Recursion 1 && Sorting Algorithms

Fibonacci sequence:

- 0th fibonacci number is 0
- 1st fibonacci number is 1
- 2nd fibonacci number is 1
- 3rd fibonacci number is 2
- 6th fibonacci number is 8

subproblem: fib(n-1), fib(n-2)
rule: fib(n-1) + fib(n-2)
basecase: n == 0 && n == 1

```
public static long fib(int n){
        if (n == 0 {
                return 0;
        )
        if (n == 1){
                return 1;
        }
        return fib(n-1)+ fib(n-2);
}
```

```
               F（4）
               /     \
         F(3)           F(2)
        /     \          /   \
   F(2)    F(1)       F(1) F(0)
  /   \      /  \
F(1)F(0)   *   *       * *   * *
```

第1层: 1

第2层: 2

第3层: 4

第4层：8
第n层：1+2+4+8+.....=2^h -1

time: recursion，所有node时间的总和，看最后一层node；脑子里面想这个tree，不断的二分，直到不能分为止，2，4，8，.....2^h；
2^h：h ？h = n；so time is 2^N
space：recursion，直上直下粉红色的路径；On

call stack:
+ local memory allocated by computer system
+ used to record the local variable before the recursion function call
+ when we go back to the same level, we still can recover what happened in this level

Better Fibonacci sequence:

用dp，也就是slinding window来解决这个

```
public long fib(int n){

	if (n < 0)
		return 0;
	if (n == 0)
		return 0;

	if (n == 1)
		return 1;
	long  a = 0;
	long b = 1;
	for (int i = 0; i < n -1; i++){
		long temp = a+b;
		a = b;
		b = temp;
	}
	return b;
}
```

time: not recursion, On

space: stack:0; heap: 0 , so O1

Example question 2: power

how to calculate a ^b, eg a = 2, b = 1000
assumption: 1) a, b are int; 2) a > 0, b >= 0;

subproblem: power(a, b/ 2)
rule:
 b is odd: power(a, b/ 2) + * powder(a, b/2) * a;
 b is even: power(a, b/ 2) + * powder(a, b/2);
base case: b == 0, a = 1; power(a, 0) = 1

```
public long power(int a, int b){
	if (b == 0)
```

```
        return 1;
    long temp = power(a, b/ 2);
    if (b % 2 == 0)
        temp * temp;
    else
        temp * temp * a;
}
```

```
          F(3,  8)                 第一层：1
          /
        F（3,  4）                 第二层：1
        /
    F（3,  2）                     第3层：1
      /
  F（3,  1）                       第4层：1
    /
  F（3,  0）                       第5层：1
                                   第n层：还是1
```

time: recursion, 二分，高度：F8，F4，F2，F1， logn；层*高 = 1 * logn = O(lgon)

space: recursion， 看tree 的高度，直上直下，粉红色的路径，Ologn

Discussion:

（重点强调） For recursion function

1. **Time complexity** analysis: Depends on the **total number of nodes** in the recursion tree. We only need to sum up each node's actions.
2. **Space complexity** analysis: Depends on the height of the recursion tree, which is equal to the height call stack. We only need to sum up the nodes along the call stack **(直上直下粉红色路径)**

Code Review：做题要求

1：document my assumpations
2: explain my idea from the high level
3: provide my code comments, helper function comments，parameters comments
4: time , space analysis
5: why use this algorithm
6: only provide best answer

怎么给别人讲code：

1： 千万别一行一行讲,
2：arguments, function 定义, signarutes， 在做什么
3： 主要逻辑, 2个for loop, 每个干什么？

SelectionSort：

1：是什么：去个例子, 你有一摞卷子, 你从这些卷子里面找到最小的放在最上面, 再从剩下的卷子里找到第二小的放在最小的那个卷子的后面, 以此类推；

2：iteration 1： find the global min -3, {-1,-3,7,4} ⇒ -3 {-1, 7, 4 }
iteration 2： find the global min -1, {-1, 7, 4} ⇒ -3 -1{ 7, 4 }
iteration 3： find the global min 4, { 7, 4} ⇒ -3, -1, 4{ 7 }
iteration 4： find the global min 7, { 7} ⇒ -3, -1, 4, 7{ }

