Jordan Allard

Algorithms, Fall '23, Activity 5: Recurrences

Consider the following pseudo code, which we will use to search for a number x in an array $A[1 \ldots m]$ by calling BSearch(A,x,1,m):

```
BSearch(array A, integer x, integer left, integer right):
    if left>right: return "Not found"
    m = (left+right)/2 (rounded down)
    if A[m]=x: return "Found at position m"
    if A[m]>x: return BSearch(A, x, left, m-1)
    if A[m]<x: return BSearch(A, x, m+1, right)
```

non recurrive: $O(1)$ $O(1)$ $O(1)$ $\rightarrow O(3) \rightarrow O(1) \rightarrow c$

recursive: $T(\frac{n}{2})$

Notice that the function operates on input of size $n = right - left + 1$.

(a) Set up a recurrence for BSearch, using $T(n)$ to denote the maximum number of steps the function makes on an input of size n. In particular:

- Next to the pseudo code above, highlight its non-recursive parts and estimate their running time in big-Oh notation.
- Next to the pseudo code above, highlight the recursive parts and estimate their running times using the function $T()$.
- Give the recurrence for $T(n)$. Do not worry about the rounding. Do not forget the base case.

$$T(n) = \begin{cases} c & \text{if } n = 0 \\ T(n/2) + c & \text{if } n > 0 \end{cases}$$

(b) Analyze the recurrence using either the unrolling technique, or the math induction.

$$T(n) = T(\tfrac{n}{2}) + c = (T(n/4) + c) + c = T(\tfrac{n}{4}) + 2c$$

$$= (T(n/8) + c) + 2c = T(n/8) + 3c$$

For k iterations:

$$T(n/2k) + kc$$

When $k = \log n$: $T(\frac{n}{2^{\log n}}) + (\log n)c = T(1) + c \log n = O(\log n)$