Binary Search and Variants

- Applicable whenever it is possible to reduce the search space by half using one query
- Search space size N
  number of queries = O(*log* N)
- Ubiquitous in algorithm design. Almost every complex enough algorithm will have binary search somewhere.

Classic example

- Given a sorted array A of integers, find the location of a target number x (or say that it is not present)

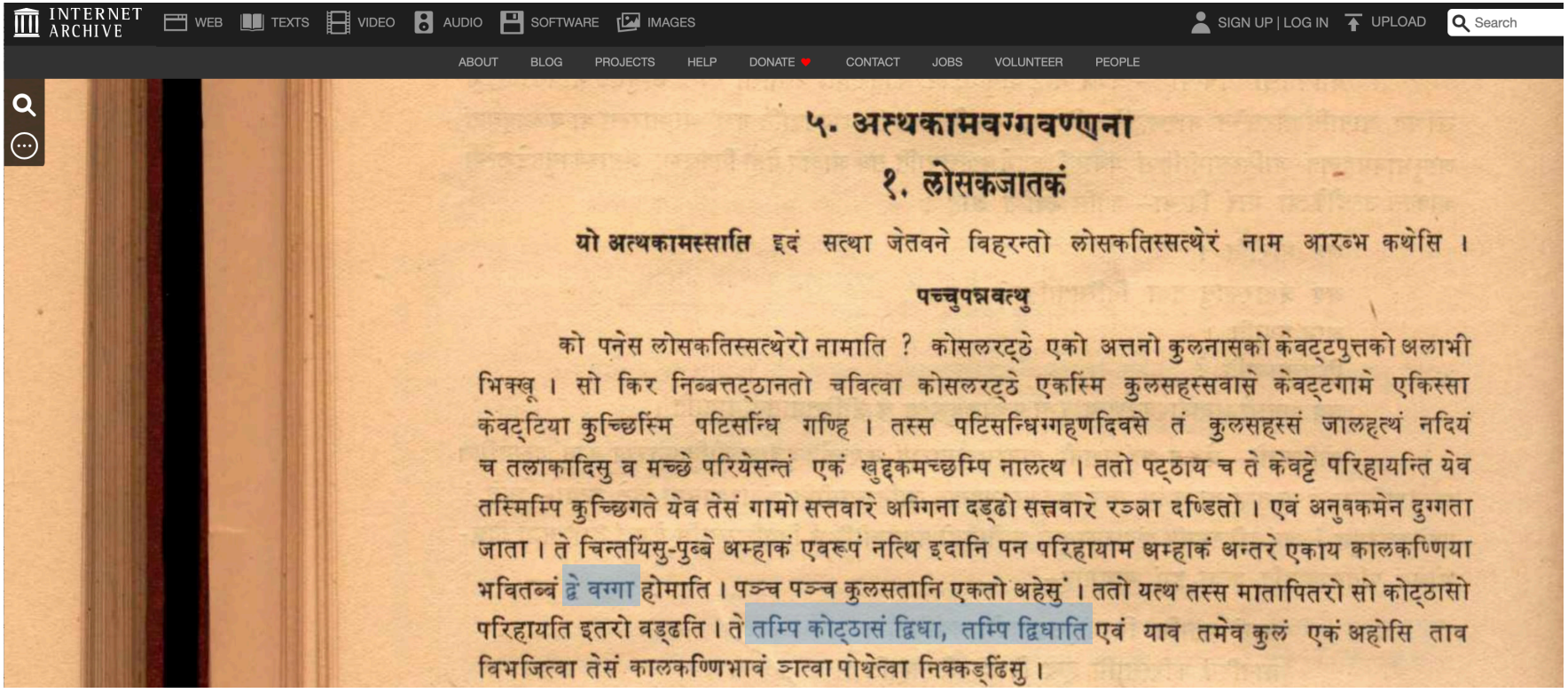
Pseudocode:

```
Initialize start ← 0, end ← n;
Locate(x, start, end){
    if (end < start) return not found;
    mid ← (start+end)/2;
    if (A[mid] = x) return mid;
    if (A[mid] < x) return Locate(x, mid+1, end);
    if (A[mid] > x) return Locate(x, start, mid);
}
```

History

- Binary search was first first mentioned by John Mauchly (1946) - The art of computer programming, vol 3, p 422

- Mention in Buddhist Jataka Tales



Other examples

- Looking for a word in the dictionary
- Finding a scene in a movie
- Debugging a linear piece of code
- Cooking
- Science/Engineering: Finding the right value of any resource
  - length of youtube ads
  - pricing of a service
- Twenty questions

