Dayou Du
N13825075
dd2645@nyu.edu

Homework 7 - SA
Fundamental Algorithms

2017-11-09

Ex 5.6

Algorithm 1 Ex 5.6

```
procedure $QuickSort(Arr[1..n])$
    Create a empty stack $S$
    $S.push((1, n))$
    repeat
        $(start, end) = S.pop()$
        if $start < end$ then
            $P \leftarrow Partition(start, end, Arr)$
            $S.push((start, P - 1))$
            $S.push((P + 1, end))$
        end if
    until $S$ is empty
end procedure
function $Partition(start, end;; Arr[1..n])$
    if $Arr[start] < Arr[end]$ then
        $Swap(Arr[start], Arr[end])$
    end if
    $Pivot \leftarrow Arr[end], L \leftarrow start, U \leftarrow end$
    repeat
        $Swap(Arr[L], Arr[U])$
        repeat
            $L \leftarrow L + 1$
        until $Arr[L] \geq Pivot$
        repeat
            $U \leftarrow U - 1$
        until $Arr[U] \leq Pivot$
    until $L \geq U$
    $Swap(Arr[start], Arr[L])$
    return $L$
end function
```

I got this question CORRECT.

Ex 5.22

standard Qsort

Use 88 as Pivot : 16 12 78 87 82 88 95 89
Use 16 as Pivot : 12 16 78 87 82 88 95 89
Use 78 as Pivot : 12 16 78 87 82 88 95 89
Use 87 and 95 as Pivot : 12 16 78 82 87 88 89 95
I got this question CORRECT.

merge sort

12 88 | 78 87 | 89 95 | 16 82
12 78 87 88 | 89 95 | 16 82
12 78 87 88 | 16 82 89 95
12 16 78 82 87 88 89 95
I got this question CORRECT.

insertion sort

12 88 | 78 87 89 95 16 82
12 78 88 | 87 89 95 16 82
12 78 87 88 | 89 95 16 82
12 78 87 88 89 | 95 16 82
I got this question CORRECT.

selection sort

12 | 88 78 87 89 95 16 82
12 16 | 78 87 89 95 88 82
12 16 78 | 87 89 95 88 82
12 16 78 82 | 89 95 88 87
I got this question CORRECT.
Note: Here I select the smallest number in each cycle. The solution select the largest one.

radix sort

0: ||1: ||2: 12 82 ||3: ||4: ||5: 95||6: 16 ||7: 87 ||8: 88 78 ||9: 89
12 82 95 16 87 88 78 89
0: ||1: 12 16 ||2: ||3: ||4: ||5: ||6: ||7: 78 ||8: 82 87 88 89 ||9: 95
12 16 78 82 87 88 89 95
I got this question CORRECT.

Ex 5.27

a)

No change. It's just reversed call tree, thus the depth of the tree is not changed.
I got this question CORRECT.

b)

Stack depth is smaller for processing small part firstly(before change) than processing large part at first(after change)
The maximum stack depth depends on the 'deepest call' for the sorting on one element.
If we process the small part as recursive and use iteration to replace the large part, the stack depth is no more than log_2n, since every recursive call will deal with no more than half of the input.
Otherwise, if we process the larger part as recursive ,the stack depth is no less than log_2n, since every recursive call will deal with no less than half of the input.
I got this question CORRECT.

c)

For the call stack, they are the same since there's no recursion calls.
For the stack that used to store informations, the depth for pushing larger part firstly(i.e. processing smaller part in the next round) is smaller.
The reason is the same as in part b), actually the stack we used in part c) is just the substitution for the call stack in part b).
I got this question CORRECT.
Note: Actually, whenever we push two parts into stack, we will process the later one in the next round. Thus, the part to be processed is equivalent with the recursive part in question (b). When we start to deal with the other part, we have already finished all the processing with the previous one. Thus processing this part is equivalent with what happened in the while-loop in question (b).

