

Analysis of Algorithms

Problem 7-1

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

```

1  i = 1;
   while ( a[i] <= x ){
3   if ( a[i] = x ) then return i;
     i = i + m;
5   if ( i > n ) return 0;
   }
7  for ( int j = i - 1; j >= max( 1, i - (m - 1) ); --j){
     if ( a[j] = x ) then return j;
9   if ( a[j] < x ) then return 0;
   }
11 return 0;
    
```

- Draw the search tree and compute the internal and external path length for $n = 10$ and $m = 3$.
- Determine C^+ and C^- for arbitrary m, k .
- What is, for given n , the best choice for m w.r.t. the running time?

Solution:

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

```

1 int binsearch( double v )
  {
3   int l, r, m;
   l = 1; r = n;
5   while ( l <= r ) {
       m = ( r + 1 ) / 2;
7   if ( v == a[m] ) return 1;
       if ( v < a[m] ) r = m - 1; else l
           = m + 1;
9   }
   return 0;
11 }

```

```

1 int binsearch2( double v )
  {
3   int l, r, m;
   l = 1; r = n;
5   while ( r - l > 1 ) {
       m = ( r + l ) / 2;
7   if ( v < a[m] ) r = m - 1; else l
           = m;
   }
9   if ( a[l] == v ) return 1;
   if ( a[r] == v ) return 1;
11  return 0;
   }

```

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 *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 $C^+ = \dots$

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*. For an array with n elements, let B_n denote the average number of **if**-instructions executed. If $n < 3$, the **while** is not entered at all. In the case of an unsuccessful search we therefore obtain $B_1 = B_2 = 2$. For a successful search, $B_1 = 1$, but $B_2 = \frac{1}{2}(1 + 2) = \frac{3}{2}$.

For $n \geq 3$, the **while**-loop is entered and one **if**-instruction is used per iteration of the **while**. 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 gives us two

recurrences:

$$\bar{B}_n = \bar{B}_{\lfloor n/2 \rfloor} + 1$$

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

for $n \geq 3$.

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.

```
1  int search( int x ) {
2    int l, r, i;
3    l = 1;
4    r = n;
5    while( r - l >= m ) {
6      i = ( l + r ) / 2;
7      if ( a[i] == x ) return 1;
8      if ( x < a[i] ) r = i - 1; else l = i + 1;
9    }
10   for( i = 1; i <= r; ++i ) {
11     if ( a[i] == x ) return 1;
12     if ( a[i] < x ) return 0;
13   }
14   return 0;
15 }
```

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)

Complete the analysis of the average number of times the `if`-instructions are executed in `binsearch2` for both a successful and an unsuccessful search (see Problem 7-2). First show that for all $k \geq 0$, the following hold:

1. $\lfloor \lfloor n/2^k \rfloor / 2 \rfloor = \lfloor n/2^{k+1} \rfloor$.

2. $\lceil \lceil n/2^k \rceil / 2 \rceil = \lceil n/2^{k+1} \rceil$.

Next, give a detailed analysis of the recurrences:

$$\bar{B}_n = \bar{B}_{\lfloor n/2 \rfloor} + 1$$

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

Solution:

For a fixed n and k , let

$$n = q_1 2^k + r_1$$

$$n = q_2 2^{k+1} + r_2$$

where $0 \leq r_1 < 2^k$ and $0 \leq r_2 < 2^{k+1}$. To establish the first identity, it is sufficient to show that $\lfloor q_1/2 \rfloor = q_2$. Therefore let $q_1 = 2q_3 + r_3$, where $0 \leq r_3 \leq 1$. This lets us write n as $q_3 2^{k+1} + r_3 2^k + r_1$. If $r_3 = 0$, then

$$n = q_3 2^{k+1} + r_1.$$

By the uniqueness of the quotient and remainder, we must have $q_3 = q_2$ and $r_1 = r_2$. Hence in this case $\lfloor q_1/2 \rfloor = q_2$. Therefore assume that $r_3 = 1$. But then $n = q_3 2^{k+1} + 2^k + r_1$ and since $r_1 < 2^k$, we have $2^k + r_1 < 2^{k+1}$. Again by the uniqueness of quotient and remainder, we must have $q_3 = q_2$ and $r_2 = 2^k + r_1$. Thus in this case too $\lfloor q_1/2 \rfloor = q_3 = q_2$. The proof for the “ceiling” version is analogous.

Now by the result that we have established, we can write \bar{B}_n as:

$$\bar{B}_n = k + \bar{B}_{\lceil n/2^k \rceil},$$

where k is the smallest non-negative integer such that $\lceil n/2^k \rceil = 1$. This amounts to finding the smallest k such that $n \leq 2^k$, which implies that $k = \lceil \log n \rceil$. Thus $\bar{B}_n = \lceil \log n \rceil + \bar{B}_1$.

Similarly, we may write B_n as:

$$B_n = k + B_{\lfloor n/2^k \rfloor},$$

where k is the smallest non-negative integer such that $\lfloor n/2^k \rfloor = 1$. This is just the smallest k such that $n < 2^k$ and this equals $\lfloor \log n \rfloor + 1$.