CS29003 Algorithms Laboratory

Assignment No: 1

Date: 15–January–2019

Polynomial and Logarithmic Running Times

A blackbox BB1 has an array A of size N, populated by all the integers in the range $0,1,2,\ldots,N-1$. The array is almost sorted except that two of its elements are swapped with one another, that is, there exist indices L,R satisfying $0 \leqslant L < R \leqslant N-1$ such that the array locations store the following integers.

i	0	1	2	$\cdots$	$L-1$	L	$L+1$	$L+2$	$\cdots$	$R-1$	R	$R+1$	$R+2$	$\cdots$	$N-1$
$A[i]$	0	1	2	$\cdots$	$L-1$	R	$L+1$	$L+2$	$\cdots$	$R-1$	L	$R+1$	$R+2$	$\cdots$	$N-1$

In other words, $A[i] = i$ for $i \neq L,R$, whereas $A[L] = R$ and $A[R] = L$. Your task is to determine the indices L and R. Since BB1 hides the array from you, you need to make appropriate blackbox queries in order to solve the problem. Depending upon what type of queries BB1 likes to entertain, you play the following games with BB1. $N = 10^9 = 1,000,000,000$ throughout this assignment.

Game 0: In this case, BB1 allows queries of the form `valquery`(q) which returns the array element $A[q]$ for a query q in the range $[0,N-1]$. You make a linear search with queries $q = 0,1,2,\ldots$ in this sequence. You see $A[q] = q$ for $0 \leqslant q \leqslant L-1$. As soon as you make the query $q = L$, you receive the response $R > q$, and your problem is solved. This algorithm requires you to make $L+1$ queries, and has $\mathrm{O}(N)$ running time.

Game 1: Let us now see whether a divide-and-conquer approach can help you. Suppose that $[l,r]$ is the current search interval (initially $l = 0$ and $r = N-1$). Take $q = \lfloor (l+r)/2 \rfloor$. Make a `valquery` on this q. If the return value is different from q, you are done. If however $A[q] = q$, you get very little help from the query, because the interval $[L,R]$ may be entirely to the left of q or entirely to the right of q, or the query q is inside the interval $[L,R]$. You are then forced to make two recursive calls: the first on the search interval $[l,q-1]$, and the second on the search interval $[q+1,r]$. At least one of these recursive calls will eventually solve your problem. Now, the running time satisfies

$$T(N) \approx 2T(N/2) + \mathrm{O}(1) = \mathrm{O}(N).$$

In other words, this divide-and-conquer algorithm is no better than the linear search of Game 0. You should have a practical implementation of the recursive procedure, that is, if the first recursive call already reveals the desired solution (L,R), there is no necessity to make the second recursive call.

Game 2: In this game, BB1 is not willing to disclose you the array elements. It will however provide you with better hints so that you can come up with a logarithmic-time binary-search algorithm. Let $q \in [0,N-1]$ be a query. Define the following quantities:

$$\delta_L(q) = |L-q|, \qquad \delta_R(q) = |R-q|, \qquad \delta(q) = \delta_L(q) - \delta_R(q).$$

If q is understood form the context, we use the shortcuts $\delta_L, \delta_R, \delta$ for these quantities. In words, δ_L is the distance of L from q, and δ_R is the distance of R from q. The third quantity δ captures the relative distance of q from L and R. Indeed, $\delta = 0$ if q is equidistant from L and R, $\delta < 0$ if q is closer to L than to R, and $\delta > 0$ if q is closer to R than to L. Define the *sign* of δ as

$$\mathrm{sgn}(\delta) = \begin{cases} 0 & \text{if } \delta = 0, \\ -1 & \text{if } \delta < 0, \\ +1 & \text{if } \delta > 0. \end{cases}$$

In this game, the blackbox BB1 allows you to make a maximum of $\lceil \log_2 N \rceil$ sign queries `sgnquery`(q) to return $\text{sgn}(\delta)$ for the queried q. In addition, it allows you to make a single query `delquery`(q) which returns the absolute value $|\delta(q)| = |\delta_L(q) - \delta_R(q)|$

Let us see how these queries let you identify L and R. Let $M = (L+R)/2$. If $L+R$ is even, then M is an integer. Otherwise, $L+R$ is an integer plus $1/2$. For $q = M$ (provided that M is an integer), the return value of the sign query on q is 0. For any $q < M$, the sign query on q returns -1, whereas for any $q > M$, the sign query on q returns $+1$. The array $S[0 \ldots N-1]$ of signs looks like $[-1,-1,\ldots,-1,0,+1,+1,\ldots,+1]$ if $L+R$ is even, or $[-1,-1,\ldots,-1,+1,+1,\ldots,+1]$ if $L+R$ is odd. You cannot construct the entire array S. That is not needed either, because S is a sorted array. You can make a binary search in S in order to identify the index u of the first element > -1 in S. You have $u = \begin{cases} (L+R)/2 & \text{if } L+R \text{ is even,} \\ (L+R+1)/2 & \text{if } L+R \text{ is odd.} \end{cases}$ Now, a delta query on the index 0 returns $v = R - L$. Solve these two equations to get the values of L and R.

Game 3: In this game, BB1 does not allow value, sign, or delta queries. Instead you can make only queries of the form `sumquery`(q) on $q \in [0, N-1]$, which returns the sum $\delta_L(q) + \delta_R(q)$. A little more than $\lceil \log_2 N \rceil$ sum queries are allowed. Devise and implement an algorithm to find L, R in this game.

Note: This game can be played by making only $O(1)$ sum queries.

How to Handle the Blackbox

The blackbox is provided to you as a precompiled binary file `BB1.o`. Download the file depending on your compiler (`gcc` or `g++`).

In the beginning of your code, include the following lines to keep your compiler happy.

```
extern void registerme ( );
extern void startgame ( int );
extern int valquery ( int );
extern int sgnquery ( int );
extern int delquery ( int );
extern int sumquery ( int );
extern void checksoln ( int, int );
```

Your `main()` function would look like the following.

```
registerme();
playgame0();
playgame1();
playgame2();
playgame3();
```

You must first call `registerme()` before doing anything else. For $i \in \{0,1,2,3\}$, the code for `playgame`i`()` would be as follows.

```
startgame(i);
... Your code for finding L and R ...
checksoln(L,R);
```

Make appropriate queries in these functions as specified in the above function prototypes. You do not have to print anything in your code. All the necessary printing will be done by the blackbox calls. The final call `checksoln(L,R)` requires you to supply $L < R$. All indices passed (L, R, and the queries q) must be in the range $[0, N-1]$.

Compile your code as:

```
gcc mycode.c BB1.o
g++ mycode.cpp BB1.o
```

Sample output

```
+++ Game 0 started
    Your solution: L = 922848586, R = 965931375
    Congrats. Your solution is correct.
    You made 922848587 value queries

+++ Game 1 started
    Your solution: L = 575234482, R = 716015846
    Congrats. Your solution is correct.
    You made 575234499 value queries

+++ Game 2 started
    Your solution: L = 246930589, R = 965824074
    Congrats. Your solution is correct.
    You made 30 sign queries
    You made 1 delta query

+++ Game 3 started
    Your solution: L = 842171917, R = 842273447
    Congrats. Your solution is correct.
    You made 16 sum queries
```

Submit a single C/C++ source file. Do not use global/static variables.