CSOR:4246 Assignment 1

Vibhuti Mahajan (UNI: vm2486)

26 September 2016

1 Solution

<u>Algorithm</u>

- The array given is sorted, so start searching for the key from the first element $A[1]$. Compare the search term with the key and return the index if both are same.
- Increment search index by a factor of 2 unless a number greater than key or an error message (∞) is encountered.
- Let 2^k be the index of the term which is greater than key or where ∞ is encountered. Perform binary search on the subarrray $A[2^{k-1}, 2^k]$

<u>Pseudo Code</u>

Binary-Search(A, left, right, key)

```
if(left==right) then
   if A[left]==key then
      return left
   else return no
   end if
else
   mid = left +⌈(right−left)/2⌉
   if A[mid]==key then
      return mid
   else
      if A[mid]¡key then
         left=mid+1
      else
         right = mid-1
      end if
      Binary-Search(A, left, right, key)
   end if
end if
```

Search(A,key)

```
i=key
while A[i]¡=key, do
   if A[i]==key
      return i
   else
      i = 2 × i
   end if
```

end while
Binary-Search(A, $\frac{i}{2}$, i, key)
<u>Correctness of Algorithm</u>
The algorithm returns the index of the element if the key is found or invokes binary search for the element in the subarray $A[2^{k-1}, 2^k]$ in case key is not found. This is correct because:

1. the array is sorted hence key will not lie in the subarray $A[1, 2^{k-1}]$ which is already checked by the earlier iterations of the while loop

2. Neither can key lie in an array element greater than $A[2^k]$ because then it will violate the the condition for the while loop.

Hence, the element must lie in $A[2^{k-1}]$. Further, binary search is performed on $A[2^{k-1}, 2^k]$ which itself is a correct algorithm as shown in the class.
<u>Running Time</u>

1. Binary search takes at most $\mathcal{O}(\log \frac{n}{2}) = \mathcal{O}(\log n)$

2. While loop is repeated at most $\log_2 n$ times.

3. All statement in while loop takes constant time. Hence running time for the while loop is $\mathcal{O}(\log n)$.

4. Assignment of key to i is a primitive calculation and takes constant time.

$\therefore$ Running time $= c_1 \log n + c_2 \log n + c_3 = \mathcal{O}(\log n)$

2 Solution

<u>Algorithm</u>

- (Assuming $\frac{k}{2}^{th}$ element exists in both the lists)Since the lists are sorted, we start by comparing the $\frac{k}{2}^{th}$ element of both the lists . Remove the elements preceding $\frac{k}{2}^{th}$ element from the smaller element among the 2 lists.

- If one of the lists has less than $\frac{k}{2}$ elements then we compare the last element of that list to $\frac{k}{2}^{th}$ element of the other and discard the smaller values.

- Update lists and the value of k after discarding the smaller elements. If r elements are discarded then value of k is updated as k-r.

- Repeat the above three steps with the updated value of k till k =1.

-

<u>Pseudo Code</u>
Rank(A,a_1,n,B,b_1,m,k)

```
if m+n<k then
    return no
else if n==0 then
    return B[b_1+k-1]
else if m==0 then
    return A[a_1k-1]
else if k==1 then
    if A[a_1]<B[b_1] then
       return A[a_1]
    else
       return B[b_1]
    end if
else
    if n < ⌊k/2⌋ then
       if A[n] < B[⌊k/2⌋] then
          k=k-n
          a_1 = a_1 + n
          n=n-n
       else
          k = k - ⌊k/2⌋
          b_1 = b_1 + ⌊k/2⌋
          m = m - ⌊k/2⌋
       end if
    else if m < ⌊k/2⌋ then
       if B[m] < A[⌊k/2⌋] then
          k=k-m
          b_1 = b_1 + m
          m=m-m
       else
          k = k - ⌊k/2⌋
          a_1 = a_1 + ⌊k/2⌋
          n = n - ⌊k/2⌋
       end if
    else if A[⌊k/2⌋] < B[⌊k/2⌋] then
       k = k - ⌊k/2⌋
       a_1 = a_1 + ⌊k/2⌋
       n = n - ⌊k/2⌋
    else
       k = k - ⌊k/2⌋
       b_1 = b_1 + ⌊k/2⌋
       m = m - ⌊k/2⌋
    end if
end if
Rank(A,a_1,n,B,b_1,m,k)
```

Initial Call : Rank(A,1,length(A), B, 1, length(B),k)

Correctness of Algorithm

- The algorithm terminates if :
    1. the sum of number of elements in both lists exceed k. No solution in this case.
    2. either of the lists is exhausted. Then getting the k-th ranked element is a constant time operation in a sorted array.

