | a_4 | a_5 | a_6 | a_7 | a_8 | a_9 |

Definition: (Peak)

Integer a_i is a *peak* if adjacent integers are not larger than a_i

Example:

| | | | | | | | | | |
|---|---|---|----|----|---|---|---|---|---|
| 4 | 3 | 9 | 10 | 14 | 8 | 7 | 2 | 2 | 2 |
|---|---|---|----|----|---|---|---|---|---|

Let $A = a_0, a_1, \dots, a_{n-1}$ be an array of integers of length n

| | | | | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| a_0 | a_1 | a_2 | a_3 | a_4 | a_5 | a_6 | a_7 | a_8 | a_9 |

Definition: (Peak)

Integer a_i is a *peak* if adjacent integers are not larger than a_i

Example:

| | | | | | | | | | |
|---|---|---|----|----|---|---|---|---|---|
| 4 | 3 | 9 | 10 | 14 | 8 | 7 | 2 | 2 | 2 |
|---|---|---|----|----|---|---|---|---|---|

Peak Finding: Simple Algorithm

Problem PEAK FINDING: Write algorithm with properties:

- 1 **Input:** An integer array of length n
- 2 **Output:** A position $0 \leq i \leq n - 1$ such that a_i is a peak

Peak Finding: Simple Algorithm

Problem PEAK FINDING: Write algorithm with properties:

- 1 **Input:** An integer array of length n
- 2 **Output:** A position $0 \leq i \leq n - 1$ such that a_i is a peak

```
int peak(int *A, int len) {
    if(A[0] >= A[1])
        return 0;
    if(A[len-1] >= A[len-2])
        return len-1;

    for(int i=1; i < len-1; i=i+1) {
        if(A[i] >= A[i-1] && A[i] >= A[i+1])
            return i;
    }
    return -1;
}
```

C++ code

Peak Finding: Simple Algorithm

Problem PEAK FINDING: Write algorithm with properties:

- 1 **Input:** An integer array of length n
- 2 **Output:** A position $0 \leq i \leq n - 1$ such that a_i is a peak

```
Require: Integer array  $A$  of length  $n$   
  if  $A[0] \geq A[1]$  then  
    return 0  
  if  $A[n - 1] \geq A[n - 2]$  then  
    return  $n - 1$   
  for  $i = 1 \dots n - 2$  do  
    if  $A[i] \geq A[i - 1]$  and  $A[i] \geq A[i + 1]$  then  
      return  $i$   
  return  $-1$ 
```

Pseudo code

Peak Finding: Problem well-defined?

Is Peak Finding well defined? Does every array have a peak?

Peak Finding: Problem well-defined?

Is Peak Finding well defined? Does every array have a peak?

Lemma

Every integer array has at least one peak.

Peak Finding: Problem well-defined?

Is Peak Finding well defined? Does every array have a peak?

Lemma

Every integer array has at least one peak.

Proof.

Let A be an integer array of length n . Suppose for the sake of a contradiction that A does not have a peak. Then $a_1 > a_0$ since otherwise a_0 is a peak. But then $a_2 > a_1$ since otherwise a_1 is a peak. Continuing, for the same reason, $a_i > a_{i-1}$ since otherwise a_{i-1} is a peak, for every $i \leq n-1$. But this implies $a_{n-1} > a_{n-2}$ and hence a_{n-1} is a peak. A contradiction. Hence, every array has a peak. \square

| | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| a_0 | a_1 | a_2 | a_3 | a_4 | a_5 | a_6 |

Peak Finding: Problem well-defined?

Is Peak Finding well defined? Does every array have a peak?

Lemma

Every integer array has at least one peak.

Proof.

Let A be an integer array of length n . Suppose for the sake of a contradiction that A does not have a peak. Then $a_1 > a_0$ since otherwise a_0 is a peak. But then $a_2 > a_1$ since otherwise a_1 is a peak. Continuing, for the same reason, $a_i > a_{i-1}$ since otherwise a_{i-1} is a peak, for every $i \leq n-1$. But this implies $a_{n-1} > a_{n-2}$ and hence a_{n-1} is a peak. A contradiction. Hence, every array has a peak. \square

| | | | | | | |
|-------|---------|-------|-------|-------|-------|-------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| a_0 | $> a_0$ | a_2 | a_3 | a_4 | a_5 | a_6 |

Peak Finding: Problem well-defined?

