Introduction

- Algorithms and Data Structures
- Peaks
  - Algorithm 1
  - Algorithm 2
  - Algorithm 3

Example: Find max

- Find max. Given a table $A[0..n-1]$, find an index $i$, such that $A[i]$ is maximal.
  - Input. Table $A[0..n-1]$.
  - Output. An index $i$ such that $A[i] \geq A[j]$ for all indices $j \neq i$.

- Algorithm.
  - Process $A$ from left-to-right and maintain value and index of maximal value seen so far.
  - Return index.

<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>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>3</td>
<td>7</td>
<td>15</td>
<td>17</td>
<td>11</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>8</td>
<td>7</td>
<td>5</td>
<td>9</td>
<td>5</td>
</tr>
</tbody>
</table>
Description of Algorithms

- Natural language.
  - Process A from left-to-right and maintain value and index of maximal value seen so far.
  - Return index.
- Program.
- Pseudocode.

```java
public static int findMax(int[] A) {
    int max = 0;
    for (i = 0; i < A.length; i++)
        if (A[i] > A[max]) max = i;
    return max;
}
```

```
FindMax(A, n)
max = 0
for i = 0 to n-1
    if (A[i] > A[max]) max = i
return max
```

Peaks

- Peak. A[i] is a peak if A[i] is as least as large as it’s neighbors:
  - A[0] is a peak if A[0] ≥ A[1].

```
<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>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>7</td>
<td>15</td>
<td>17</td>
<td>11</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>8</td>
<td>9</td>
<td>5</td>
<td>9</td>
<td>5</td>
<td>23</td>
</tr>
</tbody>
</table>
```

- Peak finding. Given a table A[0..n-1], find an index i such that A[i] is a peak.
  - Input. A table A[0..n-1].
  - Output. An index i such that A[i] is a peak.
Algorithm 1

- **Algorithm 1.** For each entry check if it is a peak. Return the index of the first peak.

- **Challenges.** How do we analyze the algorithm?

```c
PEAK1(A, n)
for i = 1 to n-2
```

- **Pseudocode.**

**Theoretical Analysis**

- **Running time/time complexity.**
  - $T(n)$ = number of steps that the algorithm performs on input of size $n$.

- **Steps.**
  - Read/write to memory ($x := y, A[i], i = i + 1, ...$)
  - Arithmetic/boolean operations (+, -, *, /, %, &&, ||, &, |, ^, ~)
  - Comparisons ($<, >, <=, =>, =, !=$)
  - Program flow (if-then-else, while, for, goto, function call, ...)

- **Worst-case time complexity.** Maximal running time over all input of size $n$.

**Theoretical Analysis**

- **Running time.** What is the running time $T(n)$ for algorithm 1?

```c
PEAK1(A, n)
for i = 1 to n-2
```

$T(n) = c_1 + (n-2)c_2 + c_3$

- $T(n)$ is a linear function of $n$: $T(n) = an + b$
- **Asymptotic notation.** $T(n) = \Theta(n)$
- **Experimental analysis.**
  - What is the experimental running time of algorithm 1?
  - How does the experimental analysis compare to the theoretical analysis?
Peaks

- Algorithm 1 finds a peak in $\Theta(n)$ time.
- Theoretical and experimental analysis agrees.
- Challenge. Can we do better?

Algorithm 1

```
FINDMAX(A, n)
max = 0
for i = 0 to n-1
    if (A[i] > A[max]) max = i
return max
```

- Observation. A maximal entry $A[i]$ is a peak.
- Algorithm 2. Find a maximal entry in $A$ med FINDMAX($A$, $n$).

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
1 3 7 15 17 11 2 3 6 8 7 5 9 5 23
```

Introduction

- Algorithms and Data Structures
- Peaks
  - Algorithm 1
  - Algorithm 2
  - Algorithm 3

Theoretical Analysis

- Running time. What is the running time $T(n)$ for algorithm 2?

```
FINDMAX(A, n)
max = 0
for i = 0 to n-1
    if (A[i] > A[max]) max = i
return max
```

$T(n) = c_4 + n \cdot c_5 + c_6 = \Theta(n)$

- Experimental analysis. Better constants?
Peaks

- **Theoretical analysis.**
  - Algorithm 1 and 2 find a peak in $\Theta(n)$ time.

- **Experimental analysis.**
  - Algorithm 1 and 2 run in $\Theta(n)$ time in practice.
  - Algorithm 2 is a constant factor faster than algorithm 1.

- **Challenge.** Can we do significantly better?

Algorithm 3

- **Clever idea.**
  - Where can a peak be relative to $A[i]$?
    - Neighbor are $\leq A[i] \implies A[i]$ is a peak.
    - Otherwise $A$ is increasing in at least one direction $\implies$ peak must exist in that direction.

  \[
  \begin{array}{ccc}
  \hline
  i-1 & i & i+1 \\
  \hline
  3 & 10 & 7 \\
  \hline
  12 & 10 & 7 \\
  \hline
  3 & 10 & 15 \\
  \hline
  12 & 10 & 15 \\
  \hline
  \end{array}
  \]

- **Challenge.** How can we turn this into a fast algorithm?
Algorithm 3

- Consider the middle entry $A[m]$ and neighbors $A[m-1]$ and $A[m+1]$.
- If $A[m]$ is a peak, return $m$.
- Otherwise, continue search recursively in half with the increasing neighbor.

```
<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>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>7</td>
<td>15</td>
<td>17</td>
<td>11</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>8</td>
<td>7</td>
<td>5</td>
<td>9</td>
<td>5</td>
<td>23</td>
</tr>
</tbody>
</table>
```

Running time.
- A recursive call takes constant time.
- How many recursive calls?
- A recursive call halves size of interval. We stop when table has size 1.
- 1. recursive call: $n/2$
- 2. recursive call: $n/4$
- ....
- $k^{th}$ recursive call: $n/2^k$
- ....
- $\Rightarrow$ After $\sim$ log-n recursive call table has size $\leq 1$.
- $\Rightarrow$ Running time is $\Theta(\log n)$

Experimental analysis. Significantly better?
Peaks

- Theoretical analysis.
  - Algorithm 1 and 2 finds a peak in $\Theta(n)$ time.
  - Algorithm 3 finds a peak in $\Theta(\log n)$ time.
- Experimental analysis.
  - Algorithm 1 and 2 run in $\Theta(n)$ time in practice.
  - Algorithm 2 is a constant factor faster than algorithm 1.
  - Algorithm 3 is much, much faster than algorithm 1 and 3.

Introduction

- Algorithms and Data Structures
- Peaks
  - Algorithm 1
  - Algorithm 2
  - Algorithm 3