3. If k=1 then we just compare the first elements of the considered sub-arrays. this is also a constant time computation.

- If we discard p smaller elements then we desire to get $k - p^{th}$ ranked element from the remaining union of lists($p < k$). Our algorithm works on this principle and it is correct because:

   1. Let the number of elements which we discard at a step be p_i and S be the number of elements already discarded. Essentially, $S = \Sigma_{n=1}^{i-1} p_n$. So we have, $S + p_i <= k$.
   2. The discarded elements belong to one exclusive list.
   3. Since the list itself is sorted hence all discarded elements are smaller than the element which was used to compare to the required element of the other list.
   4. From 1. above, it is not possible to discard potential k-th ranked element without the algorithm terminating.

- For the ideal case, the elements discarded at every step is half of what was discarded before. Hence, total elements discarded= $\frac{k}{2} + \frac{k}{4} + \frac{k}{8} ... + 1 = k$(sum of a G.P.). Hence, the algorithm returns k-th ranked element. Or generally, let it take p steps to terminate this algorithm and at every step i, a_i elements are discarded. Then we have: $a_1 + a_2 + ... + a_p = k$

Running Time

- Every recurrence takes constant time as the function rank just contains primitive computations.
- Worst case arises if we have to find the last rank. In that case, the number of times function rank is called can be at most $\log_2 n + \log_2 m$.
- Hence the running time of the algorithm is $\mathcal{O}(\log n + \log m)$

3 Solution

Let X, Y be the 2 n-bit integers whose multiplication we are interested in. We can write:

$$X = X_H.10^{\frac{2n}{3}} + X_M.10^{\frac{n}{3}} + X_L$$

$$Y = Y_H.10^{\frac{2n}{3}} + Y_M.10^{\frac{n}{3}} + Y_L$$

Hence,

$$X \times Y = (X_H Y_H)10^{\frac{4n}{3}} + (X_M Y_H + X_H Y_M)10^n +$$

$$(X_H Y_L + X_M Y_M + X_L Y_H)10^{\frac{2n}{3}} + (X_M Y_L + X_L Y_M)10^{\frac{n}{3}} + X_L Y_L$$

Let T(n) be the time taken to multiply 2 n-bit integers. We have broken down this to 9 multiplication of 2 $\frac{n}{3}$-bit integers.

$(X_H Y_H)10^{\frac{4n}{3}}$	$T(\frac{n}{3}) + c_1 n$ (shifting $\frac{4n}{3}$ places takes linear time)
$(X_M Y_H + X_H Y_M)10^n$	$2T(\frac{n}{3}) + c_2 n + c_3 n$(linear time for addition of 2 $\frac{2n}{3}$-bit integers and linear time for shifting n places)
$(X_H Y_L + X_M Y_M + X_L Y_H)10^{\frac{2n}{3}}$	$3t(\frac{n}{3}) + c_4 n + c_5 n$ (linear time for addition of 3 $\frac{2n}{3}$-bit integers)
$(X_M Y_L + X_L Y_M)10^{\frac{n}{3}}$	$2T(\frac{n}{3}) + c_6 n + c_7 n$
$X_L Y_L$	$T(\frac{n}{3})$

$\therefore$ total running time = $T(n) = 9T(\frac{n}{3}) + cn$ where $c = c_1 + c_2 + c_3 + c_4 + c_5 + c_6 + c_7$.
By using the master theorem, $a = 9, b = 3, k = 1$ & $a > b^k$ $\therefore$ Running Time = $\mathcal{O}(n^2)$