3: 2 nested for loop, the outer loop is the 挡板, the inner for loop is to find the global min

```
void selectionSort(int[] array, int n){
        if (array == null || array.length == 0){
                return;
        }

        for (int i = 0; i < n-1; i++){// 因为最后一个元素不需要sort, 外层是挡板, 当在最后一个元素时就结束
                int globalMin = i;
                for (int j = i + 1; j < n, j++){
                        if (array[j] < array[globalMin]){
                                globalMin = j;
                        }
                }
                swap(array, i, globalMin);
        }
private void swap(int[]a, int left, int right ){
        int temp = array[left];
        array[left] = array[right];
        array[right] = temp;
        }

}
```

时间复杂度分析:
```
for (int i = 0; i < n-1; i++) {//outer loop: how many iterations
     for (int j = i+1; j < n; j++)

iteration i = 0: inner       (0..n-1)   = 4
iteration i = 1: inner n-1   (1..n-1)   = 3
iteration i = 2: inner n-2   (2..n-1)   = 2
iteration i = 3: inner n-2   (2..n-1)   = 1
1+2+3+4+..+n = n(n+1)/2 -> n^2 → O(n^2)
Space = O(1)
```
Time = O(n^2)

question1: given an array stored in stack1, how to sort the numbers by using additional two stack

stack 1 input: 3 1 2 4 先把4 弹出来，golbalMin = 4 , 然后把4放到buffer里面
stack 2 buffer: 4
stack 3 result:

stack 1 input: 3 1 2 先把2 弹出来，golbalMin = 2 , 然后把2放到buffer里面
stack 2 buffer:4 2
stack 3 result:

stack 1 input: 3 1 先把1 弹出来，golbalMin = 1 , 然后把1放到buffer里面
stack 2 buffer:4 2 1
stack 3 result:

stack 1 input: 3 先把3 弹出来，golbalMin = 1 , 然后把3放到buffer里面
stack 2 buffer:4 2 1 3
stack 3 result:
这已经做了一轮了，现在找到了globalMin = 1； 所以 3 ！= globalMin，把3放回stack1；
+++
stack 1 input: 3 先把3 弹出来，golbalMin= 1， 3！= globalMIn然后把3放到input里面
stack 2 buffer:4 2 1
stack 3 result:

stack 1 input: 3 先把1弹出来，golbalMin 1 , 1== globalMin 然后把1放到result里面
stack 2 buffer:4 2
stack 3 result: 1

stack 1 input: 3 2   先把2弹出来，golbalMin 1 , 2 ! = globalMin 然后把2放到input里面
stack 2 buffer:4
stack 3 result: 1

stack 1 input: 3 2   4   先把4弹出来，golbalMin 1 , 4 ! = globalMin 然后把4放到input里面
stack 2 buffer:
stack 3 result: 1
第一轮外层for 循环结束，一次类推

这里selectionSort的思想是不变的，每轮晒出一个最小的，把每一轮都晒完，就好了

MergeSort:

```
                     index 0                  9
n space                  3,1,5,7,9,8,6,4,2,0    a[10] -> a[0]...a[4]  MERGE a[5]...a[9]
                            /               \
n/2 space            31579    line5.5   |        86420          O(1) → if LinkedList O(n)
                      /    \
n/4                 315 |  79              864 |   20           O(2) → if LinkedList O(n)
                   /  \                   /  \
n/8              31    5    79          86   4    2 0           O(4) → if LinkedList O(n)
                 /
1              3    1    5     7  9     8 6   4    2 0          O(n/2) → if LinkedList O(n)

        Time complexity above this line =   1 + 2 + 4 + ....  n/2  = O(n/2) = O(n)
                                          logn items because there are logn level
                                                     nlogn  if LinkedList

        how many levels above this line = log(n)
        space =   n + n/2 + n/4 + n/8 + n/16 + .... +  2+   1  = O(n) → O(log n) if LinkedList
        =====================================================================
        how many levels below this line = log(n)                 total time = O(nlogn)
        Time complexity below this line = O(nlogn)

           13   5      7  9      68   4    2 0   this level time complexity == O(n)
             \ /                   \ /
              135    79              468    02   this level time complexity == O(n)
                 \   /                  \   /
                 13579                   02468   this level time complexity == O(n)
                      \                 /
                        0123456789
```

时间复杂的分析：
1：用了recursion，双横线以上，mid 就只是把数组砍一刀，砍2到，所以时间就是O1，通过每一层，可以看出来是 1 2 4 8,
 所以1+2+4+8+16+... = 2^N；应该是等比数列，exponation的增长；但为什么双横线以上时间为On呢？
因为 2分法，每一层是1，2，4，8，一共有多少层呀？logn层呀，是logn个items，不是N个items，所以一共是logn项；
2^logn = 2 ^ log2 ^n = n；因为底数相同，遵循这个公式：aloga^b = b，所以双横线以上是On

