COMP 3711 Fall 2021

Design & Analysis of Algorithms

Written By: Ljm

Topic 2 **Divide & Conquer**

1 Intro: Binary Search

The main idea of Divide & Conquer is to solve a problem(such as of size n) by breaking it into one or more smaller(size less than n) problems. We use binary search example to illustrate that.

Problem: given an **sorted** array of length n, how to find the position of element x; if x does not exist in the array, output nil.

Since the array is already sorted, it has a good property that: **for each item a_i, those who are larger than a_i must be on its right side, while smaller than a_i must be on its left side.** Hence we come up with an idea that we check the middle item mid first, then we will be able to know which direction to go: left or right, depending on the comparison of mid and x(the item we're looking for). If we go left, then the right half will be directly abandoned. Then we continue this process, check middle item each time, and abandon half items each time.

Algorithm 1: BinarySearch($a[]$, $left$, $right$, x)

```
Data: a[]: the array given, x: the item to find
if left = right then
    if a[left] = x then
        return left
    else
        return nil
    end
else
    mid = ⌊(left + right)/2⌋
    if x ≤ a[mid] then
        BinarySearch(a[], left, mid, x)
    else
        BinarySearch(a[], mid + 1, right, x)
    end
end
```

First call: BinarySearch($a[]$, 1, n, x).

This algorithm is quite efficient, since each time we eliminate half of the array, with one additional comparison, until there is only one item left, when we will end the process.

Then let's analyse its time complexity. Let $T(n)$ be the number of comparisons needed for n elements, then we will have

$$T(n) = T(n/2) + 1,\ T(1) = 1$$

.

Solve this **recurrence**:

$$\begin{aligned} T(n) &= T(n/2) + 1 \\ &= [T(n/4) + 1] + 1 \\ &= T(n/4) + 2 \\ &= \cdots \\ &= T(n/2^i) + i \end{aligned}$$

This process ends when reaching $T(1)$, i.e., $i = \log_2 n$, thus, $T(n) = T(1) + \log_2 n = \log_2 n + 1$.

We can also visualize this recurrence with recursion tree: (image from `https://cs61bl.org/su21/labs/lab10/`)