Solution for Problem Set 4

201300035 方盛俊

Problem 1

(a)

1. $\langle 5, 13, 2, 25, 7, 17, 20, 15, 4\rangle$
2. $\langle 5, 13, 2, 25, 7, 17, 20, 15, 4\rangle$
3. $\langle 5, 13, 2, 25, 7, 17, 20, 15, 4\rangle$
4. $\langle 5, 25, 2, 13, 7, 17, 20, 15, 4\rangle$
5. $\langle 5, 25, 2, 13, 7, 17, 20, 15, 4\rangle$
6. $\langle 5, 25, 2, 15, 7, 17, 20, 13, 4\rangle$
7. $\langle 5, 25, 2, 15, 7, 17, 20, 13, 4\rangle$
8. $\langle 5, 25, 20, 15, 7, 17, 2, 13, 4\rangle$
9. $\langle 5, 25, 20, 15, 7, 17, 2, 13, 4\rangle$
10. $\langle 25, 5, 20, 15, 7, 17, 2, 13, 4\rangle$
11. $\langle 25, 5, 20, 15, 7, 17, 2, 13, 4\rangle$
12. $\langle 25, 15, 20, 5, 7, 17, 2, 13, 4\rangle$
13. $\langle 25, 15, 20, 5, 7, 17, 2, 13, 4\rangle$
14. $\langle 25, 15, 20, 5, 7, 17, 2, 13, 4\rangle$
15. $\langle 25, 15, 20, 5, 7, 17, 2, 13, 4\rangle$
16. $\langle 25, 15, 20, 13, 7, 17, 2, 5, 4\rangle$

(b)

1. $\langle 25, 15, 20, 13, 7, 17, 2, 5, 4\rangle$
2. $\langle 4, 15, 20, 13, 7, 17, 2, 5, 25\rangle$
3. $\langle 4, 15, 20, 13, 7, 17, 2, 5, 25\rangle$
4. $\langle 20, 15, 4, 13, 7, 17, 2, 5, 25\rangle$
5. $\langle 20, 15, 4, 13, 7, 17, 2, 5, 25\rangle$
6. $\langle 20, 15, 17, 13, 7, 4, 2, 5, 25\rangle$
7. $\langle 20, 15, 17, 13, 7, 4, 2, 5, 25\rangle$
8. $\langle 5, 15, 17, 13, 7, 4, 2, 20, 25\rangle$
9. $\langle 5, 15, 17, 13, 7, 4, 2, 20, 25\rangle$
10. $\langle 17, 15, 5, 13, 7, 4, 2, 20, 25\rangle$

11. $\langle 17, 15, 5, 13, 7, 4, 2, 20, 25 \rangle$
12. $\langle 17, 15, 5, 13, 7, 4, 2, 20, 25 \rangle$
13. $\langle 17, 15, 5, 13, 7, 4, 2, 20, 25 \rangle$
14. $\langle 2, 15, 5, 13, 7, 4, 17, 20, 25 \rangle$
15. $\langle 2, 15, 5, 13, 7, 4, 17, 20, 25 \rangle$
16. $\langle 15, 2, 5, 13, 7, 4, 17, 20, 25 \rangle$
17. $\langle 15, 2, 5, 13, 7, 4, 17, 20, 25 \rangle$
18. $\langle 15, 13, 5, 2, 7, 4, 17, 20, 25 \rangle$
19. $\langle 15, 13, 5, 2, 7, 4, 17, 20, 25 \rangle$
20. $\langle 4, 13, 5, 2, 7, 15, 17, 20, 25 \rangle$
21. $\langle 4, 13, 5, 2, 7, 15, 17, 20, 25 \rangle$
22. $\langle 13, 4, 5, 2, 7, 15, 17, 20, 25 \rangle$
23. $\langle 13, 4, 5, 2, 7, 15, 17, 20, 25 \rangle$
24. $\langle 13, 7, 5, 2, 4, 15, 17, 20, 25 \rangle$
25. $\langle 13, 7, 5, 2, 4, 15, 17, 20, 25 \rangle$
26. $\langle 4, 7, 5, 2, 13, 15, 17, 20, 25 \rangle$
27. $\langle 4, 7, 5, 2, 13, 15, 17, 20, 25 \rangle$
28. $\langle 7, 4, 5, 2, 13, 15, 17, 20, 25 \rangle$
29. $\langle 7, 4, 5, 2, 13, 15, 17, 20, 25 \rangle$
30. $\langle 7, 4, 5, 2, 13, 15, 17, 20, 25 \rangle$
31. $\langle 7, 4, 5, 2, 13, 15, 17, 20, 25 \rangle$
32. $\langle 2, 4, 5, 7, 13, 15, 17, 20, 25 \rangle$
33. $\langle 2, 4, 5, 7, 13, 15, 17, 20, 25 \rangle$
34. $\langle 5, 4, 2, 7, 13, 15, 17, 20, 25 \rangle$
35. $\langle 5, 4, 2, 7, 13, 15, 17, 20, 25 \rangle$
36. $\langle 2, 4, 5, 7, 13, 15, 17, 20, 25 \rangle$
37. $\langle 2, 4, 5, 7, 13, 15, 17, 20, 25 \rangle$
38. $\langle 4, 2, 5, 7, 13, 15, 17, 20, 25 \rangle$
39. $\langle 4, 2, 5, 7, 13, 15, 17, 20, 25 \rangle$
40. $\langle 2, 4, 5, 7, 13, 15, 17, 20, 25 \rangle$

