10.8
Binary Search

Binary Search in Sorted Arrays

Input Sorted array A of n numbers and number x

Goal Is x in A?

```
BinarySearch(A[a..b], x):
    if (b − a < 0) return NO
    mid = A[⌊(a + b)/2⌋]
    if (x = mid) return YES
    if (x < mid)
        return BinarySearch(A[a..⌊(a + b)/2⌋ − 1], x)
    else
        return BinarySearch(A[⌊(a + b)/2⌋ + 1..b], x)
```

Analysis: $T(n) = T(\lfloor n/2 \rfloor) + O(1)$. $T(n) = O(\log n)$.

Observation: After k steps, size of array left is $n/2^k$

Binary Search in Sorted Arrays

Input Sorted array A of n numbers and number x

Goal Is x in A?

```
BinarySearch(A[a..b], x):
        if (b − a < 0) return NO
        mid = A[⌊(a + b)/2⌋]
        if (x = mid) return YES
        if (x < mid)
            return BinarySearch(A[a..⌊(a + b)/2⌋ − 1], x)
        else
            return BinarySearch(A[⌊(a + b)/2⌋ + 1..b],x)
```

Analysis: $T(n) = T(\lfloor n/2 \rfloor) + O(1)$. $T(n) = O(\log n)$.

Observation: After k steps, size of array left is $n/2^k$

Binary Search in Sorted Arrays

Input Sorted array A of n numbers and number x

Goal Is x in A?

```
BinarySearch(A[a..b], x):
        if (b − a < 0) return NO
        mid = A[⌊(a + b)/2⌋]
        if (x = mid) return YES
        if (x < mid)
            return BinarySearch(A[a..⌊(a + b)/2⌋ − 1], x)
        else
            return BinarySearch(A[⌊(a + b)/2⌋ + 1..b],x)
```

Analysis: $T(n) = T(\lfloor n/2 \rfloor) + O(1)$. $T(n) = O(\log n)$.

Observation: After k steps, size of array left is $n/2^k$

Another common use of binary search

1. Optimization version: find solution of best (say minimum) value
2. Decision version: is there a solution of value at most a given value v?

Reduce optimization to decision (may be easier to think about):

1. Given instance I compute upper bound $U(I)$ on best value
2. Compute lower bound $L(I)$ on best value
3. Do binary search on interval $[L(I), U(I)]$ using decision version as black box
4. $O(\log(U(I) - L(I)))$ calls to decision version if $U(I), L(I)$ are integers

Another common use of binary search

1. Optimization version: find solution of best (say minimum) value
2. Decision version: is there a solution of value at most a given value v?

Reduce optimization to decision (may be easier to think about):

1. Given instance I compute upper bound $U(I)$ on best value
2. Compute lower bound $L(I)$ on best value
3. Do binary search on interval $[L(I), U(I)]$ using decision version as black box
4. $O(\log(U(I) - L(I)))$ calls to decision version if $U(I), L(I)$ are integers

Example

1. Problem: shortest paths in a graph.
2. Decision version: given G with non-negative integer edge lengths, nodes s, t and bound B, is there an s-t path in G of length at most B?
3. Optimization version: find the length of a shortest path between s and t in G.

Question: given a black box algorithm for the decision version, can we obtain an algorithm for the optimization version?

Example continued

Question: given a black box algorithm for the decision version, can we obtain an algorithm for the optimization version?

1. Let U be maximum edge length in G.
2. Minimum edge length is L.
3. s-t shortest path length is at most $(n-1)U$ and at least L.
4. Apply binary search on the interval $[L, (n-1)U]$ via the algorithm for the decision problem.
5. $O(\log((n-1)U - L))$ calls to the decision problem algorithm sufficient. Polynomial in input size.

THE END

...

(for now)