RWTH Aachen **SS 2012**
Lehrgebiet Theoretische Informatik Problem Set 7
Langer—Reidl—Rossmanith—Sikdar 4.6.2012

Analysis of Algorithms — Tutorial

Problem 7-1

Consider the following algorithm that searches an element x in a sorted array a of length $n = km + 1$:

```
i := 1;
while a[i] ≤ x
  if a[i] = x then return i;
  i := i + m;
  if i > n return 0;
for j = i − 1 downto max(1, i − (m − 1))
  if a[j] = x then return j;
  if a[j] < x then return 0;
return 0;
```

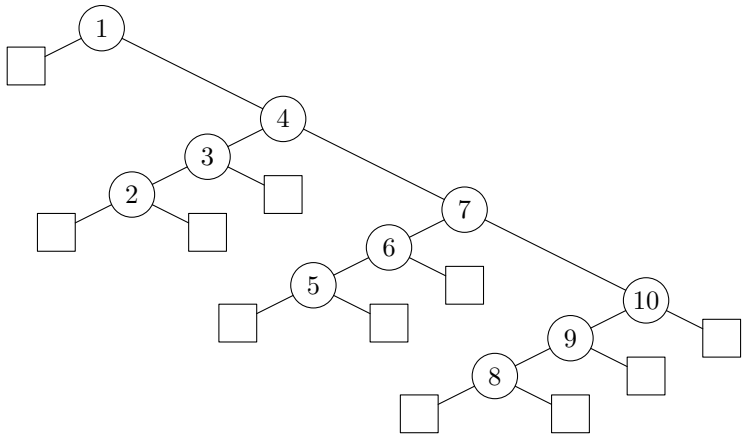
a) Draw the search tree and compute the internal and external path length for $n = 10$ and $m = 3$.

b) Determine C^+ and C^- for arbitrary m, k.

c) What is, for given n, the best choice for m w.r.t. the running time?

Solution:

a) The path lengths are $\pi(T) = 0 + 1 + 2 + 2 + 3 + 3 + 3 + 4 + 4 + 5 = 27$ and $\xi(T) = 1 + 3 + 4 + 4 + 4 + 4 + 5 + 5 + 5 + 6 + 6 = 47$:



b) It is sufficient to compute π. We then obtain

$$C^+ = \frac{\pi(T)}{n} + 1 \text{ and } C^- = \frac{\pi(T) + 2n}{n+1}.$$

Thus,

$$\begin{aligned}
\pi(T) &= \sum_{i=1}^{k} \sum_{j=i}^{i+m-1} j \\
&= \sum_{i=1}^{k} \frac{(i+m)(i+m-1)}{2} - \frac{i(i-1)}{2} \\
&= \frac{1}{2} \sum_{i=1}^{k} (2mi + m(m-1)) \\
&= \frac{mk(k+1) + km(m-1)}{2} \\
&= \frac{km^2 + mk^2}{2}.
\end{aligned}$$

Testing this for the example a) yields: $\pi(T) = (3 \cdot 9 + 3 \cdot 9)/2 = 27$.

c) In both cases, the search depths (in the average case) depends linear on $\pi(T)$. Hence we need to minimize this value. We express π using $n' := n - 1 = km$, thereby ignoring the constant. Deriving yields $n' - n'^2/k^2 = 0$. Hence, we have $k = \sqrt{n'}$. If $n = k^2 + 1$, this is the optimal value. Otherwise, we simply round to the closest integer, which is optimal because of symmetry.

Problem 7-2

Consider the following two programs for searching elements in ordered arrays:

```
int binsearch(double v)
{
  int l, r, m;
  l = 1; r = n;
  while (l ≤ r) {
    m = (r + l)/2;
    if (v ≡ a[m]) return 1;
    if (v < a[m]) r = m−1; else l = m+1;
  }
  return 0;
}
```

```
int binsearch2(double v)
{
  int l, r, m;
  l = 1; r = n;
  while (r − l > 1) {
    m = (r + l)/2;
    if (v < a[m]) r = m − 1; else l = m;
  }
  if (a[l] ≡ v) return 1;
  if (a[r] ≡ v) return 1;
  return 0;
}
```

Determine the number of executions of **if**-statements in both problems when searching for an element v, in case of both, the successful and unsuccessful search.

Please give an exact solution for *binsearch* and an estimation of the form $f(n) + O(1)$ for for *binsearch2*.

Prerequisites:

- The array contains n *different* elements.
- For the successful search, each element is searched for with equal probability.
- For the unsuccessful search, v is chosen randomly, s.t., with probability $\frac{1}{n+1}$ is "in" one of the $n+1$ possible gaps.

Solution:

The first program was handled in the lecture. We therefore know the values $C^- = \lfloor \log(n+1) \rfloor + 2 - 2^{1-\{\log(n+1)\}}$ and $C^+ = \ldots$.