2：双横线以下，每一层是On，一共logn 层， 所以是Onlogn
3：Total：On + Onlogn = Onlogn；

空间复杂度的分析：因为用了recursion，双横线以上，直上直下，粉红色的路径；
n+n/2+n/4+n/8+....~~2^n; 一共有logn层，所以是2^logn = n；~~ 所以是On；
n+n/2+n/4+n/8+...+1= 2n-1: 等比数列求和公式，所以是On；

Secondly, you missed a clause mentioned at the linked StackOverflow question: the correct statement is that **if n is a power of 2,** then $\lfloor n \rfloor + \lfloor \frac{n}{2} \rfloor + \lfloor \frac{n}{4} \rfloor + \ldots$ (which in this case is the same as $n + \frac{n}{2} + \frac{n}{4} + \cdots + 1$) is exactly $2n - 1$.

For this special case when n is a power of 2 (which is what it takes for all the numbers $\frac{n}{2}, \frac{n}{4}, \ldots$ to be integers, all the way to 1), this is easy to prove. When n is a power of 2, say $n = 2^k$, the sum is

$$2^k + \frac{2^k}{2} + \frac{2^k}{4} + \cdots + 1 = 2^k + 2^{k-1} + 2^{k-2} + \cdots + 1 = 2^{k+1} - 1 = 2n - 1$$

For example, $16 + 8 + 4 + 2 + 1 = 31$.

This can be proved by induction, or it follows from the formula for the geometric series, which states that

$$a + ar + ar^2 + \cdots + ar^{m-1} = a\frac{1 - r^m}{1 - r}.$$

In this case, we have $a = n = 2^k$, $r = 1/2$, and m (the number of terms) is $k + 1$, so the left-hand side is the sum $n + \frac{n}{2} + \cdots + 1$, and the right-hand side is

$$n\frac{1 - (1/2)^{k+1}}{1 - (1/2)} = n\frac{2 - (1/2)^k}{1} = 2n - n/2^k = 2n - 1.$$

时间：在recursion的情况下，不是多少个node，就是多少的时间复杂度，而是每个node花费时间的总和，或者是每一层加起来

空间：在recursion的情况下，90%是这个recursion tree的高度，但不严谨，mergesort就不是，应该是直上直下，粉红色路径的总和；

Question3：could we use merge sort to sort a linkedlist? what is the time complexity if so？

yes, not change the time and space complexity.

time:
1:linkedlist 找中点是On的时间复杂度，所以横线以上都是On，一共logn层，所以时间是Onlogn，的确比mergesort横线以上的（On）增加了，但是总的不变；说横线一下还是nlogn
total：Onlogn +nlogn =nlogn

space: 一个recursion function，他的空间和什么有关？直上直下，粉红色的路径；总共有多少个node？logn 个node，所以是logn；这个就死记硬背把现在；

mergesort算法是递归的，因此对于数组和链表大小写，它需要O（log n）堆栈空间。但是数组的情况下还分配了一个额外的O（n）空间，它占据了堆栈所需的O（log n）空间。所以数组版本是O（n），链表版本是O（log n）。

Question 4: given a string A1B2C3D4, how to convert it to another string ABCD1234

```
                    A1B2C3D4
              A1B2    |         C3D4
     A1   B2                 C3    D4
   A 1    B   2....
   ========================================
   A1    B2
         AB12                CD34                2nd last step
              ABCD1234                           last step

1st   A
        i
2nd   1
      j
solu = {A 1}
```

String 也可以sort，可以用ASCII 码来转换

1：横线以上和mergesort一样，都是中间切一刀

2：横线一下是如何merge 一个string和数字？

1st　　A：65

　　　　i

2nd　　1：49

　　　　j

soul = {}

谁小移动谁，letter和letter是alphbet order，数字和数字之间是nature order

3 QuickSort

时间：

average case

1) 不看下面2个递归, 只看递归以上的代码, 它的bottle neck 在哪? while loop; while loop 最多会执行多少次? 如果用left 和right 表示的话? R-L; 和R和L 的距离成正比; 所以quickSrot调用一次是R-L 的时间复杂度;
2) 总共调用多少次呢?

```
            n                                        n
      n/2          n/2
                                           n/2+n/2=2n/2=n
  n/4    n/4      n/4    n/4
                                       n/4+n/4+..=4n/4=n
111111111111    1111111111111                 n
```

总共调用了n次,但recursion 的高度是logn, 也就是层数是logn, 只要能被 不断2分的,都是logn,每层以 n/2 的速度来减小; n(1/2)^x = 1

3) n* logn

空间：
average case

stack:取决于recursiontree 的高度,是一个宝塔型的结构,每一次也就是哪些logcal variable, 是O1, 总共多少层? == tree的高度; 是logn; callstack的高度是logn;这里是每次砍掉一半;

heap: 0

logn

question 7：what is the worst case of quicksort？can you provide some examples?

