Algorithms?

A procedure that solves a computational problem

Computational Problem?

- Sort an array of \( n \) numbers
- Find the median of an array
- How often does “Juliet” appear in Shakespeare’s “Romeo And Juliet”?
- How do we factorize a large number?
- Shortest/fastest way to travel from Bristol to Glasgow?
- Is it possible to partition the set \{17, 8, 4, 22, 9, 28, 2\} into two sets s.t. their sums are equal? \{8, 9, 28\}, \{2, 4, 17, 22\}
What we want and how we work

Efficiency

The faster the better (runtime)
Use as little memory as possible (space complexity)

Mathematics

We will prove that algorithms run fast and use little memory
We will prove that algorithms are correct
Tools: Induction, algebra, sums, ..., rigorous arguments

Theoretical Computer Science

No implementations in this course. But please go ahead and write code...!
What you get out of this course

Goals

- First steps towards becoming an algorithms designer
- Learn techniques that help you design & analyze algorithms
- Understand a set of well-known algorithms

Systematic Approach to Problem/Puzzle Solving

- Study a problem at hand, discover structure within problem, exploit structure and design algorithms
- Useful in all areas of Computer Science
- Interview questions, Google, Facebook, Amazon, etc.
My Goals

- Get you excited about Algorithms
- Shape new generation of Algorithm Designers at Bristol (who solve all my open problems...)

Algorithms in Bristol

- 1st year: Algorithms (Algorithms 1)
- 2nd year: Data Structures and Algorithms (Algorithms 2)
- 3rd year: Advanced Algorithms (Algorithms 3)
- 4th year: in progress (Algorithms 4)

Projects, Theses, PhD students, Seminars
Course Structure

Teaching Units

- Lectures: Mondays 10-11am, Tuesdays 2-3pm, Room PHYS BLDG G42 POWELL, Instructor: Dr. Christian Konrad
- Exercise classes/in-class tests: Tuesdays 3pm-4pm (every fortnight), Room MVB 1.11

Assessment

- Exam: Counts 90%
- One In-class test: Counts 10% (March, 12th) (Extra time? let me know as soon as possible)
- You pass the course if your final grade is at least 40%
Teaching Staff

- Unit Director: Christian Konrad
- TAs: Thomas Delaney, Igor Dolecki, Nazaal Ibrahim, David Manda, Perla Jazmin Mayo Diaz de Leon, Matthew Owusu, Theano Xirouchaki

Drop-in Sessions

- Thursdays 5-6pm, MVB 3.44
- Fridays 1-2pm, MVB 3.44

My Office Hours Tuesdays 4-5pm in MVB 3.06 (to be confirmed!)
How to Succeed in this Course

Advice

- Make sure you understand the course material
- Work on provided exercises!
- Come to our drop in sessions
- Work on provided exercises!!
- Piazza for discussions and questions
- Work on provided exercises!!!
- Come to my office hours
- Course material has changed significantly from last year

Course webpage

http://people.cs.bris.ac.uk/~konrad/courses/COMS10007/coms10007.html

- News, announcements
- Download slides, exercises, etc.
Let $A = a_0, a_1, \ldots, a_{n-1}$ be an array of integers of length $n$

Definition: (Peak)
Integer $a_i$ is a peak if adjacent integers are not larger than $a_i$

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

\begin{tabular}{cccccccccc}
  0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
  \hline
  $a_0$ & $a_1$ & $a_2$ & $a_3$ & $a_4$ & $a_5$ & $a_6$ & $a_7$ & $a_8$ & $a_9$
\end{tabular}

**Definition:** (Peak)
Integer $a_i$ is a *peak* if adjacent integers are not larger than $a_i$

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

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_0$</td>
<td>$a_1$</td>
<td>$a_2$</td>
<td>$a_3$</td>
<td>$a_4$</td>
<td>$a_5$</td>
<td>$a_6$</td>
<td>$a_7$</td>
<td>$a_8$</td>
<td>$a_9$</td>
<td></td>
</tr>
</tbody>
</table>

**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 \( A = a_0, a_1, \ldots, a_{n-1} \) be an array of integers of length \( n \)

\[
\begin{array}{cccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\hline
a_0 & a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 & a_8 & a_9 \\
\end{array}
\]

**Definition:** (Peak)
Integer \( a_i \) is a *peak* if adjacent integers are not larger than \( a_i \)

**Example:**

\[
\begin{array}{cccccccccc}
4 & 3 & 9 & 10 & 14 & 8 & 7 & 2 & 2 & 2 \\
\end{array}
\]
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
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

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

    for (int i = 1; i < len - 1; i += 1) {
            return i;
    }
    return -1;
}
```

**C++ code**
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 $n$

```plaintext
if $A[0] \geq A[1]$ then
    return 0
if $A[n - 1] \geq A[n - 2]$ then
    return $n - 1$
for $i = 1 \ldots n - 2$ do
    if $A[i] \geq A[i - 1]$ and $A[i] \geq A[i + 1]$ then
        return $i$
return $-1$
```

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

**Lemma**

*Every integer array has at least one peak.*

**Proof.**
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.
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.
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.
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.
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.
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!)
How fast is our Algorithm?

