Written Assignment 1

薛劭杰 1930026143

Question1.

There are two cases of recursion. One is A[mid] > x and another is A[mid]<x, the basis case is left ≤ right or A[mid]=middle. Both the recursions of the two cases are the T(n/2).

```
$binarySearch\ (A, left, right, x)\{$
    $if\ (left <= right)\{$                                  1
        $mid = left + (right - left)\ /\ 2$                   4
        $if\ (A[mid] == x)\{$                                 1
            $return\ mid;$                                1
        $\}$
        $if\ (A[mid] > x)\{$                                  1
            $return\ binarySearch\ (A, left, mid - 1, x)$         2+T(n/2)
        $\}$
        $return\ binarySearch\ (A, mid + 1, r, x)$                2+T(n/2)
        $\}$
    $return\ -1;$                                            1
$\}$
```

$$\begin{cases} T(n) = T\left(\frac{n}{2}\right) + C, n > 1 \\ T(1) = O(1) \end{cases}$$

$T(n) = T\left(\frac{n}{2}\right) + C = T\left(\frac{n}{4}\right) + C + C = \ldots = T\left(\frac{n}{2^i}\right) + C * i$. In the i^{th} times recursion, let $i = log_2 n$, $T(n) = T(1) + C * log_2 n, T(1)\ also\ is\ a\ constant$. So, the computational cost of this algorithm is $\theta(logn)$.

Question2.

(a) $T(n) = 2T\left(\frac{n}{2}\right) + O(n), T(1) = 1.$

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n) = 2\left(2T\left(\frac{n}{4}\right) + O\left(\frac{n}{2}\right)\right) + O(n)$$

$$= \ldots = 2^i * T\left(\frac{n}{2^i}\right) + 2^{i-1} * O\left(\frac{n}{2^{i-1}}\right) + 2^{i-2} * O\left(\frac{n}{2^{i-2}}\right) + \ldots + 2 * O\left(\frac{n}{2}\right) + O(n)$$

$= 2^i * T\left(\frac{n}{2^i}\right) + i * O(n)$, in the i^{th} times recursion.

Let the $2^i = n \rightarrow i = \log_2 n$, $T(n) = n * T(1) + \log_2 n * O(n)$. So, the upper bound of the algorithm is $O(nlogn)$

(b) $T(n) = 2T\left(\frac{n}{4}\right) + O(n), T(1) = 1.$

$T(n) = 2T\left(\frac{n}{4}\right) + O(n) = 2\left(2T\left(\frac{n}{16}\right) + O\left(\frac{n}{4}\right)\right) + O(n)$

$= \dots = 2^i * T\left(\frac{n}{4^i}\right) + 2^{i-1} * O\left(\frac{n}{4^{i-1}}\right) + 2^{i-2} * O\left(\frac{n}{4^{i-2}}\right) + \dots + 2 * O\left(\frac{n}{4}\right) + O(n)$

$= 2^i * T\left(\frac{n}{4^i}\right) + \frac{2^{i-1}}{4^{i-1}} O(n) + \dots + \frac{2}{4} * O(n) + O(n)$

$= 2^i * T\left(\frac{n}{4^i}\right) + \frac{1-\frac{1}{2}^i}{1-\frac{1}{2}} * O(n)$, in the i^{th} times recursion.

Let the $4^i = n \to i = \log_4 n,\ T(n) = n^{\frac{1}{2}} * T(1) + (2 - n^{-\frac{1}{2}}) * O(n)$

The upper bound is $O(n)$.

(c) $T(n) = 4T\left(\frac{n}{2}\right) + 1, T(1) = O(1)$.

$T(n) = 4T\left(\frac{n}{2}\right) + 1 = 4\left(4T\left(\frac{n}{4}\right) + 1\right) + 1$

$= \dots = 4^i * T\left(\frac{n}{2^i}\right) + 4^{i-1} * 1 + 4^{i-2} * 1 + \dots + 4 * 1$

$= 4^i * T\left(\frac{n}{2^i}\right) + 4^{i-1} + 4^{i-2} + \dots + 4^0 = 4^i * T\left(\frac{n}{2^i}\right) + \frac{4^i - 1}{3}$, in the i^{th} times recursion.

Let the $2^i = n \to i = \log_2 n, T(n) = n^2 T(1) + \frac{n^2-1}{3} = O(n^2)$

The upper bound is $O(n^2)$.

