1 Peak Interest

findPeak(r):

if r.left == null, return r

else if r.value > r.left.value and r.value > r.right.value, return r

else if r.left.value > r.value return findPeak(r.left)

else return findPeak(r.right)

Correctness:

Claim 1. *The algorithm returns a peak for any tree of height d.*

Proof by induction on d.

Base Case: $d = 1$. The algorithm returns r, which is trivially a peak.

Inductive Hypothesis: Assume the algorithm is correct for all trees of height k.

Inductive Case: If r.value > r.left.value and r.value > r.right.value, then r is a peak and the algorithm returns r. Otherwise, findPeak is called on either left or right (without loss of generality, say left), and by the IH, returns a peak v in the subtree rooted at left. Call the full tree T and the subtree rooted at left T_L. If v is not the root of T_L, then every one of v's neighbors in T is also in T_L. By the IH, v is a peak in T_L, and is therefore a peak in T. Otherwise, v is r.left. In this case, the IH tells us that v.value > v.left.value and v.value > v.right.value. Moreover, our algorithm only called itself on r.left if r.value < r.left.value = v.value. Hence, v has greater value than its neighbors and is a peak. □

Runtime: The runtime satisfies the recurrence $T(n) = T(n/2) + O(1)$. The solution to this recurrence can be obtained by unrolling, and is $O(\log n)$.

2 Twinzies

Notation: A[i;;j] denotes the subarray consisting of A[i] through A[j], inclusive.

median(A,B):

n = A.length

if n = 1, return max(A[0],B[0])

```
else if n is even
    if A[n/2] < B[n/2-1]
        return median(A[n/2;;n-1],B[0;;n/2-1])
    else if B[n/2] < A[n/2-1]
        return median(A[0;;n/2-1],B[n/2;;n-1])
    else if B[n/2-1] < A[n/2-1] < A[n/2] < B[n/2]
        return A[n/2]
    else if A[n/2-1] < B[n/2-1] < B[n/2] < A[n/2]
        return B[n/2]
    else if A[n/2-1] < B[n/2-1] < A[n/2] < B[n/2]
        return A[n/2]
    else if B[n/2-1] < A[n/2-1] < B[n/2] < A[n/2]
        return B[n/2]
else if n is odd
    if A[(n-1)/2] < B[(n-1)/2]
        return median(A[(n-1)/2;;n-1],B[0;;(n-1)/2])
    else
        return median(A[0;;(n-1)/2],B[(n-1)/2;;n-1])
```

Correctness:

Claim 2. *The algorithm medium() is correct for all inputs of size $\leq n$ for all n.*

Proof by induction on n.

Base Case: When $n = 1$, the max of the two elements in A and B is the median. The algorithm returns this value.

Inductive Hypothesis: Assume the algorithm is correct for all inputs of size $\leq k$ for some k.

Inductive Case: We will break the proof into the same cases the algorithm uses.

n is even: We will refer to the cases in the algorithm in order as cases 1 through 6. Note that each successive pair of cases is has A and B switched.

1. In this case, consider $i \in$ A[0;;n/2-1]. We know that all the elements of A[n/2;;n-1] and B[n/2-1;;n-1] are greater than A[i]. This is at least $n + 1$ elements. The median has exactly

$n-1$ elements greater, so A[i] is less than the median. Similarly, consider an element i in B[n/2;;n-1]. There are at least $n+1$ elements that are less than B[i], namely those in A[0;;n/2] and B[0;;n/2-1]. The median is greater than exactly n elements, so B[i] is greater than the median. Therefore the median for A and B is in the arrays in our recursive calls.

Moreover these new arrays have length $n/2+1$. We have removed $n/2-1$ elements which are greater than the median, leaving $n-1-(n/2-1) = n/2 = (n/2+1)-1$. It follows that the median of these two arrays is exactly the median of A and B. By our inductive hypothesis, our recursive call will return this value, so our algorithm is correct.

2. Symmetric with case 1.
3. In this case, there are exactly $n-1$ elements greater than A[n/2]: namely the elements in A[n/2+1;;n-1] and B[n/2;;n-1]. All other elements are less than A[n/2]. It follows that A[n/2] is the median.
4. Symmetric with case 3.
5. The argument from case 3 applies here as well.
6. Symmetric with case 5.

n is odd: We will argue just the first case. The second case is symmetric. Consider an element i in A[0;;(n-1)/2-1]. There are at least n elements greater than A[i], namely those in A[(n+1)/2;;n-1] and B[(n-1)/2;;n-1]. It follows that A[i] is less than the median. Similarly, consider an element i in B[(n+1)/2;;n-1]. There are at least $n+1$ elements less than B[i], namely those in A[0;;(n-1)/2] and B[0;;(n-1)/2]. It follows that B[i] is greater than the median.

Moreover, these new arrays have length $(n+1)/2$. We have removed $(n-1)/2$ elements which are greater than the median, leaving $n-1-(n-1)/2 = n/2-1/2 = (n-1)/2 = (n+1)/2-1$. It follows that the median of these two arrays is the median of A and B. By our inductive hypothesis, our recursive call will return this value, so our algorithm is correct. □

Runtime: We have $T(n) \leq T(n/2+1) + O(1)$. Informally, it is fine to simplify this as $T(n) = T(n/2) + O(1)$, which can be solved by unrolling to be $O(\log n)$. To prove this completely formally, we can prove by induction that the solution is $O(\log n)$ in the same way we proved the runtime of modular exponentiation, from lecture 7. (For those of you studying for the midterm, this is not a technique you need to know.)

3 How Many Sums?

Algorithm:

1. Let $\mathbf{a}$ be a vector of length n where for i from 0 to $n-1$, a_i is 1 if and only if $i \in A$.
2. Define $\mathbf{b}$ similarly for B.
3. Compute the convolution $\mathbf{c}$ of $\mathbf{a}$ and $\mathbf{b}$, using the FFT.

4. Return **c**.

Correctness: Element k of the convolution of a and b is

$$\sum_{i,j:i+j=k} a_i b_j$$

Note that a_i is 1 if and only if $i \in A$, and similarly with b_j. Hence, element k of the convolution will be the number of pairs (i, j) such that $i \in A$ and $j \in B$ and $i + j = k$. This is exactly the value we desire.

Runtime: We are simply computing the convolution of two vectors of length n. This is an $O(n \log n)$ operation if we use FFT.