Ex 5.34

Algorithm 2 Ex 5.34

```
Input: The input array $A[1..n]$
Output: The index of item $x$
function $BinarySearch(start, end, x; ; A[1..n])$
    $startVal \leftarrow A[start]$
    $mid \leftarrow \lfloor (start + end)/2 \rfloor$
    $midVal \leftarrow A[mid]$
    if $midVal == x$ then
        return $mid$
    end if
    if $midVal < startVal$ then
        if $midVal < x < startVal$ then
            return $BinarySearch(mid + 1, end, x, A)$
        else
            return $BinarySearch(start, mid - 1, x, A)$
        end if
    else
        if $startVal \leq x < midVal$ then
            return $BinarySearch(start, mid - 1, x, A)$
        else
            return $BinarySearch(mid + 1, end, x, A)$
        end if
    end if
end function
```

Note:
1. At beginning we call $BinarySearch(1, n, x, A)$
2. The function can correctly deal with a fully sorted array, thus we do NOT need a standard binary search.
I got this question CORRECT.

5.U

Let set S contains all the permutations for the numbers in A. Thus S contains $Size_S = n!$ elements.

Assume each item S_i in S contains $Inver_i$ inversion pairs and $n(n-1)/2 - Inver_i$ non-inversion pairs. The whole set S contains $Inver_S = \sum_i Inver_i$ inversion pairs and $n!n(n-1)/2 - Inver_S$ non-inversion pairs.

Now we invert every S_i in S as S'_i and then form a new set S'. Then S' contains $Inver_{S'} = n!n(n-1)/2 - Inver_S$ inversion pairs. Note that S' is also all the permutations of A, thus actually S' is the same as S.

Thus we have $Inver_S = Inver_{S'}$, which gives $Inver_S = n!n(n-1)/2 - Inver_S$, $Inver_S = n!n(n-1)/4$

Thus, on average, each item have $Inver_S / Size_S = n(n-1)/4$ inversion pairs.
I got this question CORRECT.

5.V

Algorithm 3 Ex 5.V

```
Input: The original array $Data[1..n]$ and it's length $n$
Output: The number of inversion pairs in $Data$, and also the array is sorted
function $IMergeSortDriver(n;;Data[1..n])$
    return $IMergeSort(1, n, Data)$
end function
function $IMergeSort(start, end;;Data)$
    if $start \geq end$ then
        return $0$
    end if
    $mid \leftarrow \lfloor (start + end)/2 \rfloor$
    $count \leftarrow IMergeSort(start, mid) + IMergeSort(mid + 1, end)$
    Create empty array $temp[1..(end - start + 1)]$
    $i \leftarrow start, j \leftarrow mid + 1, k \leftarrow 1$
    while $i \leq mid$ or $j \leq end$ do
        if $i > mid$ then
            $temp[k] \leftarrow Data[j]$
            $k \leftarrow k + 1, j \leftarrow j + 1$
        else
            if $j > end$ then
                $temp[k] \leftarrow Data[i]$
                $k \leftarrow k + 1, i \leftarrow i + 1, count \leftarrow count + (j - mid - 1)$
            else
                if $Data[i] < Data[j]$ then
                    $temp[k] \leftarrow Data[i]$
                    $k \leftarrow k + 1, i \leftarrow i + 1, count \leftarrow count + (j - mid - 1)$
                else
                    $temp[k] \leftarrow Data[j]$
                    $k \leftarrow k + 1, j \leftarrow j + 1$
                end if
            end if
        end if
    end while
    Copy $temp[1..(end - start + 1)]$ to $Data[start, end]$
    return $count$
end function
```

I got this question CORRECT.
Note: we can put sentinel ∞ at the end of two parts of arrays to avoid messy bound checking

5.W

a)

Let x_i be the number of inversion pairs as (j,i) , $i \in [2,n], j \in [1,i-1]$. We have $\sum_i x_i = I$

Then when we dealing with number $A[i], i \in [2,n]$, the numbers in $A[1..i-1]$ is already in correct order, and the last x_i numbers is larger than $A[i]$.

Thus, as the procedure of Insertion Sort, $A[i]$ will compare with each of these x_i numbers, and with the first number that smaller than $A[i]$, which consumes $x_i + 1$ compare operations.

There's no operation with the first element, thus the total comparison number is $\sum_i x_i + n - 1 = I + n - 1$
I got this question CORRECT.

b)

As we calculated in part a, there's about $I + n - 1$ compare operations.