(d) $T(n) = T\left(n^{\frac{1}{2}}\right) + 1,\ T(1) = 1, T(2) = 1$

$T(n) = T\left(n^{\frac{1}{2}}\right) + 1 = T\left(n^{\frac{1}{4}}\right) + 1 + 1$

$= \dots = T\left(n^{2^{-i}}\right) + i * 1$, in the i^{th} times recursion.

Let the $n^{2^{-i}} = 2, i = \log_2(\log_2 n), T(n) = 1 + \log_2(\log_2 n)$
The upper bound is $O(\log(\log n))$.

Question3.
(1). In the case 1: $\log_2 n$ *is not an integer*
The total time cost: $1 + 1 + (\log_2 n + 1) * (2 + 2) + (\log_2 n + 2) + 1 = 5\log_2 n + 9$
In the case 2: $1 + 1 + (\log_2 n * (2 + 2)) + (\log_2 n + 1) + 1 = 5 * \log_2 n + 4$
Both the two cases are $\theta(logn)$

(2) There are three layers of nested loops,

The first loops out: $1 + 1 + n + 1 + n$ (Times of getting into loop is n)

Times of getting into second loop: $\sum_{j=n-1}^{1} \log\left(\frac{n}{j}\right)$ because $j * 2^x = n \to x = \log_2 \frac{n}{x}$

$\log_2 \frac{n}{n-1} + \log_2 \frac{n}{n-2} + \dots + \log_2 \frac{n}{1}$

$= (\log_2 n - \log_2 (n-1)) + (\log_2 n - \log_2 (n-2)) + \dots + (\log_2 n - \log_2 1)$

$= (n-1) * \log_2 n - (\log_2 (n-1) + \log_2 (n-2) + \dots + \log_2 1)$

$= (n-1) * \log_2 n - (\log_2 [(n-1) * (n-2) * \dots * 1])$

$= (n-1) * \log_2 n - (\log_2 (n-1)!)$

The second loops out: $1 + 1 + n + 1 + (n-1) * \log_2 n - (\log_2 (n-1)!) + (n-1) * \log_2 n - (\log_2 (n-1)!) + n$

Times of getting into second loop: $\sum_{j=n-1}^{1} j * \frac{1-2^{\log_2\left(\frac{n}{j}+1\right)}}{1-2}$(Sum of geometric sequence)

$$(n-1) * \frac{1-2^{\log_2\left(\frac{n}{n-1}+1\right)}}{1-2} + (n-2) * \frac{1-2^{\log_2\left(\frac{n}{n-2}+1\right)}}{1-2} + \dots + 1 * \frac{1-2^{\log_2\left(\frac{n}{1}+1\right)}}{1-2}$$

$= (n-1) * n + (n-2) * n + \dots + 1 * n = ((n-1)!) * n$

The second loops out: $1 + 1 + n + 1 + (n-1) * \log_2 n - \left(\log_2 \frac{n^2}{2}\right) + (n-1) * \log_2 n - \left(\log_2 \frac{n^2}{2}\right) + n - 1 + \left((n-1)!\right) * n * 3 + \left((n-1)!\right) * n * 3 + n - 1 + 1$

To conclude, the total time cost is: $2 + 3n + 2 * ((n-1) * \log_2 n - (\log_2 (n-1)!)) + 6n(n-1))$.

So $O(n^2)$.

(3)