```
Require: Integer array A of length n
            return 0
            return n - 1
        for i = 1 ... n - 2 do
                return i
        return -1
```

How often do we look at the array elements? (worst case!)
- A[0] and A[n - 1]: twice
- Overall: $2 + 2 + (n - 2) \cdot 4 = 4(n - 1)$

Can we do better?!
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** $\text{Fast-Peak-Finding}(A[0, \lfloor n/2 \rfloor - 1])$
5. **else** **return** $\lfloor n/2 \rfloor + 1 + \text{Fast-Peak-Finding}(A[\lfloor n/2 \rfloor + 1, n - 1])$

Comments:
- **Fast-Peak-Finding** is *recursive* (it calls itself)
- $\lfloor x \rfloor$ is the floor function ($\lceil x \rceil$: ceiling)
Example:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>7</td>
<td>22</td>
<td>47</td>
<td>36</td>
<td>33</td>
<td>31</td>
<td>30</td>
<td>25</td>
<td>21</td>
<td>20</td>
<td>15</td>
<td>7</td>
<td>4</td>
<td>10</td>
<td>22</td>
</tr>
</tbody>
</table>
Example:

Check whether $A\left\lfloor n/2 \right\rfloor = A\left\lfloor 16/2 \right\rfloor = A[8]$ is a peak.
Example:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>7</td>
<td>22</td>
<td>47</td>
<td>36</td>
<td>33</td>
<td>31</td>
<td>30</td>
<td>25</td>
<td>21</td>
<td>20</td>
<td>15</td>
<td>7</td>
<td>4</td>
<td>10</td>
<td>22</td>
</tr>
</tbody>
</table>

Example:

\[
\begin{array}{cccccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
3 & 7 & 22 & 47 & 36 & 33 & 31 & 30 & 25 & 21 & 20 & 15 & 7 & 4 & 10 & 22 \\
\end{array}
\]

Length of subarray is 8
Example:

Example:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>7</td>
<td>22</td>
<td>47</td>
<td>36</td>
<td>33</td>
<td>31</td>
<td>30</td>
<td>25</td>
<td>21</td>
<td>20</td>
<td>15</td>
<td>7</td>
<td>4</td>
<td>10</td>
<td>22</td>
</tr>
</tbody>
</table>

Length of subarray is 4
Peak Finding: Example

Example:

\[
\begin{array}{ccccccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
\hline
3 & 7 & 22 & 47 & 36 & 33 & 31 & 30 & 25 & 21 & 20 & 15 & 7 & 4 & 10 & 22 \\
\end{array}
\]

Check whether \( A[\lfloor n/2 \rfloor] = A[\lfloor 4/2 \rfloor] = A[2] \) is a peak
### Example:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
<td>7</td>
<td>22</td>
<td>47</td>
<td>36</td>
<td>33</td>
<td>31</td>
<td>30</td>
<td>25</td>
<td>21</td>
<td>20</td>
<td>15</td>
<td>7</td>
<td>4</td>
<td>10</td>
<td>22</td>
</tr>
</tbody>
</table>

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 \texttt{Fast-Peak-Finding}(A[3]), which returns 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 \texttt{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):

\[
R(n) \leq R(\lfloor n/2 \rfloor) + 1 \leq R(n/2) + 1 = R(\lfloor n/4 \rfloor) + 2 \\
\leq R(n/4) + 2 = \cdots \leq \lceil \log n \rceil.
\]

- 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.

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 $A$
- Critical case: $\lfloor n/2 \rfloor - 1$ is a peak in $A[0, \lfloor n/2 \rfloor - 1]$
- Condition in step 4 guarantees $A[\lfloor n/2 \rfloor - 1] \geq A[\lfloor n/2 \rfloor]$ and hence $\lfloor n/2 \rfloor - 1$ is a peak in $A$ as well (very important!)
4(n − 1) versus 5 log n

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

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

Conclusion: 5 log n is so much better than 4(n – 1)!