Problem 2

Overview:

Create a list called L and build a maximum heap H with k largest elements from k sorted lists. In n-loop, extract maximum from the heap and save it to the array L, then add new element from the list where extracted maximum existed to the heap. Finally we get a sorted list L.

Algorithm:

Let k sorted lists called S. We assert that the lists are arranged from small to large.

Algorithm 1 Sort

```
function SORT(S)
    Let L be a new empty list, H be a new empty heap
    for i = 1 to k do
        H.insert(S[i].removeLast())
    end for
    for i = 1 to n do
        x = H.extractMax()
        L.addToHead(x)
        Let T be the original list of x
        if T is not empty then
            H.insert(T.removeLast())
        end if
    end for
    return L
end function
```

Time Complexity:

The first loop be executed k times and each times be executed in time $O(\log k)$ of H .insert(). The total time of the first loop is $O(k \log k)$

The first loop be executed n times and each times be executed in time $O(\log k)$ of H .extractMax() and H.insert(). The total time of the first loop is $O(n \log k)$

$$T(n) = c_0 + O(k \log k) + O(n \log k) = O(n \log k)$$

Problem 3

(a)

We assume for this problem that the input size n is always a power of 2.

We prove that after the function `Unusual()` the array will be sorted. And the two subarrays $A[1...n/2]$ and $A[n/2+1...n]$ of input array $A[1...n]$ of `Unusual()` are both sorted.

Basis: When $n = 2$, the subarrays $A[1]$ and $A[2]$ are both sorted normally, and after swap operation, the new array $A'[1, 2]$ are sorted.

I.H: The array $A[1...n/2]$ and $A[n/2+1...n]$ are both sorted.

I.S:

Before the for-loop, we can know that the array $A[1...n/2]$ and $A[n/2+1...n]$ are both sorted.

So we know that $A[1...n/4] \prec A[n/4+1...n/2]$ and $A[n/2+1...3n/4] \prec A[3n/4+1...n]$ (The sign $A \prec B$ mean that for any element a in A, we have that a is smaller than any element b in B).

After the for-loop, which swap 2nd and 3rd quarters, we can know that the new subarray $A[1...n/4] \prec A[n/2+1...3n/4]$ and $A[n/4+1...n/2] \prec A[3n/4+1...n]$.

We rename the four parts as A, B, C, D, so that $A \prec C$ and $B \prec D$.

After `Unusual(A[1...n/2])` and `Unusual(A[n/2+1...n])`, $A[1...n/2]$ and $A[n/2+1...n]$ are sorted.

Now we prove that the current $A[1...n/4]$ are the smallest part of the four parts.

For any element $e \in A[1...n/4]$, for example, $e \in B$, so we can know that $e \prec A[n/4+1...n/2]$ and $e \prec D$.

Because $e \in A[1...n/4]$ and $e \in B$, so there at least an element $a \in A[n/4+1...n/2]$ and $a \in A$. We know $A \prec C$, so $e \prec A[n/4+1...n/2] \Rightarrow e \leqslant a \Rightarrow e \prec C$.

Finally, we can know that for any element e in $A[1...n/4]$, we have that $e \prec A[n/4+1...n/2]$, $e \prec C$, $e \prec D$. We prove that the current $A[1...n/4]$ are the smallest part of the four parts.

Similarly, we can prove that current $A[3n/4+1...n]$ are the largest part of the four parts.