1. Pseudo-code:
   Multiplication-6(X,Y,n)
   Xq1=X/$10^{\frac{2n}{3}}$
   Xr1=X%$^{\frac{2n}{3}}$ #modulus gives the remainder
   Xq2=Xr1/$10^{\frac{n}{3}}$
   Xr2=Xr1%$10^{\frac{n}{3}}$
   X_H=Xq1
   X_M=Xq2
   X_L=Xr2
   Yq1=Y/$10^{\frac{2n}{3}}$
   Yr1=Y%$^{\frac{2n}{3}}$
   Yq2=Yr1/$10^{\frac{n}{3}}$
   Yr2=Yr1%$10^{\frac{n}{3}}$
   Y_H=Yq1
   Y_M=Xq2
   Y_L=Xr2
   return $X_H Y_H.(10^{\frac{4n}{3}} - 10^n - 10^{\frac{2n}{3}}) + X_M Y_M(10^{\frac{2n}{3}} - 10^n - 10^{\frac{n}{3}}) + X_L Y_L(-10^{\frac{2n}{3}} - 10^{\frac{n}{3}}) + (X_H + X_M)(Y_H + Y_M)10^n + (X_H + X_L)(Y_H + Y_L)10^{\frac{2n}{3}} + (X_M + X_L)(Y_M + Y_L)10^{\frac{n}{3}}$

   Correctness:

   - Dividing the n-bit integers by $10^{\frac{2n}{3}}$ gives the top $\frac{n}{3}$ bits of the concerned integer as the quotient. The remainder is the remaining $\frac{2n}{3}$ bits in their original order. Further dividing the remainder by $10^{\frac{n}{3}}$ gives the middle $\frac{n}{3}$ bits as the quotient and remainder as the lowest $\frac{n}{3}$ bit integers.
     (a) $(X_M Y_H + X_H Y_M) = (X_M + X_H)(Y_M + Y_H) - X_M Y_M - X_H Y_H$
     (b) $(X_H Y_L + X_L Y_H) = (X_H + X_L)(Y_H + Y_L) - X_H Y_H - X_L Y_L$
     (c) $(X_M Y_L + X_L Y_M) = (X_M + X_L)(Y_M + Y_L) - X_M Y_M - X_L Y_L$

     Hence, $X \times Y$ can be written as additions or subtraction of 5 multiplication of 2 $\frac{n}{3}$-bit integers (can be atmost $\frac{n}{3} + 1$).

$$\therefore X \times Y = X_H Y_H.10^{\frac{4n}{3}} + a.10^n + b.10^{\frac{2n}{3}} + X_M Y_M.10^{\frac{2n}{3}} + c.10^{\frac{n}{3}} + X_L Y_L$$
$$= X_H Y_H.(10^{\frac{4n}{3}} - 10^n - 10^{\frac{2n}{3}}) + X_M Y_M(10^{\frac{2n}{3}} - 10^n - 10^{\frac{n}{3}}) + X_L Y_L(-10^{\frac{2n}{3}} - 10^{\frac{n}{3}}) +$$
$$(X_H + X_M)(Y_H + Y_M)10^n + (X_H + X_L)(Y_H + Y_L)10^{\frac{2n}{3}} + (X_M + X_L)(Y_M + Y_L)10^{\frac{n}{3}}$$

2. Multiplication takes $T(\frac{n}{3})$ time because asymptotically, $\frac{n}{3} + 1 = \frac{n}{3}$. Addition and subtraction take linear time , hence $\mathcal{O}(n)$ because we can have at most 2n digits.
   Hence, running time= $T(n) = 6T(\frac{n}{3}) + cn$
   Using master theorem, $a = 6, b = 3, k = 1$& $a > b^k$
   $\therefore T(n) = \mathcal{O}(n^{\log_3 6}) = \mathcal{O}(n^{1.63})$.
   For 2-split integer multiplication, the running time is $\mathcal{O}(n^{\log_2 3}) = \mathcal{O}(n^{1.58})$&$n^{1.58} = \Omega(n^{1.63})$ Hence, 2-split integer multiplication is faster/ more-efficient.

3. For 5 multiplications,
   $T(n) = 5T(\frac{n}{3}) + cn$
   Using master theorem, $a = 5, b = 3, k = 1$ & $a > b^k$
   $\therefore T(n) = \mathcal{O}(n^{\log_3 5}) = \mathcal{O}(n^{1.46})$&$n^{1.46} = \Omega(n^{1.58})$
   Hence, 5 multiplication for a 3-split integer multiplication is faster than a 2-split multiplication.