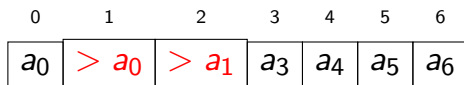
Is Peak Finding well defined? Does every array have a peak?

Lemma

Every integer array has at least one peak.

Proof.

Let A be an integer array of length n . Suppose for the sake of a contradiction that A does not have a peak. Then $a_1 > a_0$ since otherwise a_0 is a peak. But then $a_2 > a_1$ since otherwise a_1 is a peak. Continuing, for the same reason, $a_i > a_{i-1}$ since otherwise a_{i-1} is a peak, for every $i \leq n-1$. But this implies $a_{n-1} > a_{n-2}$ and hence a_{n-1} is a peak. A contradiction. Hence, every array has a peak. \square



Peak Finding: Problem well-defined?

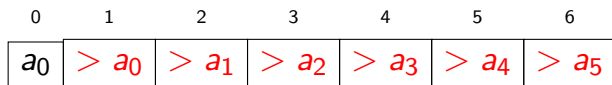
Is Peak Finding well defined? Does every array have a peak?

Lemma

Every integer array has at least one peak.

Proof.

Let A be an integer array of length n . Suppose for the sake of a contradiction that A does not have a peak. Then $a_1 > a_0$ since otherwise a_0 is a peak. But then $a_2 > a_1$ since otherwise a_1 is a peak. Continuing, for the same reason, $a_i > a_{i-1}$ since otherwise a_{i-1} is a peak, for every $i \leq n-1$. But this implies $a_{n-1} > a_{n-2}$ and hence a_{n-1} is a peak. A contradiction. Hence, every array has a peak. \square



Peak Finding: Problem well-defined?

Is Peak Finding well defined? Does every array have a peak?

Lemma

Every integer array has at least one peak.

Proof.

Every maximum is a peak. (Shorter and immediately convincing!)



Peak Finding: How fast is the Simple Algorithm?

How fast is our Algorithm?

```
Require: Integer array  $A$  of length  $n$   
if  $A[0] \geq A[1]$  then  
    return 0  
if  $A[n - 1] \geq A[n - 2]$  then  
    return  $n - 1$   
for  $i = 1 \dots n - 2$  do  
    if  $A[i] \geq A[i - 1]$  and  $A[i] \geq A[i + 1]$  then  
        return  $i$   
return  $-1$ 
```

How often do we look at the array elements? (worst case!)

- $A[0]$ and $A[n - 1]$:

Peak Finding: How fast is the Simple Algorithm?

How fast is our Algorithm?

```
Require: Integer array  $A$  of length  $n$   
if  $A[0] \geq A[1]$  then  
    return 0  
if  $A[n - 1] \geq A[n - 2]$  then  
    return  $n - 1$   
for  $i = 1 \dots n - 2$  do  
    if  $A[i] \geq A[i - 1]$  and  $A[i] \geq A[i + 1]$  then  
        return  $i$   
return  $-1$ 
```

How often do we look at the array elements? (worst case!)

- $A[0]$ and $A[n - 1]$: twice

Peak Finding: How fast is the Simple Algorithm?

How fast is our Algorithm?

```
Require: Integer array  $A$  of length  $n$   
if  $A[0] \geq A[1]$  then  
    return 0  
if  $A[n - 1] \geq A[n - 2]$  then  
    return  $n - 1$   
for  $i = 1 \dots n - 2$  do  
    if  $A[i] \geq A[i - 1]$  and  $A[i] \geq A[i + 1]$  then  
        return  $i$   
return  $-1$ 
```

How often do we look at the array elements? (worst case!)

- $A[0]$ and $A[n - 1]$: twice
- $A[1] \dots A[n - 2]$:

Peak Finding: How fast is the Simple Algorithm?

How fast is our Algorithm?

```
Require: Integer array  $A$  of length  $n$   
if  $A[0] \geq A[1]$  then  
    return 0  
if  $A[n - 1] \geq A[n - 2]$  then  
    return  $n - 1$   
for  $i = 1 \dots n - 2$  do  
    if  $A[i] \geq A[i - 1]$  and  $A[i] \geq A[i + 1]$  then  
        return  $i$   
return  $-1$ 
```

How often do we look at the array elements? (worst case!)

