Algorithms over data sets: Binary Search

Algorithms

Alberto Valderruten, Elena Hernández Pereira,
José M. Casanova

Computer Science Department
Faculty of Computer Science

UNIVERSIDADE DA CORUÑA

- Example of a *logarithmic algorithm*
- Given x and an *ordered* array a_1, a_2, ... a_n of integer,

$$\text{return: } \begin{cases} i \text{ if } \exists a_i = x \\ \text{“element not found”} \end{cases}$$

$\rightarrow$ Compare x and a_{middle}, with $middle = (i + j) div 2$, being $a_i..a_j$ the *search space*:

1. $x = a_{middle}$: finish (interruption)
2. $x > a_{middle}$: continue searching in $a_{middle+1}..a_j$
3. $x < a_{middle}$: continue searching in $a_i..a_{middle-1}$

- ¿number of iterations? $\leftrightarrow$ size d evolution in the search space

Invariant: $d = j - i + 1$

$$\text{¿How is } d \text{ decreasing? } \begin{cases} i \leftarrow middle + 1 \\ j \leftarrow middle - 1 \end{cases}$$

- *Worst case*: the normal termination of the loop is reached $\equiv i > j$

```
    function Binary Search(x, a[1..n]): position
      {a: ordered array with no decreasing ordered }

{1}   i := 1 ; j := n ;              {search space: i..j}
{2}   while i <= j do
{3}       middle := (i + j) div 2 ;
{4}       if a[middle] < x then
{5}           i := middle + 1
{6}       else if  a[middle] > x then
{7}           j := middle - 1
{8}       else return  middle       {the loop stops}
      end while;
{9}   return "element not found" {normal loop end}
    end function
```

... worst case

- Being $< d, i, j >$ iteration $< d', i', j' >$:
  1. $i \leftarrow middle + 1$:
  $$\begin{aligned} i' &= (i+j) div 2 + 1 \\ j' &= j \\ d' = j' - i' + 1 &= j - (i+j) div 2 - 1 + 1 \\ &\leq j - (i+j-1)/2 \\ &= (j-i+1)/2 \\ &= d/2 \end{aligned}$$
  $$\boxed{\rightarrow d' \leq d/2}$$
  2. $j \leftarrow middle - 1$:
  $$\begin{aligned} i' &= i \\ j' &= (i+j) div 2 - 1 \\ d' = j' - i' + 1 &= (i+j) div 2 - i - 1 + 1 \\ &\leq (i+j)/2 - i \\ &< (j-i+1)/2 \\ &= d/2 \end{aligned}$$
  $$\boxed{\rightarrow d' < d/2} \text{ (decreases faster)}$$

... worst case

- ¿$T(n)$? Being d_l: d *after* the l-th iteration
  $$\begin{cases} d_0 = n \\ d_l \leq d_{l-1}/2 \ \forall l \geq 1 \end{cases} \quad \text{(induction)} \rightarrow d_l \leq n/2^l$$
  until $d < 1 \rightarrow l = \lceil log_2 n \rceil + 1 = O(logn)$ iterations
  Each iteration is $\Theta(1)$ (rules) $\Rightarrow T(n) = O(logn)$
- **Alternative reasoning**: *thinking in a recursive version*
  $$T(n) = \begin{cases} 1 & \text{if } n = 0, 1 \\ T(n/2) + 1 & \text{if } n > 1 \end{cases}$$
  Theorem Divide and Conquer: $l = 1, b = 2, c = 1, k = 0, n_0 = 1$
  Case $l = b^k \Rightarrow T(n) = \Theta(n^k logn) \rightarrow T(n) = \Theta(logn)$
- **Conclusions**:
  - Think about a recursive version would be useful
  - is Divide and Conquer? $\rightarrow$ $\boxed{\text{Reduction algorithms}}$ $(l = 1)$
  - $T(n) = \Theta(logn) \leftrightarrow$ data is in memory
    (Computational model)

Exercise:

- Characterise the best case (when does it occur?)
- Best case analysis
- Characterise the worst case
- Demonstrate, applying the rules in detail, that each iteration is $\Theta(1)$
- Design a recursive version of the binary search algorithm
- Analyse the resulting recursive algorithm applying the recurrence resolution theorem Divide and Conquer

⋆ Brassard, G. and Bratley, P. Fundamentals of algorithmics. Prentice Hall, 1996.