$$\begin{cases} T(1) = O(1) \\ T(n) = c + (n-1) \quad n \neq 1 \end{cases}$$

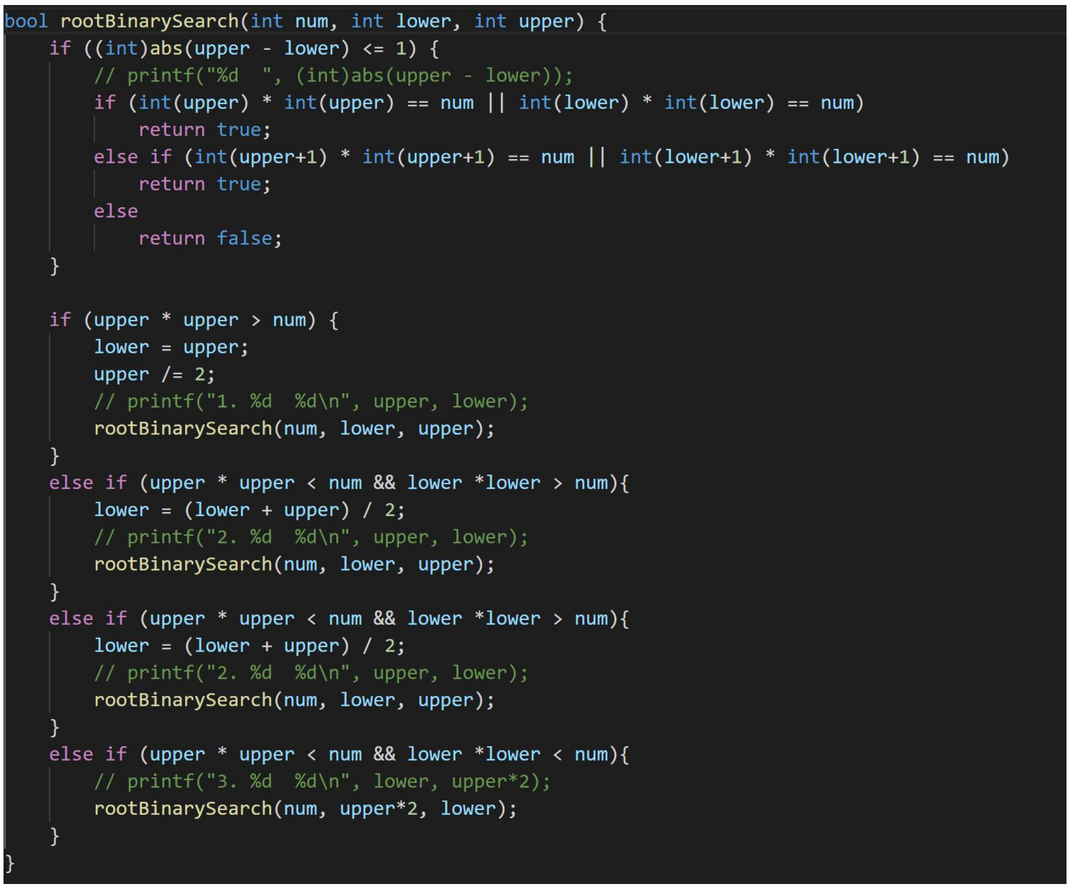
$T(n) = c + T(n-1) = c + c + T(n-2)$

$= \mathrm{i} * \mathrm{c} + \mathrm{T}(\mathrm{n} - \mathrm{i})$ in the i^{th} times iteration.

When $i = n-1$, $T(n) = (n-1) * c * T(1) = cn - c$

So, time cost is: $\theta(n)$.

Question4.



```
bool rootBinarySearch(int num, int lower, int upper) {
    if ((int)abs(upper - lower) <= 1) {
        // printf("%d  ", (int)abs(upper - lower));
        if (int(upper) * int(upper) == num || int(lower) * int(lower) == num)
            return true;
        else if (int(upper+1) * int(upper+1) == num || int(lower+1) * int(lower+1) == num)
            return true;
        else
            return false;
    }

    if (upper * upper > num) {
        lower = upper;
        upper /= 2;
        // printf("1. %d  %d\n", upper, lower);
        rootBinarySearch(num, lower, upper);
    }
    else if (upper * upper < num && lower *lower > num){
        lower = (lower + upper) / 2;
        // printf("2. %d  %d\n", upper, lower);
        rootBinarySearch(num, lower, upper);
    }
    else if (upper * upper < num && lower *lower > num){
        lower = (lower + upper) / 2;
        // printf("2. %d  %d\n", upper, lower);
        rootBinarySearch(num, lower, upper);
    }
    else if (upper * upper < num && lower *lower < num){
        // printf("3. %d  %d\n", lower, upper*2);
        rootBinarySearch(num, upper*2, lower);
    }
}
```

```
$PERFECTSQUARE(n, prev, pnext)$
    $IF\ (The\ difference\ of\ prev\ and\ pnext\ <\ 1)$
        $IF\ (int(pnext)\ or\ int(prev)\ square\ =\ n)$
            $RETURN\ True$
        $ELS\ IF\ (int(pnext+1)\ 's\ square\ or\ int(prev+1)'s\ square\ =\ n)$
            $RETURN\ True$
        $ELSE$
            $RETURN\ False$
    $IF\ (The\ square\ of\ pnext\ >\ n)$
        $PERFECTSQUARE\ (n, pnext, pnext/2)$
    ELSE IF $(The\ square\ of\ pnext\ <\ n\ \&\&\ The\ square\ of\ prev\ >\ n)$
        $PERFECTSQUARE\ (n, (prev+pnext)/2, pnext)$
    $ElSE\ IF\ ((The\ square\ of\ pnext\ <\ n\ \&\&\ The\ square\ of\ prev\ <\ n))$
        $PERFECTSQUARE\ (n, pnext*2, prev)$
```