- $A[0]$ and $A[n - 1]$: twice
- $A[1] \dots A[n - 2]$: 4 times

Peak Finding: How fast is the Simple Algorithm?

How fast is our Algorithm?

```
Require: Integer array  $A$  of length  $n$   
if  $A[0] \geq A[1]$  then  
    return 0  
if  $A[n - 1] \geq A[n - 2]$  then  
    return  $n - 1$   
for  $i = 1 \dots n - 2$  do  
    if  $A[i] \geq A[i - 1]$  and  $A[i] \geq A[i + 1]$  then  
        return  $i$   
return  $-1$ 
```

How often do we look at the array elements? (worst case!)

- $A[0]$ and $A[n - 1]$: twice
- $A[1] \dots A[n - 2]$: 4 times
- Overall: $2 + 2 + (n - 2) \cdot 4 = 4(n - 1)$

Peak Finding: How fast is the Simple Algorithm?

How fast is our Algorithm?

```
Require: Integer array  $A$  of length  $n$   
if  $A[0] \geq A[1]$  then  
  return 0  
if  $A[n-1] \geq A[n-2]$  then  
  return  $n-1$   
for  $i = 1 \dots n-2$  do  
  if  $A[i] \geq A[i-1]$  and  $A[i] \geq A[i+1]$  then  
    return  $i$   
return  $-1$ 
```

How often do we look at the array elements? (worst case!)

- $A[0]$ and $A[n-1]$: twice
- $A[1] \dots A[n-2]$: 4 times
- Overall: $2 + 2 + (n-2) \cdot 4 = 4(n-1)$

Can we do better?!

Peak Finding: An even faster Algorithm

Finding Peaks even Faster: FAST-PEAK-FINDING

- 1 if A is of length 1 then return 0
- 2 if A is of length 2 then compare $A[0]$ and $A[1]$ and return position of larger element
- 3 if $A[\lfloor n/2 \rfloor]$ is a peak then return $\lfloor n/2 \rfloor$
- 4 Otherwise, if $A[\lfloor n/2 \rfloor - 1] \geq A[\lfloor n/2 \rfloor]$ then return FAST-PEAK-FINDING($A[0, \lfloor n/2 \rfloor - 1]$)
- 5 else return $\lfloor n/2 \rfloor + 1 +$
FAST-PEAK-FINDING($A[\lfloor n/2 \rfloor + 1, n - 1]$)

Comments:

- FAST-PEAK-FINDING is *recursive* (it calls itself)
- $\lfloor x \rfloor$ is the floor function ($\lceil x \rceil$: ceiling)

Peak Finding: Example

Example:

| | | | | | | | | | | | | | | | |
|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 3 | 7 | 22 | 47 | 36 | 33 | 31 | 30 | 25 | 21 | 20 | 15 | 7 | 4 | 10 | 22 |

Peak Finding: Example

Example:

| | | | | | | | | | | | | | | | |
|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 3 | 7 | 22 | 47 | 36 | 33 | 31 | 30 | 25 | 21 | 20 | 15 | 7 | 4 | 10 | 22 |

Check whether $A[\lfloor n/2 \rfloor] = A[\lfloor 16/2 \rfloor] = A[8]$ is a peak

Peak Finding: Example

Example:

| | | | | | | | | | | | | | | | |
|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 3 | 7 | 22 | 47 | 36 | 33 | 31 | 30 | 25 | 21 | 20 | 15 | 7 | 4 | 10 | 22 |

If $A[7] \geq A[8]$ then **return** FAST-PEAK-FINDING($A[0, 7]$)

Peak Finding: Example

Example:

| | | | | | | | | | | | | | | | |
|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 3 | 7 | 22 | 47 | 36 | 33 | 31 | 30 | 25 | 21 | 20 | 15 | 7 | 4 | 10 | 22 |

Length of subarray is 8

Peak Finding: Example

Example:

| | | | | | | | | | | | | | | | |
|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 3 | 7 | 22 | 47 | 36 | 33 | 31 | 30 | 25 | 21 | 20 | 15 | 7 | 4 | 10 | 22 |

Check whether $A[\lfloor n/2 \rfloor] = A[\lfloor 8/2 \rfloor] = A[4]$ is a peak

Peak Finding: Example

Example:

| | | | | | | | | | | | | | | | |
|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 3 | 7 | 22 | 47 | 36 | 33 | 31 | 30 | 25 | 21 | 20 | 15 | 7 | 4 | 10 | 22 |