1: privot 要么选的是最大的，要么选最小的，每次都要把另一边多有的都要排一遍；
时间：

worst case
点背的情况,就是总是找privot 是最大的, 或最小的, 一边是
n - 1 长度; 另一边是 没有;   然后又是 n - 2 长度, 没有;

n
n - 1
n - 2
n - 3

n + (n-1)+(n-2)+(n-3)+... = n^2

3)   O (n^2)

空间：

worst case
stack: n
n
n - 1
n - 2
n - 3
高度是 n, n调用n-1, 然后n-1再调用n-2, 调用关系是 n, 所以高度是n;这里是每次砍掉1;
heap: 0

O(n)

quicksort一般都是在java和c++内层所实现的方法，因为一般不会那么坏；

question 8：Array Shuffling

Question 8: Array Shuffling

Given an array with integers, **move** all “0s” to the right-end of the array. (2个挡板, 3个区域)

Example int[] a = {1, 0, -4, -5, 0, 2, 3}; → { 1, -4, -5, 2, 3, 0, 0 }

index	0	1	2	3	4	5	6
A[7]	**1**	**0**	**-4**	**-5**	**0**	**2**	**3**

i → ← j

initialization (2个挡板，3个区域，相向而行):

i = 0 → : all numbers to the left-hand side of i **(not including i)** are all “non-zero”s; i is the **current index** to move

[i, j] : (unknown area to explore);

j = ← size-1 : all numbers to the right hand side of j **(not including j)** are all “zero”s ,

所有非0 归到右边，i的左侧不包含i都是非0

j的右侧不包含j都是 0

相向而行就可以了；

question 9：rainbowsort

aaaaaa bbbbbb XXXXX CCCCC

i j k

i: i的左侧不包含i都为a；

j: j的左侧不包含j都为b；

k：k的右侧，不包含k，都是c

case1：if input【j】== a；

aaaaaa bbbbbb AXXXX CCCCC

i j k

违背了rule1，因为i的左侧不包含i，都是a，所以swap（i，j），

aaaaaa Abbbbb bXXXX CCCCC

i j k

需要 i++， j++; 保持i的左侧不包含i都为a，j的左侧不包含j都为b

```
aaaaaa Abbbbb  bXXXX CCCCC
         i        j    k
```

case2：if input【j】== b；

```
aaaaaa bbbbbb  BXXXX CCCCC
        i       j      k
```

这里很简单，只需要j++

```
aaaaaa bbbbbb  BXXXX CCCCC
        i         j    k
```

case3：if input【j】== c；

```
aaaaaa bbbbbb  CXXXX CCCCC
        i       j      k
```

违背了rule3，k的右侧，不包含k都是c；swap（j，k）

```
aaaaaa bbbbbb  XXXXC CCCCC
        i       j      k
```

还需要k--，保持rule3；至于j是否需要--，我们并不知道，因为swap过来的是个未知数x

```
aaaaaa bbbbbb  XXXXC CCCCC
        i       j    k
```

```
public int[] rainbowSort(int[] array) {
   // Write your solution here
   if (array == null || array.length == 0){
     return array;
   }
   int i = 0;
   int j = 0;
   int k = array.length - 1;
   while (j <= k){
     if (array[j] == 1){
       swap(array, j,k);
       k--;
```

```
    }else if (array[j] == 0){
      j++;
    }else{
      swap(array, i,j);
      i++;
      j++;
    }
  }
  return array;
}
private void swap(int[] array, int left, int right){
  int tmp = array[left];
  array[left] = array[right];
  array[right] = tmp;
}
```

question10：four color or even more（Systematic way of thinking）

aaaaaa bbbbbb cccccc XXXXX dddddddddd
i j k t

i：i的左侧不包含i，都是a
j：j的左侧不包含j，都是b
k：k的左侧不包含k，都是c
t：t的右侧不包含t，都是d
caseA：

aaaaaa bbbbbb cccccc AXXXX dddddddddd
i j k t
违背了rule1；先swap（j，k）
aaaaaa bbbbbb Accccc cXXXX dddddddddd
i j k t
违背了rule1；再swap（j，i）