4. (I am aware of this concept from my undergrad study in algorithms)
   Pseudo Code

Multiplication-5(X,Y,n)
#Initisalisation of X_H, X_M, X_L, Y_H, Y_M and Y_L is the same as the 6-multiplication problem
$a = X_H Y_H$
$b = X_L Y_L$
$c = (X_H + X_M + X_L)(Y_H + Y_M + Y_L)$
$d = (X_H - X_M + X_L)(Y_H - Y_M + Y_L)$
$e = (X_H - 2X_M + 4X_L)(Y_H - 2Y_M + 4Y_L)$
return $a10^{\frac{4n}{3}} + \frac{3a+3b+2c-6d+e}{6}10^n + \frac{-2a-2b+c+d}{2}10^{\frac{2n}{3}} + \frac{-3a-3b+c+3d-e}{6}10^{\frac{n}{3}} + b$

Let

(a) $a = X_H Y_H$

(b) $b = X_L Y_L$

(c) $c = (X_H + X_M + X_L)(Y_H + Y_M + Y_L) = X_H Y_H + X_L Y_L + (X_H Y_M + X_M Y_H) + (X_M Y_L + X_L Y_M) + (X_H Y_L + X_L Y_H + X_M Y_M) = a + b + x + y + z$, where:
   - $x = (X_H Y_M + X_M Y_H)$
   - $y = (X_M Y_L + X_L Y_M)$
   - $z = (X_H Y_L + X_L Y_H + X_M Y_M)$

(d) $d = (X_H - X_M + X_L)(Y_H - Y_M + Y_L) = a + b - x - y - z$

(e) $e = (X_H - 2X_M + 4X_L)(Y_H - 2Y_M + 4Y_L) = a + 16b - 2x - 8y - 4z$

a,b,c,d,e are the 5 multiplication terms. We have to express x, y, z as addition and/or subtractions of a, b, c, d and e. Solving the three linear equation for three variables, we get:

- $x = \frac{3a+3b+2c-6d+e}{6}$
- $y = \frac{-3a-3b+c+3d-e}{6}$
- $z = \frac{-2a-2b+c+d}{2}$

(a) The multiplication terms are multiplications of 2 integers of size at most $\frac{n}{3} + 1 (= \frac{n}{3}$, asymptotically)

(b) Multiplication and division of an n-bit integer with a single digit integer is linear corresponding to n..

(c) Hence, the running time for the above algorithm can be written as : $T(n) = 5T(\frac{n}{3}) + cn = \mathcal{O}(n^{1.46})$

4 Solution

<u>Pseudo Code</u>
swap(A,i,j)
if A[i]¿A[j] then
    temp=A[j]
    A[j]=A[i]
    A[i]=temp
end if
return (A)

modify-sort(A,left, right)

f	g	$\mathcal{O}$	o	Ω	ω	Θ
$\log^5 n$	$10\log^3 n$			Yes	Yes	
$n^2\log(2n)$	$n\log n$			Yes	Yes	
$\sqrt{\log n}$	$\log\log n$			Yes	Yes	
$n^2+n^{\frac{1}{3}}$	$n^2\log n+n^{\frac{5}{2}}$	Yes	Yes			
$\sqrt{n}+1500$	$n^{\frac{1}{3}}+\log n$			Yes	Yes	
$\frac{3^n}{n^2}$	$2^n\log n$			Yes	Yes	
$n^{\log n}$	2^n	Yes	Yes			
2^n	$\frac{3^n}{n^{\log n}}$	Yes	Yes			
n^n	$n!$			Yes	Yes	
$\log^n n$	$\log n!$			Yes	Yes	

Table 1: Solution Problem 5

if (right-left)¡1 then
  return (A)
else if (right-left)==1 then
  return swap(A, left, right)
else
  modify-sort(A,left, left+$\lfloor\frac{2(right-left)}{3}\rfloor$)
  modify-sort(A, right-$\lfloor\frac{2(right-left)}{3}\rfloor$right)
  modify-sort(A,left, left+$\lfloor\frac{2(right-left)}{3}\rfloor$)

Initial call: modify-sort(A, 1, n)

- The swapping of 2 numbers is a primitive computation and takes constant time to execute that. Hence swap is a constant time function.
- If $T(n)$ is the time taken for the algorithm to run on n elements, then each recursive step takes $T(\frac{2n}{3})$ time.
- There are 3 recursions in each recursive call.

Hence, $T(n) = 3T(\frac{2n}{3}) + c$
Using the master theorem, $a = 3, b = \frac{3}{2}, k = 0 \therefore a > b^k$
Thus time taken to run the algorithm $= T(n) = \mathcal{O}(\log_{\frac{3}{2}} 3) = \mathcal{O}(n^{2.71})$. Since the time taken is greater than insertion sort, merge sort etc, I will probably not use the algorithm in future.

5 Solution

check table above