For `Unusual(A[n/4+1...3n/4])`, because $A[1...n/2]$ and $A[n/2+1...n]$ are both sorted, the $A[n/4+1...n/2]$ and $A[n/2+1...3n/4]$ are both sorted. After the statement, we get the new sorted middle half array $A[n/4+1...3n/4]$.

So finally we can get the sorted array $A[1...n]$.

(b)

Let the input array be $A = \langle 3, 4, 1, 2 \rangle$.

After `Cruel(A[1...n/2])` and `Cruel(A[n/2+1...n])`, the array was still $\langle 3, 4, 1, 2 \rangle$.

In the `Unusual(A[1...n])` call, before recursive `Unusual()`, without for-loop, the array was still $\langle 3, 4, 1, 2 \rangle$.

After `Unusual(A[1...n/2])` and `Unusual(A[n/2+1...n])`, the array was still $\langle 3, 4, 1, 2 \rangle$.

After `Unusual(A[n/4+1...3n/4])`, the array became $\langle 3, 1, 4, 2 \rangle$, which was not sorted totally.

So the modified algorithm is not correct.

(c)

Let the input array be $A = \langle 3, 4, 1, 2 \rangle$.

After `Cruel(A[1...n/2])` and `Cruel(A[n/2+1...n])`, the array was still $\langle 3, 4, 1, 2 \rangle$.

In the `Unusual(A[1...n])` call, after for-loop, the array became $\langle 3, 1, 4, 2 \rangle$.

After `Unusual(A[1...n/2])`, the array was still $\langle 1, 3, 4, 2 \rangle$.

After `Unusual(A[n/4+1...3n/4])`, the array was still $\langle 1, 3, 4, 2 \rangle$.

After `Unusual(A[n/2+1...n])`, the array became $\langle 1, 3, 2, 4 \rangle$, which was not sorted totally.

So the modified algorithm is not correct.

(d)

Unusual: $T_1(n) = 3T_1(\frac{n}{2}) + \frac{n}{4} = O(n^{\log 3})$

Cruel: $T_2(n) = 2T_2(\frac{n}{2}) + T_1(n)$

Because $n^{\log 3} > 2 \cdot (\frac{n}{2})^{\log 3}$, the sums of each layer is degressive.

Using master theorem we know:

So $T_2(n) = O(n^{\log 3})$

Problem 4

(a)

Using induction.

Basis:

There are three basis:

When $p = 1$ and $r = n$, the first times of loop and the shallow stack, after $q \leftarrow$ Partition(A, p, r), we make sure that $A[1...q] \prec A[q+1...n]$.

When $p = r = 1$, at the end of first times of loop and the deepest stack, $A[1]$ is sorted (When $p = r = 2$, the case is similar).

When $p = r$, the only statement which was execute is $p \leftarrow$ Partition(A, p, r), It change nothing, and $A[p]$ or $A[r]$ is sorted.

I.H:

There are two hypotheses:

At the beginning of each times of loop, the $A[1...p-1]$ is sorted and $A[1...p-1] \prec A[p...n]$.

TRQuickSort($A, p, q-1$) will make $A[p...q-1]$ be sorted.

I.S:

Firstly, we prove that at the end of each times of loop, $A[1...q]$ is sorted and $A[1...q] \prec A[q+1...n]$.

For each times of loop, using I.H's first hypothesis, we know the $A[1...p-1]$ is sorted and $A[1...p-1] \prec A[p...n]$.

After $q \leftarrow$ Partition(A, p, r), we know that $A[p...q-1] \prec A[q] \prec A[q+1...r]$.

Using I.H's second hypothesis, it is that TRQuickSort($A, p, q-1$) will make $A[p...q-1]$ be sorted.

So $A[p...q]$ was sorted after TRQuickSort($A, p, q-1$), and $A[p...q] \prec A[q+1...r]$.

So at the end of each times of loop, $A[1...q]$ is sorted and $A[1...q] \prec A[q+1...n]$.

Secondly, we prove that after all loop, $A[p...r]$ will be sorted.

Because for each times of loop, the $A[1...q]$ is sorted, the final q is r or $r-1$ and $A[1...q] \prec A[q+1...n]$ for each q, the $A[p...r]$ will be sorted.

(b)

Create an input array $A = \langle 1, 2, 3, ..., n \rangle$ for TRQuickSort.

As we can see, the $q \leftarrow$ Partition(A, p, r) didn't change the input array. At the first time, $q = r$.

Then, call `TRQuickSort(A,p,q-1)`, which is same with `TRQuickSort(A,p,r-1)`.

Similarly, the next `TRQuickSort(A,p,r-1)` will call `TRQuickSort(A,p,r-2)`, which divide problem from n into 1 and $n-1$.