If $A[3] \geq A[4]$ then **return** FAST-PEAK-FINDING($A[0, 3]$)

Peak Finding: Example

Example:

| | | | | | | | | | | | | | | | |
|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 3 | 7 | 22 | 47 | 36 | 33 | 31 | 30 | 25 | 21 | 20 | 15 | 7 | 4 | 10 | 22 |

Length of subarray is 4

Peak Finding: Example

Example:

| | | | | | | | | | | | | | | | |
|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 3 | 7 | 22 | 47 | 36 | 33 | 31 | 30 | 25 | 21 | 20 | 15 | 7 | 4 | 10 | 22 |

Check whether $A[\lfloor n/2 \rfloor] = A[\lfloor 4/2 \rfloor] = A[2]$ is a peak

Peak Finding: Example

Example:

| | | | | | | | | | | | | | | | |
|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 3 | 7 | 22 | 47 | 36 | 33 | 31 | 30 | 25 | 21 | 20 | 15 | 7 | 4 | 10 | 22 |

If $A[1] \geq A[2]$ then **return** FAST-PEAK-FINDING($A[0, 1]$)

Peak Finding: Example

Example:

| | | | | | | | | | | | | | | | |
|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 3 | 7 | 22 | 47 | 36 | 33 | 31 | 30 | 25 | 21 | 20 | 15 | 7 | 4 | 10 | 22 |

Else **return** FAST-PEAK-FINDING($A[3]$), which returns 3

Peak Finding: How fast is the Improved Algorithm?

How often does the Algorithm look at the array elements?

Peak Finding: How fast is the Improved Algorithm?

How often does the Algorithm look at the array elements?

- Without the recursive calls, the algorithm looks at the array elements at most 5 times

How often does the Algorithm look at the array elements?

- Without the recursive calls, the algorithm looks at the array elements at most 5 times
- Let $R(n)$ be the number of calls to FAST-PEAK-FINDING when the input array is of length n . Then:

$$R(1) = R(2) = 1$$

How often does the Algorithm look at the array elements?

- Without the recursive calls, the algorithm looks at the array elements at most 5 times
- Let $R(n)$ be the number of calls to `FAST-PEAK-FINDING` when the input array is of length n . Then:

$$R(1) = R(2) = 1$$

$$R(n) \leq$$

How often does the Algorithm look at the array elements?

- Without the recursive calls, the algorithm looks at the array elements at most 5 times
- Let $R(n)$ be the number of calls to FAST-PEAK-FINDING when the input array is of length n . Then:

$$R(1) = R(2) = 1$$

$$R(n) \leq R(\lfloor n/2 \rfloor) + 1, \text{ for } n \geq 3.$$

How often does the Algorithm look at the array elements?

- Without the recursive calls, the algorithm looks at the array elements at most 5 times
- Let $R(n)$ be the number of calls to FAST-PEAK-FINDING when the input array is of length n . Then:

$$R(1) = R(2) = 1$$

$$R(n) \leq R(\lfloor n/2 \rfloor) + 1, \text{ for } n \geq 3.$$

- Solving the recurrence (see lecture on recurrences):

How often does the Algorithm look at the array elements?

- Without the recursive calls, the algorithm looks at the array elements at most 5 times
- Let $R(n)$ be the number of calls to FAST-PEAK-FINDING when the input array is of length n . Then:

$$R(1) = R(2) = 1$$

$$R(n) \leq R(\lfloor n/2 \rfloor) + 1, \text{ for } n \geq 3.$$

- Solving the recurrence (see lecture on recurrences):

$$\begin{aligned} R(n) &\leq R(\lfloor n/2 \rfloor) + 1 \leq R(n/2) + 1 = R(\lfloor n/4 \rfloor) + 2 \\ &\leq R(n/4) + 2 = \dots \leq \lceil \log n \rceil. \end{aligned}$$

How often does the Algorithm look at the array elements?

