Algorithm and Data Structure Analysis (ADSA)

Correctness and Invariants: Example Binary Search

(Book Chapter 2)

Overview

- Correctness of algorithms/programs
- Invariants
- Binary Search

Correctness of Algorithms

You want to have algorithms that are

- Correct
- and efficient

Correctness has highest priority.

You want to be sure that your program does what you want.

Invariants

- Invariants are a powerful tool to show correctness of an algorithm/program
- An invariant is a property of an algorithm that holds during the execution of the program.
- It is often used to show correctness of loops
- Define: Preconditions, Invariants, and Postconditions

Often it is non-trivial to find an invariant.

Example

Consider the following while loop

```
int x=10;
while (x < 20){
x=x+1;
}
```

Precondition: Before entering the while-loop x < 20 holds.

Invariant: During the execution of the while-loop x <=20 holds.

Postcondition: After execution of the while-loop x<=20 but not x<20 holds. This implies that x=20 holds.

Let's play a game

a = {1, ..., 15} consists of all integers from 1, ..., 15.

Player 1 picks a secret number x of a.

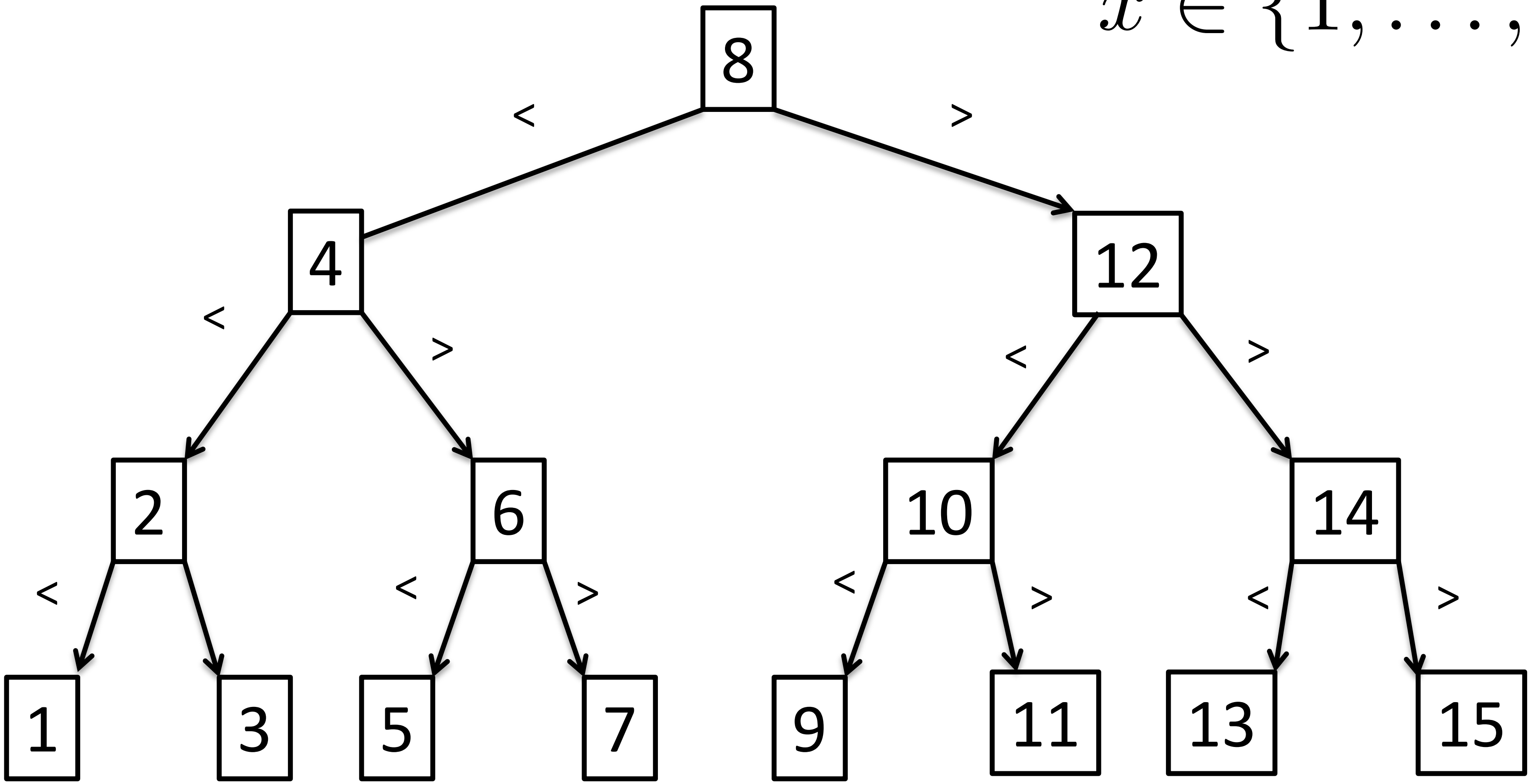
Player 2 has to guess x.

