Technische Universität München
Institut für Informatik
Theoretical Computer Science

WS 2017/18

Fundamental Algorithms 5 - Solution Examples

Exercise 1 (Optimizing SeqSearch)

1. Optimize the algorithm SeqSearch for sorted arrays, i.e. that it stops the search as soon as the elements become too large.
2. Make a reasonable assumption for the probability that x is found at position i or not found at all, and give an estimate of the average number of comparisons that are required.
3. How does the complexity differ from the regular SeqSearch algorithm? What causes this difference?

Solution:
A modified version of SeqSearch is Note that this version only requires one comparisons per loop iteration

Algorithm 1: SeqSearchMod

```
Input: A: Array[1..n]
       x: Element
Result: Index i such that A[i] = x or -1 if x not in A
for i = 1 to n do
    if x ≥ A[i] then
        if x = A[i] then return i;
        return −1;
    end
end
return −1;
```

except for the last one, which potentially requires two comparisons.

We use the same assumptions as in the lecture:

- x occurs either once or not at all in A
- The probability that $x = A[i]$ is independent of the position i, which we denote by p

Thus, we can follow

- $p \leq \frac{1}{n}$ in general $\Rightarrow$ probability that $x \notin A$: $(1 - np)$
- $p = \frac{1}{n}$, iff $x \in A$

Nothing changed so far. (Average) Number of comparisons

- In case of success: $i + 1$
- In case of non-success: $\frac{1}{2}n$.

Therefore, the expected number of comparisons is:

$$\bar{C}(n) = \sum_{i=1}^{n} p(i+1) + (1-np)\frac{1}{2}n = p\left(\frac{n(n+1)}{2} + n\right) + \frac{1}{2}(1-np)n.$$

If $x \in A$, we can simplify to

$$\bar{C}(n) = \frac{n(n+1)}{2n} + 1 = \frac{1}{2}(n+1) + 1.$$

In this case, we obtain one more comparison as in the uninformed setting, since we are not exploiting the sortedness.

Exercise 2 (Binary Search)

Formulate a recurrence equation for the number of comparisons in the BinarySearch algorithm of the lecture. Solve this equation to estimate the time complexity of BinarySearch.

Hint: You may assume that the input is of size 2^k for some $k \in \mathbb{N}$.

Solution:

Since $n = 2^k$, we can neglect $\lfloor\cdot\rfloor$ and $\lceil\cdot\rceil$ as usual. There are two comparisons per function call. The recurrence formula is

$$T(n) = T\left(\tfrac{1}{2}n\right) + 2$$

here are plenty of ways to solve the recurrence formula:

1. Direct:

$$\begin{aligned} T(n) &= T\left(\tfrac{1}{2}n\right) + 2 \\ &= T\left(\tfrac{1}{4}n\right) + 2 + 2 \\ &= T\left(\tfrac{1}{2^3}n\right) + 2 + 2 + 2 \\ &= \ldots \\ &= T\left(\tfrac{2^k}{2^k}\right) + 2k \\ &= 2 + 2k = 2(k+1) \\ &\in \Theta(k) = \Theta(\log n) \end{aligned}$$

2. Master Theorem: We have $a = 1$, $b = 2$, $f(n) = 2$, and

$$f(n) = 2 \in \Theta(n^{\log_2 1}) = \Theta(n^0).$$

Hence

$$T(n) \in \Theta(n^{\log_2 1} \log n) = \Theta(\log n).$$

3. Substitution method …