aaaaaa Abbbbb bccccc cXXXX dddddddddd
i j k t
i++；j++；k++；
aaaaaa Abbbbb bccccc cXXXX dddddddddd
i j k t

caseB：

aaaaaa bbbbbb cccccc BXXXX dddddddddd
i j k t
违背了rule2； swap（j，k）

aaaaaa bbbbbb Bccccc cXXXX dddddddddd
i j k t
j++； k++；

aaaaaa bbbbbb Bccccc cXXXX dddddddddd
i j k t

caseC：

aaaaaa bbbbbb cccccc CXXXX dddddddddd
i j k t
违背了rule3；k++；

aaaaaa bbbbbb cccccc CXXXX dddddddddd
i j k t

caseD：

aaaaaa bbbbbb cccccc DXXXX dddddddddd
i j k t
违背了rule4：swap（k，t）
aaaaaa bbbbbb cccccc XXXXD dddddddddd
i j k t

t--；
aaaaaa bbbbbb cccccc XXXXD dddddddddd
i j k t

1	2	3	4	5	-1	-1	-1
i					j		

clarity: input is a int[] array, output is int[] array, clarity the question, make sure I understand the question 100%
assumption: 是不是需要inplace？ 如果不是inplace sol = {1，-1，2，-1}，i++， j++；
result:
test:

1	2	3	4	5	-1	-1	-1
	i				j		

i：current index

j: is the 1st neg element is in the input, swap()

```
public int[] rainbowSortII(int[] array) {
    // Write your solution here
    if (array == null || array.length == 0){
      return array;
    }
    int i=0, j = 0, k = 0, t = array.length -1;
    while (k <= t){
        if (array[k] == 0){
          swap(array,k,j);
          swap(array,i,j);
          i++;
          j++;
          k++;
        }else if (array[k] == 1){
          swap(array,k,j);
          j++;
          k++;
        }else if (array[k] == 2){
          k++;
        }else{
          swap(array,k,t);
          t--;
        }
    }
    return array;
  }
  public void swap(int[] array, int left, int right){
        int tmp = array[left];
```

```
        array[left] = array[right];
        array[right] = tmp;
    }
```

aaaaaa bbbbbb cccccc dddddd XXXXX eeeeee

i j k t m

i：i的左侧不包含i，都是a
j：j的左侧不包含j，都是b
k：k的左侧不包含k，都是c
t：t的左侧不包含t，都是d
[t m]： to be partitioned area
m: m的右侧不包含m，都是e

cast A:
aaaaaa bbbbbb cccccc dddddd AXXXX eeeeee

i j k t m

swap(t,k)
aaaaaa bbbbbb cccccc Adddddd dXXXX eeeeee

i j k t m

swap(k,j)
aaaaaa bbbbbb Accccc cddddd dXXXX eeeeee

i j k t m

swap(j,1)
aaaaaa Abbbbb bccccc cddddd dXXXX eeeeee

i j k t m

i++, j++, k++,t++;
aaaaaa Abbbbb bccccc cddddd dXXXX eeeeee

i j k t m

cast B:
aaaaaa bbbbbb cccccc dddddd BXXXX eeeeee

i j k t m

swap(t,k);
aaaaaa bbbbbb cccccc Bddddd dXXXX eeeeee

i j k t m

swap(k,j)

aaaaaa bbbbbb Bccccc cddddd dXXXX eeeeee
i j k t m

j++; k++; t++;

aaaaaa bbbbbb Bccccc cddddd dXXXX eeeeee
i j k t m

cast c:
aaaaaa bbbbbb cccccc dddddd CXXXX eeeeee
i j k t m

swap(t, k)

aaaaaa bbbbbb cccccc Cddddd dXXXX eeeeee
i j k t m

k++;
t++;

aaaaaa bbbbbb cccccc Cddddd dXXXX eeeeee
i j k t m

cast: D
aaaaaa bbbbbb cccccc dddddd DXXXX eeeeee
i j k t m

t++;

aaaaaa bbbbbb cccccc dddddd DXXXX eeeeee
i j k t m

```
cast E:
aaaaaa bbbbbb  cccccc dddddd EXXXX   eeeeee
            i           j          k          t        m


swap(t, m)

aaaaaa bbbbbb  cccccc dddddd xXXXE   eeeeee
            i           j          k          t        m

m--;

aaaaaa bbbbbb  cccccc dddddd xXXXE   eeeeee
            i           j          k          t      m
```