- Without the recursive calls, the algorithm looks at the array elements at most 5 times
- Let $R(n)$ be the number of calls to FAST-PEAK-FINDING when the input array is of length n . Then:

$$R(1) = R(2) = 1$$

$$R(n) \leq R(\lfloor n/2 \rfloor) + 1, \text{ for } n \geq 3.$$

- Solving the recurrence (see lecture on recurrences):

$$\begin{aligned} R(n) &\leq R(\lfloor n/2 \rfloor) + 1 \leq R(n/2) + 1 = R(\lfloor n/4 \rfloor) + 2 \\ &\leq R(n/4) + 2 = \dots \leq \lceil \log n \rceil. \end{aligned}$$

- Hence, we look at most at $5 \lceil \log n \rceil$ array elements!

Why is the Algorithm correct?!

Steps 1,2,3
are clearly
correct

- 1 if A is of length 1 then return 0
- 2 if A is of length 2 then compare $A[0]$ and $A[1]$ and return position of larger element
- 3 if $A[\lfloor n/2 \rfloor]$ is a peak then return $\lfloor n/2 \rfloor$
- 4 Otherwise, if $A[\lfloor n/2 \rfloor - 1] \geq A[\lfloor n/2 \rfloor]$ then return $\text{FAST-PEAK-FINDING}(A[0, \lfloor n/2 \rfloor - 1])$
- 5 else return $\lfloor n/2 \rfloor + 1 + \text{FAST-PEAK-FINDING}(A[\lfloor n/2 \rfloor + 1, n - 1])$

Why is step 4 correct? (step 5 is similar)

Why is the Algorithm correct?!

Steps 1,2,3
are clearly
correct

- 1 if A is of length 1 then return 0
- 2 if A is of length 2 then compare $A[0]$ and $A[1]$ and return position of larger element
- 3 if $A[\lfloor n/2 \rfloor]$ is a peak then return $\lfloor n/2 \rfloor$
- 4 Otherwise, if $A[\lfloor n/2 \rfloor - 1] \geq A[\lfloor n/2 \rfloor]$ then return $\text{FAST-PEAK-FINDING}(A[0, \lfloor n/2 \rfloor - 1])$
- 5 else return $\lfloor n/2 \rfloor + 1 + \text{FAST-PEAK-FINDING}(A[\lfloor n/2 \rfloor + 1, n - 1])$

Why is step 4 correct? (step 5 is similar)

- Need to prove: peak in $A[0, \lfloor n/2 \rfloor - 1]$ is a peak in A

Why is the Algorithm correct?!

Steps 1,2,3
are clearly
correct

- 1 if A is of length 1 then return 0
- 2 if A is of length 2 then compare $A[0]$ and $A[1]$ and return position of larger element
- 3 if $A[\lfloor n/2 \rfloor]$ is a peak then return $\lfloor n/2 \rfloor$
- 4 Otherwise, if $A[\lfloor n/2 \rfloor - 1] \geq A[\lfloor n/2 \rfloor]$ then return $\text{FAST-PEAK-FINDING}(A[0, \lfloor n/2 \rfloor - 1])$
- 5 else return $\lfloor n/2 \rfloor + 1 + \text{FAST-PEAK-FINDING}(A[\lfloor n/2 \rfloor + 1, n - 1])$

Why is step 4 correct? (step 5 is similar)

- Need to prove: peak in $A[0, \lfloor n/2 \rfloor - 1]$ is a peak in A
- This is trivially true for every position $i < \lfloor n/2 \rfloor - 1$, since both cells adjacent to $A[i]$ are also contained in $A[0, \lfloor n/2 \rfloor - 1]$
- **Critical case:** $\lfloor n/2 \rfloor - 1$ is a peak in $A[0, \lfloor n/2 \rfloor - 1]$

Peak Finding: Correctness (2)

Why is the Algorithm correct?!