The first case of iteration: $T(n) = T\left(\frac{n}{2}\right) + c$

The second case of iteration: $T(n) = T\left(\frac{1}{2}n\right) + c$

$$T(n) = T\left(\frac{1}{2}n\right) + c = T\left(\frac{1}{4}n\right) + c + c = \dots = T(\left(\frac{1}{2}\right)^i \text{n}) + c * i$$

Let $i = \log_2 \text{n}$, $T(n) = T(1) + \log_2 \text{n}$, $\theta(logn)$

For any number into this function, it should cost log(n) time whether it get into the first case or the second case. To conclude, the time cost of this algorithm is $\theta(logn)$.

Question5.

```
STEPWAY(stp)
    IF (stp == 1 OR stp == 0)
        RETURN 1;
    IF (stp == 2)
        RETURN 2;
    IF (stp == 3)
        RETURN 4;      // $T(1) + T(2) + 1$
    RETURN STEPWAY($stp$ - 1) + STEPWAY($stp$ -2) + STEPWAY($stp$ -3);
```

We know that:

If there is one steps, the way is: {**1**}

If there are two steps: the way is: {**1,1**}, {**2**}

If there are three steps: the way is {**1,1,1**}, {**1,2**}, {**2,1**},{**3**}

If there are four steps: the way is: {1, **1,1,1**}, {1, **1,2**}, {1, **2,1**}, {1, **3**}; {2, **1,1**}, {2, **2**}; {3, **1**}

We can find that:

Except for the first element, the case is same as the total of others. That is because the first element can be seen as the first steps you get. Therefore, the left is the case you can find.

$\mathrm{T(n)} = \mathrm{T(n-1)} + \mathrm{T(n-2)} + \mathrm{T(n-3)}$

Since the $n-1$ iterations must be more the others. It should take more time cost because if let it iteration, there will be: $T(n-1) = T(n-2) + T(n-3) + T(n-4)$. By that analogy to the basis case, $(n-i) = T(n-i-1) + T(n-i-2) + O(1)$

$T(n) < \mathrm{T(n-1)} + \mathrm{T(n-1)} + \mathrm{T(n-1)}$,

$= 3 * \mathrm{T(n-1)} = 3 * 3 * \mathrm{T(n-2)} = 3 * 3 * 3 * \mathrm{T(n-3)}$

$= \ldots = 3\text{^}\mathrm{i} * \mathrm{T(n-i)}$, in the i^{th} times iteration.

Let $\mathrm{i} = \mathrm{n} - 1$ and we get the $3^{\mathrm{n}-1} * \mathrm{T(1)} < 3^n$. So, the time cost is: $O(3^n)$

Question6.

Divide all the items into $[\frac{n}{4}]$ sets in which each, except possibly the last, contains 4 items. It just pays the $O(n)$. For each set, we should sort the 4 items and find the median one. The time cost is the $\frac{n}{4} * C \quad (C\ is\ a\ constant) = O(n)$. Take these $\frac{n}{4}$ medians and put them in another array. In the recursively calculate the median of these medians. So, the time cost of the recursion is $T\left(\frac{n}{4}\right)$. In other part, we should calculate the partition the original array by the pivot. If the p is the pivot of the array, p is the beginner of array and the r is the ender. $(k = p - q + 1)$

$T(n) \leq O(n) + T(n/4) + T(max\{k - 1, n - k\})$

We can find the how many elements are greater or less than x at least. There are $\frac{n}{2}$ sets which exists a median is bigger than pivot at least. Except the set contains x and the set with fewer than 4 items. There are only $\frac{1}{2} * \frac{n}{4} - 2$ sets with more than 2 items greater than the pivot. So, there are at least $2\left(\frac{1}{2} * \left\lceil\frac{n}{4}\right\rceil - 2\right) \geq \frac{n}{4} - 4$ items greater than pivot. In the contrary, there are at most $n - (\frac{n}{4} - 4)$ items smaller than pivot. $T(max\{k\ -\ 1, n\ -\ k\}) = T(\frac{3n}{4} + 4)$.

Induction: $T(n) \leq cn\ for\ some\ constant\ and\ \forall n$

$T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{3}{4}n + 4\right) + an$

$= c * \left\lceil\frac{n}{4}\right\rceil + c + \frac{3}{4}cn + 4c + an$

$= cn + (5c + an)$ must be greater than cn. For all the c and a is positive and c is large enough.

So the induction is false and we cannot say that the algorithm run in the linear time.