So we can prove that it will call n times `TRQuickSort()`. The stack depth is $\Theta(n)$.

(c)

Algorithm 2 ModifiedTRQuickSort

function $\textsc{ModifiedTRQuickSort}()$
    while $p < r$ **do**
        $q \leftarrow \text{Partition}(A, p, k)$
        if $q < (p+r)/2$ **then**
            $\text{TRQuickSort}(A, p, q-1)$
            $p \leftarrow q + 1$
        else
            $\text{TRQuickSort}(A, q+1, r)$
            $r \leftarrow q - 1$
        end if
    end while
end function

Why the modification can guarantee $\Theta(\lg n)$ worst-case stack depth?

The TRQuickSort is the modified version QuickSort algorithm, which replaces one recursive call by using an iterative control structure. We can replace the second recursive call, it is obvious that we can replace the first recursive call.

We use a if statement so that we can let the input subarray of TRQuickSort is always the small part of the two parts. In the worst case, we divide the array into two subarrays with same length. We can make sure that it is $\Theta(\lg n)$ worst-case stack depth.

Problem 5

(a)

Algorithm:

Algorithm 3 Sort

function $\textsc{Sort}(A[1...n])$
    for $\text{i} = n - \sqrt{n}$ **to** 0 step $\sqrt{n}/2$ **do**
        for $\text{j} = 0$ **to** i step $\sqrt{n}/2$ **do**

SqrtSort(j)
end for
end for
end function

Correctness:

It is like a bubble sort. Bubble sort compare and swap two elements in an array. The `Sort` function view $\sqrt{n}/2$ elements as an element or block.

- Loop invariant: After the i-th loop, $A[i+\sqrt{n}/2...i+\sqrt{n}]$ is the largest block of $A[1...i+\sqrt{n}]$ and $A[i+\sqrt{n}/2...i+\sqrt{n}]$ is sorted.
- Proof:
    - Initialization: After the first loop, $A[n-\sqrt{n}/2...n]$ is the largest block and it is sorted.
    - Maintain: After the inner for-loop, the largest block was send to the tail of $A[1...i+\sqrt{n}]$, so $A[i+\sqrt{n}/2...i+\sqrt{n}]$ is the largest block of $A[1...i+\sqrt{n}]$ and $A[i+\sqrt{n}/2...i+\sqrt{n}]$ is sorted.
    - Termination: $A[1...n]$ is sorted.

Time Complexity:

We calculate the times of we call `SqrtSort`.

$$\because \frac{n-\sqrt{n}}{\sqrt{n}/2} = 2\sqrt{n}-2$$

$$\therefore T(n) = \frac{(1+2\sqrt{n}-2)(2\sqrt{n}-2)}{2} = O(n)$$

(b)

Algorithm 4 SqrtSort

---

function SQRTSORT(k)
if $\sqrt{n} == 1$ **then**
return $A[k+1]$
else if $\sqrt{n} == 2$ **then**
if $A[k+1] > A[k+2]$ **then**
Swap($A[k+1]$, $A[k+2]$)
end if
else
Sort($A[k+1...k+\sqrt{n}]$)
end if
end function

Time Complexity:

$T(2) = \Theta(1)$

$T(n) = n \cdot T(\sqrt{n})$

令 $m = \lg n$

$\therefore T(2^m) = 2^m \cdot T(2^{m/2})$

$\therefore S(m) = m \cdot T(\frac{m}{2}) = m \cdot \frac{m}{2} \cdot \frac{m}{2^2} \cdot T(\frac{m}{2^3}) = \frac{m^{\lg m}}{2^0 \cdot 2^1 \cdots 2^{\lg m}} = O(\frac{m^{\lg m}}{2^{(\lg m)^2/2}})$

$\therefore T(n) = O(\frac{(\lg n)^{\lg \lg n}}{2^{(\lg \lg n)^2/2}})$

Using the other way, because $T(n) = n \cdot T(\sqrt{n})$, we guess $T(n) = n^2$. We can prove it by substitute it so that $T(n) = n^2 = n \cdot T(\sqrt{n}) = n \cdot (\sqrt{n})^2 = n^2$.

$\therefore T(n) = n^2$.

Problem 6

(a)

Let the expected value of `OneInThree()` be E_a, the probability of `OneInThree()` returning 1 be p_a.

