MCS-375: Algorithms: Analysis and Design Handout #SS1
San Skulrattanakulchai
Gustavus Adolphus College
Nov 24, 2014

Binary Search

Given n items, we want to preprocess them so we can repeatedly search for a "query" item. This is what directory assistance does in a telephone search system. (There we query names.) Preprocessing is done only once but querying will be the main operation so should be optimized.

If the items are totally arbitrary, it seems we have no choice but to keep them in a simple data structure like an array or a list, and to implement querying by linear search. But linear search takes $\Theta(n)$ time in the worst-case. To speed up querying, we can use hashing.

It's a whole different story if the items are drawn from a universe with a total order. We can put the items in a balanced binary search tree. This method achieves $O(\log n)$ time per query after a one-time initialization of the tree that takes $O(n \log n)$ time.

Binary search also achieves the $O(\log n)$ bound per query after a one-time $O(n \log n)$ time initialization, but is faster in practice because it has less overhead.

In binary search we first place the given items in an array $A[1..n]$ in nondecreasing order. We then search for each query item x by executing the following procedure.

The procedure maintains the invariant $\boxed{A[\ell] \leq x \leq A[h] \text{ if } x \text{ is in the array}}$.

```
$\ell \leftarrow 1; h \leftarrow n;$
while $\ell \leq h$ do {
    $p \leftarrow \lfloor (\ell + h)/2 \rfloor$
    if $x = A[p]$ then return $p$
    else if $x < A[p]$ then $h \leftarrow p - 1$
    else /* $x > A[p]$ */ $\ell \leftarrow p + 1$
}
return "$x$ not present"
```

Running Time Analysis:

Running time is $O(\log n)$ because each probe takes time $O(1)$, and there are $O(\log n)$ probes because

(i) Each iteration halves the size of the array being searched. If we start with s elements between ℓ and h, the next array has $\leq s/2$ elements, e.g., starting with 11 elements gives 5 elements; starting with 10 elements gives 4 or 5 elements.

(ii) A number can be halved $\leq \lg n$ times before it reaches 1.

A Binary Search Problem

The code below is meant to return $\lfloor\sqrt{n}\rfloor$. But there's a bug, e.g., it returns 3 for $\lfloor\sqrt{6}\rfloor$. Find and modify the buggy statement so the procedure works correctly. The time should be $O(\log n)$.

```
$\ell \leftarrow 1; h \leftarrow n;$
**while** $\ell < h$ **do** {
    $p \leftarrow \lceil(\ell + h)/2\rceil$
    **if** $p * p > n$ **then** $h \leftarrow p - 1$
    **else** /* $p * p \leq n$ */ $\ell \leftarrow p + 1$
}
**return** $\ell$;
```

Hint: Note that $\lfloor\sqrt{n}\rfloor$ is the largest integer whose square is $\leq n$. Maintain the invariant $\ell \leq \lfloor\sqrt{n}\rfloor \leq h$.