In an unsuccessful search, $2C^-$ **if**-instructions are executed, since there are C^- runs and each run contains two **if**-instructions.

In an successful search, we require $2C^+ - 1$ **if**-instructions, because the second **if**-instructions in the last run is not reached anymore.

Let us now consider *binsearch2*. If the current part of the array contains $n = r - l + 1$ elements, we denote the number of **if**-instructions by B_n. If there are less then three remaining elements, the **while** will not be entered at all. In the case of an unsuccessful search we therefore obtain $B_1 = B_2 = 2$.

For larger n, the **while**-loop is entered and one **if**-instructions is used. If $a[m] > v$, the algorithm searches on the left of the current element, and otherwise on the right. The remaining array thus becomes shorter, either $\lfloor n/2 \rfloor$ or $\lceil n/2 \rceil$. This yields the two recurrences

$$\overline{B}_n = \overline{B}_{\lceil n/2 \rceil} + 1$$
$$\underline{B}_n = \underline{B}_{\lfloor n/2 \rfloor} + 1$$

for $n > 2$ and $\overline{B}_n = \underline{B}_n = 2$ for $n \leq 2$. Obviously, we have

$$\underline{B}_n \leq B_n \leq \overline{B}_n.$$

A closed form can now be computed easily. We have $\underline{B}_{2^k} = k + 1$ for $k > 0$ and $\underline{B}_n = \lfloor \log(n) \rfloor + 1$ for $n > 2$, $\underline{B}_{2^k + r} = \underline{B}_{2^k}$ for $r < 2^k$.

Similar, we obtain $\overline{B}_{2^k} = k + 1$ for $k > 0$. However, here we have $\overline{B}_{2^k + r} = \overline{B}_{2^k} + 1$ for $0 < r < 2^k$. Thus, $\overline{B}_n = \lceil \log(n) \rceil + 1$ for $n > 2$.

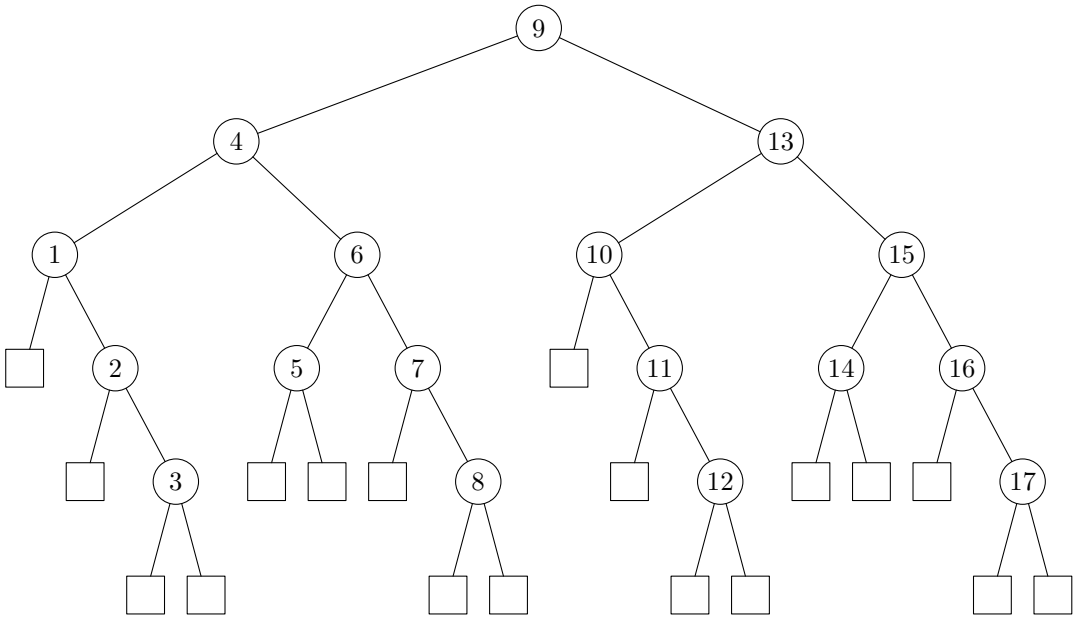
In conclusion, we therefore obtain $B_n = \log(n) + O(1)$.

Homework Assignment 7-1 (10 Points)

Consider the following algorithms for searching an element x in an ordered array a of length n. Here, m, is some fixed, but known integer.

Draw the search tree and compute internal and external path lengths for $n = 17$ and $m = 3$.

Solution:



Obviously, we obtain $\pi(T) = 44$ and $\xi(T) = 78$.

Homework Assignment 7-2 (10 Points)

The external path length of a search tree for binary search — and thus the average number of comparisons – was given in the lecture, but its correctness has not been proved. Prove it!

Hint: Find a recurrence formula for the number of comparisons.