- Player 2: Can query in each step a number y
- Answer of Player 1 is either
  - found if x=y
  - x is greater than y
  - x is smaller than y

You are Player 2:
What is your strategy to use the smallest number of queries to reveal x?

Binary Search

$$x \in \{1, \ldots, 15\}$$



At most 3 comparisons to determine x

Problem Statement:

- Given: A sorted array a[1 ..n] of pairwise distinct elements, i.e.

  $a[1] < a[2] < \ldots < a[n]$, and an element x

  $a[0] = -\infty$ and $a[n+1] = \infty$

- Find: Index i such that $a[i-1] < x \le a[i]$

Binary Search

Divide-and-conquer algorithm

Procedure:

- Choose index index $m \in [1..n]$
- Compare x with a[m]
- If x=a[m] we are done
- If x < a[m], search in the part of the array before a[m].
- If x > a[m], search in the part of the array after a[m].

- Use two indices l and r.
- Maintain the invariant

$$(I) \quad 0 \leq l < r \leq n+1 \quad \text{and} \quad a[l] < x < a[r]$$

- Start with l=0 and r=n+1
- Choose m in the middle of the interval defined by l and r.
- If x ≠ a[m], change l or r accordingly.
- If l and r are consecutive indices then x is not contained in the array.

Binary Search Program

$(\ell, r) := (0, n+1)$
while *true* **do**
  invariant I // i.e., invariant (I) holds here
  if $\ell + 1 = r$ **then return** "$a[\ell] < x < a[\ell+1]$"
  $m := \lfloor (r+\ell)/2 \rfloor$ // $\ell < m < r$
  $s := compare(x, a[m])$ // -1 if $x < a[m]$, 0 if $x = a[m]$, $+1$ if $x > a[m]$
  if $s = 0$ **then return** "x *is equal to* $a[m]$";
  if $s < 0$
    then $r := m$ // $a[\ell] < x < a[m] = a[r]$
    else $\ell := m$ // $a[\ell] = a[m] < x < a[r]$

Choose the middle of current interval

Mehlhorn/Sanders (page 35)

Invariant Part 1

$$0 \le l < r \le n + 1$$

Loop is entered with $0 \le l < r \le n + 1$

If $l + 1 = r$, we stop

Otherwise, $l + 2 \le r$

and hence $l < m < r$.

Implies that m is a legal array index

If $x = a[m]$, we stop

Otherwise we set either $r = m$ or $l = m$

and hence $0 \le l < r \le n + 1$ at the end of the loop.

Invariant Part 2

$a[l] < x < a[r]$

Loop is entered with $a[l] < x < a[r]$

If $l + 1 = r$, we stop

Otherwise, $l + 2 \leq r$

and hence $l < m < r$.

If $x = a[m]$, we stop

If $x < a[m]$, we set $r = m$ which implies $a[l] < x < a[r]$ at the end of the loop.

If $x > a[m]$, we set $l = m$ which implies $a[l] < x < a[r]$ at the end of the loop.

Termination

- If an iteration is not the last one, we either increase l or decrease r.
- Hence r-l decreases.
- Implies that the search terminates.

Runtime

Theorem:

Binary search finds an element in a sorted array of size n in $2 + \lfloor \log n \rfloor$ comparisons between elements.

Proof:

Study the number of indices i with $l < i < r$.

There are $r - l - 1$ such indices.

We call the number of such indices the size of the problem.

Idea:

Show that each iteration (except the last one) halves the size of the problem.

Proof

Let $r - l - 1$ be the size of the problem.

Then the size of the problem decreases to

$$\begin{aligned} &\max\{r - \lfloor (r+\ell)/2 \rfloor - 1, \lfloor (r+\ell)/2 \rfloor - \ell - 1\} \\ &\quad \le \max\{r - ((r+\ell)/2 - 1/2) - 1, (r+\ell)/2 - \ell - 1\} \\ &\quad = \max\{(r-\ell-1)/2, (r-\ell)/2 - 1\} = (r-\ell-1)/2 \end{aligned}$$

Hence the size of the problem is at least halved

Proof

We start with problem size $r - l - 1 = n + 1 - 0 - 1 = n$.

After 1 iterations: $r - l - 1 \leq \lfloor n/2 \rfloor$.

After k iterations: $r - l - 1 \leq \lfloor n/2^k \rfloor$.

Interation $k + 1$ is the last if we enter it with $r = l + 1$.

This holds if $n/2^k < 1$.

Choosing $k = \log n + 1$ implies $n/2^k < 1$.

Hence, at most $2 + \log n$ iterations are performed.

Number of comparisons is natural number which implies the $2 + \lfloor \log n \rfloor$ bound. $\square$

Summary

- Invariants are an important tool to show correctness of algorithms/programs.
- Binary Search is effective to locate elements in a sorted array.
- Algorithm maintains two invariants.
- It halves the problem size in each iteration.
- This implies O(log n) comparisons.