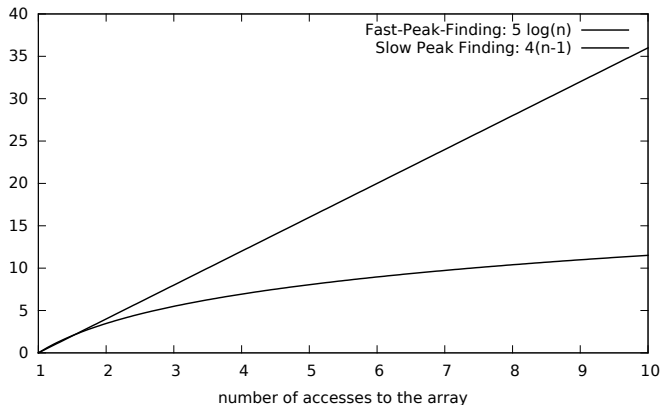
Steps 1,2,3
are clearly
correct

- 1 if A is of length 1 then return 0
- 2 if A is of length 2 then compare $A[0]$ and $A[1]$ and return position of larger element
- 3 if $A[\lfloor n/2 \rfloor]$ is a peak then return $\lfloor n/2 \rfloor$
- 4 Otherwise, if $A[\lfloor n/2 \rfloor - 1] \geq A[\lfloor n/2 \rfloor]$ then return FAST-PEAK-FINDING($A[0, \lfloor n/2 \rfloor - 1]$)
- 5 else return $\lfloor n/2 \rfloor + 1 +$
FAST-PEAK-FINDING($A[\lfloor n/2 \rfloor + 1, n - 1]$)

- **Critical case:** $\lfloor n/2 \rfloor - 1$ is a peak in $A[0, \lfloor n/2 \rfloor - 1]$
- Need to guarantee that $A[\lfloor n/2 \rfloor] \leq A[\lfloor n/2 \rfloor - 1]$ since otherwise $\lfloor n/2 \rfloor - 1$ would not be a peak
- This, however, follows from the condition in step 4! □

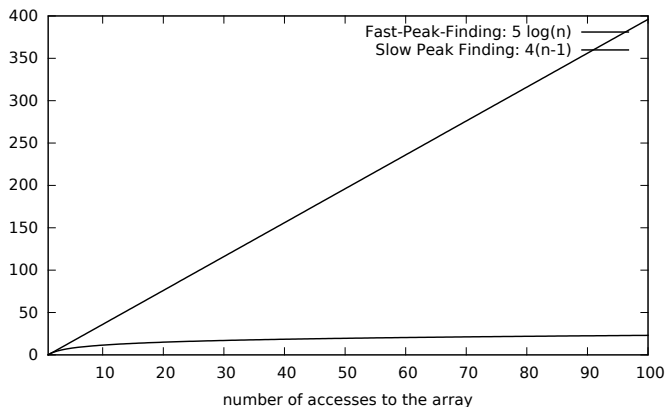
Peak Finding: Runtime Comparison

$4(n - 1)$ versus $5 \log n$



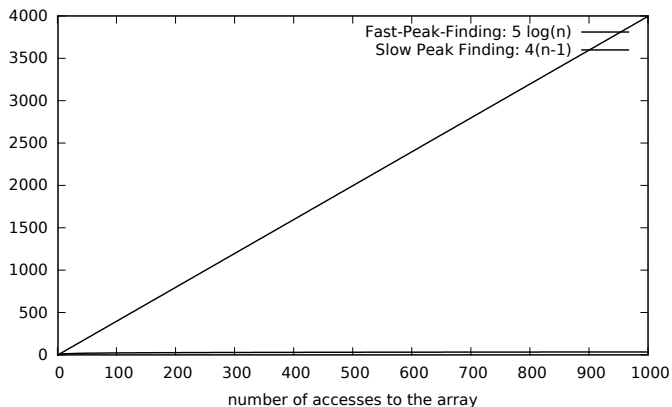
Peak Finding: Runtime Comparison

$4(n - 1)$ versus $5 \log n$



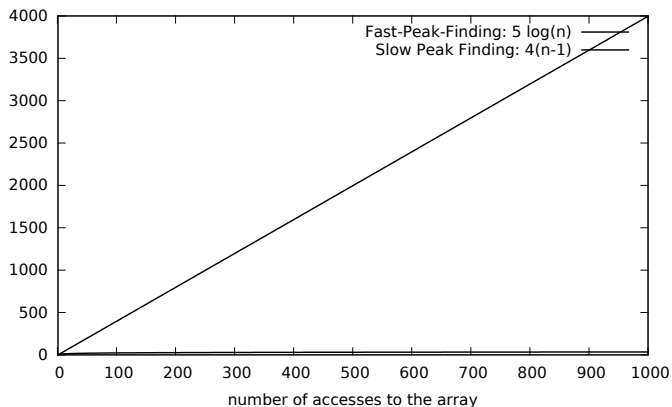
Peak Finding: Runtime Comparison

$4(n - 1)$ versus $5 \log n$



Peak Finding: Runtime Comparison

$4(n - 1)$ versus $5 \log n$



Conclusion: $5 \log n$ is so much better than $4(n - 1)$!