Egg drop problem

- In a building with n floors,
  find the highest floor from where egg can be dropped without breaking.
- $O(\log n)$ egg drops are sufficient.
- Binary search for the answer.
- Drop an egg from floor x
  - if the egg breaks, answer is less than x
  - if the egg doesn't break, the answer is at least x
- Using the standard binary search idea, start with $x = n/2$.
  If egg breaks, then go to $x = n/4$ and it doesn't then go to $x = 3n/4$.
  And so on

Egg drop: unknown range

- In a building with *unknown number of* floors,
  find the highest floor h from where egg can be dropped without breaking.
- $O(\log h)$ egg drops sufficient?
- Exponential search: Try floors, 1, 2, 4, …, till the egg breaks.
- The egg will beak at floor 2^{k+1}, where
  $2^k \leq h < 2^{k+1}$
- Then binary search in the range $[2^k , 2^{k+1}]$
- Total number of egg drops $\leq k+1+k = 2k+1 \leq 2 \log h + 1$.
- Is there a better way?

Lower bound

- Search space size: N
  Are $log\ N$ queries necessary?

- Yes. When each query is a yes/no type, then the search space gets divided into two parts with each query (some solutions correspond to yes and others to no).

- One of the parts will be at least $N/2$.

- In worst case, with each query, we get the larger of the two parts.

- To reduce the search space size to 1, we need $log\ N$ queries

Lower bound

- Search space size: N
  Are $log\ N$ queries necessary? Another argument.
- Suppose you make k queries.
- There are 2^k possible outcomes.
- If $2^k < N$, then two different elements must lead to same outcomes for all queries.
- We won't be able to say which of the two is correct.

Lower bound

- Search space size: N
  Are $\log N$ queries necessary?
- Another argument based on information theory.
- Each yes/no query gives us 1 bit of information.
- The final answer is a number between 1 and N, and thus, requires $\log N$ bits of information.
- Hence, $\log N$ queries are necessary.
- Ignore this argument if it is hard to digest.

Exercise: subarray sum

- Given an array with n positive integers,
  and a number S,
  find the minimum length subarray whose sum is at least S?

- Subarray is a contiguous subset, i.e.,
  A[i], A[i+1], A[i+2], …, A[j-1], A[j]

- [10, 12, 4, 9, 3, 7, 14, 8, 2, 11, 6]

  S = 27

- Can you design an $O(n \log n)$ algorithm?

Exercise: subarray sum

$O(n^3)$ algorithm:

```
for (l ← 1 to n){ / / looping over all possible lengths
        for ( j ← 0 to n-l-1){   / / looping over all possible starting points
            T ← sum(A[j], A[j+1], …, A[j+l-1])
            if T ≥ S
                then return l and the subarray (j, j+l-1);
        }
}
```

- Two for loops one inside the other, each making at most n iterations.

- Sum function is another for loop with at most n iterations.

Exercise: subarray sum

$O(n^2)$ algorithm:

```
for (l ← 1 to n){ // looping over all possible lengths
        T ← sum of first l numbers.
        for (j ← 0 to n-l-1){ // looping over all possible starting points
            if T ≥ S then return l and the subarray (j, j+l-1);
            T ← T - A[j] + A[l+j];
        }
}
```

- Two for loops one inside the other. Each makes at most n iterations. Hence, $O(n^2)$ time.

Exercise: subarray sum

O($n \log n$) algorithm. Approach 1:

Binary search for the minimum length l.

For the current value of l:

- Check if there is a subarray of length l with sum at least S.
- This check can be done in $O(n)$ time. See the inner loop on previous slide.
- If YES then try a smaller value of l
- If NO then try a larger value of l

Exercise: subarray sum

- O(*n log n*) algorithm. Approach 2 (suggested by students):

- First compute all the prefix sums and store in an array.
  O(n) time.

- prefix_sum[0] ← A[0];
  for (i ← 1 to n-1)
  prefix_sum[*i*] = prefix_sum[*i-1*]+A[*i*];

- Any subarray sum from *i* to *j* can now be computed in O(1) time as prefix_sum[*j*] - prefix_sum[*i-1*].

- Now, for each choice of starting point, do a binary search for the minimum end point such that the subarray sum is at least *S*.
  - If sum of a subarray < *S*, then choose a larger end point, otherwise smaller.

Exercises

- Given two sorted arrays of size n, find the median of the union of the two arrays.
  $O(\log n)$ time?

- Given a convex function $f(x)$ oracle, find an integer x which minimizes $f(x)$.

- Land redistribution:
  given list of landholdings $a_1, a_2, \ldots, a_n$,
  given a floor value f,
  find the right ceiling value c