Algorithmn HW4

5140379032 JIN YI FAN

Problem 6.35

Let two pointers scan the array sperately for neg and pos numbers, then swap

Require: $A[]$, a set of integers

Ensure: negative integers left, positive right

$n \leftarrow 0$
$p \leftarrow (A.size - 1)$
while n and p are not crossed over **do**
    while $A[n] < 0$ **do** ▷ find the first non-negative number
        $n \leftarrow n + 1$
    end while
    while $A[p] \geq 0$ **do** ▷ find the first negative number
        $p \leftarrow p - 1$
    end while
    SWAP$(A[n], A[p])$
end while
return $A[]$

Problem 6.2

The recursive expressions of SLOWMINMAX are changed as:

$$C(n) = \begin{cases} 0 & n = 1 \\ 2 \cdot C(n/2) + 2 & n > 1 \end{cases}$$

Given that $n = 2^k$ and $B(k) = C(2^k)$, we have:

$$B_0 = C(2^0) = 0 \tag{1}$$
$$B_1 = C(2^1) = 2 \cdot C(2^0) + 2 = 2 \tag{2}$$
$$B_k = C(2^k) = 2 \cdot C(2^{k-1}) + 2 = 2 \cdot B_{k-1} + 2 \ (k > 1) \tag{3}$$

$\therefore B_k + 2 = 2 \cdot (B_{k-1} + 2) \Rightarrow B_k = 4 \cdot 2^{k-1} - 2 \ (k > 1)$
and the equation stands when $k = 0$ and $k = 1$
$\therefore \forall k \geq 0, B_k = 2^{k+1} - 2$
$\therefore C(n) = B(\log_2(n)) = 2 \cdot 2^{\log_2(n)} - 2 =$ **2n-2**
The number of comparison in this case is greater than in MINMAX
because the recursion tree in this case is 1-step deeper than that in MINMAX, which contains an extra 2^{k-1}, or say $n/2$ comparisons.

Problem 6.52

Require: an array of n integers $A[]$
Ensure: the second largest element

```
function scdMin(low, high, A[])
    if high − low ≤ 1 then                                        ▷ end situation
        return (min(A[low], A[high]), max(A[low], A[high]))
    else                       ▷ divide into 2 and get the smallest 2 numbers every time
        mid ← ⌊low + (high − low)/2⌋
        (can1l, can1h) ← scdMin(low, mid, A)
        (can2l, can2h) ← scdMin(mid + 1, high, A)
        return (min(can1l, can2l), max(can1l, can2l))
    end if
end function
(l, h) ←scdMin(0, A.size − 1, A)                                ▷ start of main function
return h
```

Counting inversions

Require: an array of n integers $a[]$
Ensure: the number of inversion in $a[]$

```
function invers(A[], res)
    if A.size ≤ 2 then
        if A[A.size − 1] < A[0] then
            res ← res + 1
            swap(A[1], A[0])
        end if
    else
        mid ← ⌊A.size/2⌋
        invers(A[0 : mid], res)                              ▷ divide A[] into two parts
        invers(A[mid + 1 : A.size − 1], res)
        i, j ← 0, mid + 1                                      ▷ start of MERGE process
        while either part doesn't finish scanning do
            append min(A[i], A[j]) to B[]       ▷ B[] is to store temporary sorted elements
            if A[i] > A[j] then                                   ▷ an inversion appears
                res ← res + (j + 1)           ▷ j + 1 is the number of numbers less than A[i]
                j ← j + 1
            else
                i ← i + 1
            end if
        end while
        append the rest part to B[]                              ▷ end of MERGE process
        A ← B
    end if
end function
r ← 0                                                          ▷ start of main function
return invers(a, r)
```

Space complexity of Quicksort

Define the depth of target leaf h, and the partition scale $\alpha = (size\ of\ first\ part)/(size\ of\ second\ part)$, so we have:

$$n \cdot \alpha^h = 1$$

which means the number of elements in the first part is reduced to 1 after h times' $\frac{\alpha}{1-\alpha}$ partitions.
$\therefore$ we have

$$h = \frac{\log_2(n)}{\log_2(1/\alpha)}$$

To let $h \leq \log_2(n)$, we must have $\alpha \leq 1/2$
$\therefore$ only use recursion for the smaller part of every partition, and use iteration instead for the rest

function QUICKSORT($A[], low, high$)
    while $low < high$ **do** ▷ for larger part, do iteration
        SPLIT(A, mid)
        if $(mid - 1) - low - 1 < high - (mid + 1) - 1$ **then** ▷ only do recursion for smaller part
            QUICKSORT($A, low, mid - 1$)
            $low \leftarrow mid + 1$
        else
            QUICKSORT($A, mid + 1, high$)
            $high \leftarrow mid - 1$
        end if
    end while
end function

Nuts and bolts

pick a *nut*, compare to all *bolts* and find its match, divide *bolts* in two
use the matched *bolt*, compare to all *nuts*, divide *nuts* in two
recursively do these steps for all subsets divided in the previous steps

Alike *Quicksort*, so it is $O(N \log N)$, but I do not know how do exactly $N \log N$ comparisons.

Oil pipeline

Require: an array $wells[]$ containing n wells' ys
Ensure: the median of the array
function MED($array[], num$)
    divide $array[]$ in $\lfloor num/5 \rfloor$ 5-element sets, a rest $num\%5$ set
    for each set **do**
        find its median using INSERTIONSORT, pick smaller if even
        append this median to $tempmid[]$
    end for
    if $tempmid.size = 1$ **then**
        return $tempmid[0]$
    else
        return MED($tempmid, tempmid.size$) ▷ bottom-up-recursively do the search
    end if
end function
return MED($wells, n$) ▷ main function, return the y of the main pipe