1D and 2D Peak Finding

Peak Finder

One-dimensional Version

Position 2 is a peak if and only if $b \geq a$ and $b \geq c$. Position 9 is a peak if $i \geq h$.

```
1 2 3 4 5 6 7 8 9
a b c d e f g h i
```

Figure 1: a-i are numbers

Problem: Find a peak if it exists (Does it always exist?)

Straightforward Algorithm

```
6 7 4 3 2 1 4 5
```

Start from left

```
1 2 \ldots n/2 \ldots n-1 n
```

Figure 2: Look at $n/2$ elements on average, could look at $n$ elements in the worst case

What if we start in the middle? For the configuration below, we would look at $n/2$ elements. Would we have to ever look at more than $n/2$ elements if we start in the middle, and choose a direction based on which neighboring element is larger that the middle element?
Can we do better?

![Figure 3: Divide & Conquer](image)

- If $a[n/2] < a[n/2 - 1]$ then only look at left half $1 \ldots n/2 - 1$ to look for peak
- Else if $a[n/2] < a[n/2 + 1]$ then only look at right half $n/2 + 1 \ldots n$ to look for peak
- Else $n/2$ position is a peak: WHY?

$$a[n/2] \geq a[n/2 - 1]$$
$$a[n/2] \geq a[n/2 + 1]$$

What is the complexity?

$$T(n) = T(n/2) + \sum_{i=1}^{\Theta(1)} = \Theta(1) + \ldots + \Theta(1) (\log_2(n) \text{ times}) = \Theta(\log_2(n))$$

to compare $a[n/2]$ to neighbors

In order to sum up the $\Theta(i)$'s as we do here, we need to find a constant that works for all. If $n = 1000000$, $\Theta(n)$ algo needs 13 sec in python. If algo is $\Theta(\log n)$ we only need 0.001 sec.

Argue that the algorithm is correct.

Two-dimensional Version

![Figure 4: Greedy Ascent Algorithm: $\Theta(nm)$ complexity, $\Theta(n^2)$ algorithm if $m = n$](image)

$a$ is a 2D-peak iff $a \geq b, a \geq d, a \geq c, a \geq e$
### Attempt #1: Extend 1D Divide and Conquer to 2D

- Pick middle column $j = m/2$.
- Find a 1D-peak at $i, j$.
- Use $(i, j)$ as a start point on row $i$ to find 1D-peak on row $i$.

**Attempt #1 fails**

**Problem:** 2D-peak may not exist on row $i$

End up with 14 which is not a 2D-peak.
Attempt # 2

- Pick middle column \( j = m/2 \)
- Find global maximum on column \( j \) at \((i, j)\)
- Compare \((i, j - 1), (i, j), (i, j + 1)\)
- Pick left columns of \((i, j - 1) > (i, j)\)
- Similarly for right
- \((i, j)\) is a 2D-peak if neither condition holds ← WHY?
- Solve the new problem with half the number of columns.
- When you have a single column, find global maximum and you’re done.

Example of Attempt #2

![Example Table]

Complexity of Attempt #2

If \( T(n, m) \) denotes work required to solve problem with \( n \) rows and \( m \) columns

\[
T(n, m) = \begin{cases} 
T(n, m/2) + \Theta(n) & (\text{to find global maximum on a column — (n rows)}) \\
\underbrace{\Theta(n) + \ldots + \Theta(n)}_{\log m} & \\
\Theta(n \log m) = \Theta(n \log n) & \text{if } m = n
\end{cases}
\]

Question: What if we replaced global maximum with 1D-peak in Attempt #2? Would that work?
Attempt # 3

- Pick middle column \( j = m/2 \) and middle row \( i = n/2 \).

- Find global maximum on both row \( i \) and column \( j \) such as \((a, b)\) in figure.

- If \((a, b)\) is on the middle row, compare with \((a, b - 1)\) and \((a, b + 1)\). If the global maximum is on the column, compare with \((a - 1, b)\) and \((a + 1, b)\). If the maximum is at the intersection of \( i \) and \( j \), return \((a, b)\) as a local maximum.

- Recurse on the subsection of the matrix with a square that contains a value greater than the value at \((a, b)\). For example, if \((a, b - 1) > (a, b)\) recurse on the upper right corner.

- Similarly for other possible values of \((a, b)\).

- \((a, b)\) is a 2D local peak if it is greater than its surrounding cells. For example, 
  \((a, b) > (a, b - 1)\) and \((a, b) > (a, b + 1)\).

- Solve the new problem with \(\frac{1}{4}\) of the original graph.

---

**Attempt # 3 Counterexample**

**Problem:** We can recurse on a section of the graph that has no 2D-peak in the recursed square.

Consider the following example:

Running the algorithm for the first time finds global peak 25 among the squares in the middle row and column. Comparing with neighbors finds that 26 is greater than 25 and we recurse on the lower left corner. Running the algorithm on the lower left square, we find global peak, 5, in the middle row and column. 5 is less than 6 so we recurse on the square to the right. Running the algorithm again gives 7 as a local maximum. However, 7 is smaller than 8 so it is not a local maximum. But since we have already looked at the middle column, we don’t recurse on it again. Therefore, running this algorithm does not
give a 2D-peak in this case.

We can find the only peak in this 2D graph by tracing the trail of increasing values around the square till we get 26. This method ultimately does not perform better than $O(nm)$ in the worst case.

However, if we consider the boundary as presented in class, we would find 26 as the maximum in the second recursion and thus find the 2D peak.
For proof of the 2D peak finding algorithm (with borders shown above) and complexity analysis, please refer to class and recitation notes.