Computing Science 204 (B1)
Finding the Smallest and Largest Keys
Friday, July 21

Example. *Finding the Max and Min*

Finding the smallest and largest keys in a list of size n separately takes, in the worst case, $W(n) = 2(n-1)$ comparisons of list entries. However, we can improve this by pairing the keys, and find which key in each pair is smaller; this can be done with about $\frac{n}{2}$ comparisons. We can then find the smallest of all the smaller keys with about $\frac{n}{2}$ comparisons and the largest of all the larger keys with about $\frac{n}{2}$ comparisons. In this way, we find both the smallest and the largest keys with only about $\frac{3n}{2}$ comparisons.

This algorithm is implemented below; where the call `MaxMin(0,N-1)` to the recursive procedure finds the largest and smallest elements in the array

$$A[0],\ A[1],\ A[2],\ \ldots,\ A[N-1].$$

The largest entry is returned in `Large` and the smallest entry is returned in `Small.`

```
MaxMin(L,R)
int L,R;
{
   int M,LargeLeft,LargeRight,SmallLeft,SmallRight;
   if (R - L == 0){
       Large = A[L]; Small = A[R];
   {
   else if (R - L == 1){
       if (A[L] > A[R]){
           Large = A[L]; Small = A[R];
       }
       else{
           Large = A[R]; Small = A[L];
       }
   }
   else{
       M = (R + L)/2;
       MaxMin(L,M);
       LargeLeft = Large; SmallLeft = Small;
       MaxMin(M+1,R);
       LargeRight = Large; SmallRight = Small;
       if (LargeLeft > LargeRight)
           Large = LargeLeft;
       else
           Large = LargeRight;
       if (SmallLeft < SmallRight)
           Small = SmallLeft;
       else
           Small = SmallRight;
   }
}
```

For $N \geq 1$, let T_N be the number of comparisons needed for the algorithm to find the maximum and minimum entries in an array of size N.

(a) Find T_1 and T_2, and show that T_N satisfies the recurrence relation $T_N = T_{\lfloor \frac{N}{2} \rfloor} + T_{\lfloor \frac{N+1}{2} \rfloor} + 2$ for $N > 2$.

(b) Solve this recurrence relation for the case that N is a power of 2.

proof.

(a) In the call to `MaxMin(0,N-1)` when $N = 1$, we have $L = 0$, and $R = N - 1 = 0$, so that $R - L = 0$. In this case there are no comparisons made, only the assignments `Large = A[0]` and `Small = A[0]`, then the algorithm returns. Therefore, $T_1 = 0$.

When $N = 2$, then $L = 0$, and $R = N - 1 = 1$, so that $R - L = 1$. In this case, there is only one comparison made, namely `A[0] > A[1]`, and two assignments are made depending on the outcome of this comparison, and then the algorithm returns. Therefore, $T_2 = 1$.

However, when $N > 2$, the assignment $M = \left\lfloor \dfrac{N-1}{2} \right\rfloor$ is made, and the algorithm then calls itself recursively on the two sub-arrays

$$\underbrace{A[0], A[1], \ldots, A[M]}_{\ell_L} \qquad \text{and} \qquad \underbrace{A[M+1], A[M+2], \ldots, A[N-1]}_{\ell_R}$$

and makes two more comparisons `LargeLeft > LargeRight` and `SmallLeft < SmallRight`. Note that here $\ell_L = M + 1$ and $\ell_R = N - (M + 1)$ are the sizes of the left and right subarrays, respectively. So, the number of comparisons needed for the algorithm to find the maximum and minimum entries in an array of size N satisfies the recurrence relarion

$$\begin{aligned} T_N &= T_{\ell_L} + T_{\ell_R} + 2 \\ T_1 &= 0 \\ T_2 &= 1 \end{aligned}$$

where $\ell_L = \left\lfloor \dfrac{N-1}{2} \right\rfloor + 1$, and $\ell_R = N - \ell_L$. In order to find ℓ_L and ℓ_R, we consider two cases.

(i) If N is even, say $N = 2k$, then $M = \left\lfloor \dfrac{N-1}{2} \right\rfloor = \left\lfloor \dfrac{2k-1}{2} \right\rfloor = k - 1$ so that $M + 1 = k$, therefore

$$\ell_L = k = \left\lfloor \frac{N}{2} \right\rfloor \qquad \text{and} \qquad \ell_R = N - k = k = \left\lfloor \frac{N+1}{2} \right\rfloor.$$

(ii) If N is odd, say $N = 2k + 1$, then $M = \left\lfloor \dfrac{N-1}{2} \right\rfloor = \left\lfloor \dfrac{2k}{2} \right\rfloor = k$ so that $M + 1 = k + 1$, therefore

$$\ell_L = k + 1 = \left\lfloor \frac{N+1}{2} \right\rfloor \qquad \text{while} \qquad \ell_R = N - (k+1) = k = \left\lfloor \frac{N}{2} \right\rfloor.$$

Thus, in either case, T_N satisfies the recurrence relation

$$\begin{aligned} T_N &= T_{\lfloor \frac{N}{2} \rfloor} + T_{\lfloor \frac{N+1}{2} \rfloor} + 2 \\ T_1 &= 0 \\ T_2 &= 1 \end{aligned}$$

for $N > 2$.

(b) If $N = 2^k$, where $k > 1$, then

$$\left\lfloor \frac{N}{2} \right\rfloor = 2^{k-1} \qquad \text{and} \qquad \left\lfloor \frac{N+1}{2} \right\rfloor = \left\lfloor 2^{k-1} + \frac{1}{2} \right\rfloor = 2^{k-1},$$

and therefore, if we let $b_k = T_{2^k}$ for $k \geq 1$, then b_k satisfies the recurrence relation

$$\begin{aligned} b_k &= 2\,b_{k-1} + 2 \\ b_1 &= 1 \end{aligned}$$

for $k > 1$. We can solve this in the usual manner, for example, for $k > 1$,

$$\begin{aligned}
b_2 &= 2\,b_1 + 2 \\
b_3 &= 2\,b_2 + 2 = 2^2 b_1 + 2^2 + 2 \\
b_4 &= 2\,b_3 + 2 = 2^3 b_1 + 2^3 + 2^2 + 2 \\
&\vdots \\
b_k &= 2^{k-1} b_1 + \left[2^{k-1} + 2^{k-2} + \cdots + 2^2 + 2\right] \\
b_k &= 2^{k-1} b_1 + 2\left[2^{k-2} + 2^{k-3} + \cdots + 2^2 + 2 + 1\right] \\
b_k &= 2^{k-1} b_1 + 2\left(\frac{2^{k-1} - 1}{2 - 1}\right) \\
b_k &= 2^{k-1} b_1 + 2 \cdot 2^{k-1} - 2 \\
b_k &= 3 \cdot 2^{k-1} - 2
\end{aligned}$$

and therefore, $T_N = \frac{3}{2} N - 2$ for $N = 2^k$, where $k \geq 1$.

$\square$

Note: It is left as an exercise to show that the number of comparisons of keys in the general case (N not a power of 2) is given by

$$T_N = \begin{cases} \dfrac{3n}{2} - 2 & \text{if } n \text{ is even} \\[2ex] \dfrac{3n}{2} - \dfrac{3}{2} & \text{if } n \text{ is odd} \end{cases}$$

and that this algorithm is optimal for the worst case performance.