Whenever we dealing with $A[i]$ and we find a number larger than it, we will swap them, which expends x_i swap operations. Thus there's totally $\sum_i x_i = I$ swap operations.

The other work(loop control, boundary check, etc.) expends no more than a constant factor of $n + I$
Thus, in total, the work is $\Theta(I + n)$ for Insertion Sort.
I got this question CORRECT.

c)

Number of compare operations:

For $i \in [2,n]$, we will compare each pair of adjacent numbers. i.e $(j, j+1), j \in [1..i-1]$.

Thus there's totally $n(n-1)/2$ compare operations.

Number of swap operations:

Considering a typical swap operation as follows:

$A[1], A[2], ...A[i], A[i+1], ...A[n] \rightarrow A[1], A[2], ...A[i+1], A[i], ...A[n], A[i+1] < A[i]$.

The inversion count for the left is exactly one more than the inversion count on the right, because the inversions related with $A[j], j \neq i, j \neq i+1$ is unchanged, and we eliminated inversion pair$(A[i], A[i+1])$

Thus in total Bubble Sort expends I swap operations.

Total Work:

The other work(loop control, boundary check, etc.) expends no more than a constant factor of n^2. Note that $I = O(n^2)$.
Thus, in total, the work is $\Theta(n^2)$ for Bubble Sort.
I got this question CORRECT.

5.Radix

a)

$n^{21} = (n^3)^7$
$radix_a = n^3$
I got this question CORRECT.

b)

$\frac{1}{4} < 1$
$radix_b = \sqrt{n}+1$
The range of the numbers is $[0, \sqrt{n}]$, which means there are totally $\sqrt{n} + 1$ different numbers in the range, not $\sqrt{n}$ different numbers. Thus we should use $\sqrt{n} + 1$ buckets. The fault I made here is that I didn't look carefully about the range, and normally there was a "-1" for the upper bound.
Lesson to learn: Don't be mislead by the Inertial Thinking.

c1)

$(10/1)(2n + n) = 30n$
I got this question CORRECT.

c2)

$(10/0.5)(2n + \sqrt{n}) = 20(2n + \sqrt{n})$
I got this question CORRECT.

c3)

$(10/0.1)(2n + n^{\frac{1}{10}}) = 100(2n + n^{\frac{1}{10}})$
I got this question CORRECT.

c4)

$(10/2)(2n + n^2) = 5(2n + n^2)$
I got this question CORRECT.

c5)

$(10/1)(2n + n/2) = 25n$
I got this question CORRECT.

c6)

$(10/1)(2n + 2n) = 40n$
I got this question CORRECT.

c7)

$(150/15)(2 \times 2^{25} + 2^{15}) = 10(2^{26} + 2^{15})$
I got this question CORRECT.

c8)

$(150/25)(2 \times 2^{25} + 2^{25}) = 9 \times 2^{26}$
I got this question CORRECT.

c9)

$(150/35)(2 \times 2^{25} + 2^{35}) = \frac{30}{7}(2^{26} + 2^{35})$
Although they are very close.The round number here have to be a integer, which means it should be $\lceil \frac{30}{7} \rceil$ instead of $\frac{30}{7}$. Thus the answer should be $5(2^{26} + 2^{35})$.
The fault I made here is that I concentrated on the "math equations" and forgot about the physical modeling. Lesson to learn: mathematic abstraction is good for a quick-computation, and physical modeling is useful for checking the results (recalling a common sense).

c10)

Let $f(n)$ be the number of compare operations (between Array Elements) for n elements using Merge Sort(in the worst case).
$f(n) = n - 1 + 2f(\frac{n}{2}), n > 1$
$f(1) = 0$
Thus $f(n) = nlog_2 n - n + 1$
When $n = 2^{25}$, $f(2^{25}) \approx 2^{25} \times log_2 2^{25} - 2^{25} = 24(2^{25})$
Or, if we use sentinels to avoid messy boundary checkings , the comparison for the worst case will be $25(2^{25})$ but there's less other boundary checking operations.
I got this question CORRECT.