$\therefore E_a = \frac{1}{2} \times 0 + \frac{1}{2} \times (1 - E_a)$

$\therefore E_a = \frac{1}{3}$

$\because E_a = 1 \cdot p_a + 0 \cdot (1 - p_a) = p_a$

$\therefore p_a = \frac{1}{3}$

(b)

Let the exact expected number of `OneInThree()` be E_b.

$\because E_b = \frac{1}{2} \times 1 + (\frac{1}{2})^2 \times 2 + (\frac{1}{2})^3 \times 3 + \cdots = \sum_{k=1}^{\infty} (\frac{1}{2})^k \cdot k$

$\therefore E_b = \sum_{k=1}^{\infty} (\frac{1}{2})^k \cdot k = \frac{1}{2} + \sum_{k=2}^{\infty} (\frac{1}{2})^k \cdot k = \frac{1}{2} + \sum_{k=1}^{\infty} (\frac{1}{2})^{k+1} \cdot (k+1)$

$$\because \frac{1}{2}E_b = \sum_{k=1}^{\infty}(\frac{1}{2})^{k+1} \cdot k$$

$$\therefore E_b - \frac{1}{2}E_b = \frac{1}{2}E_b = \frac{1}{2} + \sum_{k=1}^{\infty}(\frac{1}{2})^{k+1} = \sum_{k=1}^{\infty}(\frac{1}{2})^{k} = 1$$

$$\therefore E_b = 2$$

(c)

Let the expected value of `OneInTwo()` be E_c, the probability of `BiasedCoin()` returning 1 be p.

Algorithm 5 OneInTwo

```
function ONEINTWO()
    a = BiasedCoin()
    b = BiasedCoin()
    if (a == 1 and b == 1) or (a == 0 and b == 0) then
        return OneInTwo()
    else if a == 1 and b == 0 then
        return 1
    else if a == 0 and b == 1 then
        return 0
    end if
end function
```

$$\because E_c = [p^2 + (1-p)^2]E_c + p(1-p) \cdot 1 + p(1-p) \cdot 0$$

$$\therefore E_c = \frac{1}{2}$$

(d)

Let the exact expected number of `BiasedCoin()` be E_d, $p_d = p^2 + (1-p)^2$.

$$\because E_d = (1-p_d) \cdot 0 + p_d(1-p_d) \cdot 2 + p_d^2(1-p_d) \cdot 4 + \cdots = 2(1-p_d)\sum_{k=1}^{\infty} p_d^k \cdot k$$

$$\therefore E_d = 2(1-p_d)\sum_{k=1}^{\infty} p_d^k \cdot k = 2(1-p_d)[p_d + \sum_{k=2}^{\infty} p_d^k \cdot k] = 2(1-p_d)[p_d + \sum_{k=1}^{\infty} p_d^{k+1} \cdot (k+1)]$$

$$\because p_d E_d = 2(1-p_d)\sum_{k=1}^{\infty} p_d^{k+1} \cdot k$$

$$\therefore E_d - p_d E_d = (1-p_d)E_d = 2(1-p_d)[p_d + \sum_{k=1}^{\infty} p_d^{k+1}] = 2(1-p_d)\sum_{k=1}^{\infty} p_d^{k}$$

$$\therefore E_d = 2\sum_{k=1}^{\infty} p_d^{k} = 2 \cdot \lim_{n\to\infty} \frac{p_d(1-p_d^n)}{1-p_d} = \frac{2p_d}{1-p_d} = \frac{-2p^2+2p-1}{p\,(p-1)}$$

Problem 7

Because $2^{19} = 524288 < 1000000 < 2^{20} = 1048576$, so the number contains 20 bits. So we need asks 20 times for all bits of the number. We assure that the first bit is the lowest bit.

Algorithm 6 Query

```
function QUERY()
    sum = 0
    for i = 1 to 20 do
        if query the i-th bit == 1 then
            sum = sum + 2^(i-1)
        end if
    end for
    return sum
end function
```

So the $T(n) = c_0 + c_1 \log n + c_2 = \Theta(\log n) = 20$ times

The the upper bound and lower bound are both $O(\log n)$ or 20 times.

It is obvious that the upper bound is correct.

Prove that the lower bound is correct, too:

Using the adversary argument.

Assure that we only ask $n-1=19$ times, like $?0000000000000000000$. Can we determine the number is zero or 10000000000000000000 (binary number)?

If we guess the number is zero, the adversary Eve can say that the number is 2^{19}.

If we guess the number is 2^{19}, the adversary Eve can say that the number is zero.

And if we use other algorithm like dichotomy, it is also impossible to know the number less than 20 times question.

So the lower bound is $O(\log n)$ or 20 times.