Ada hw1 b03902107 鄭格承

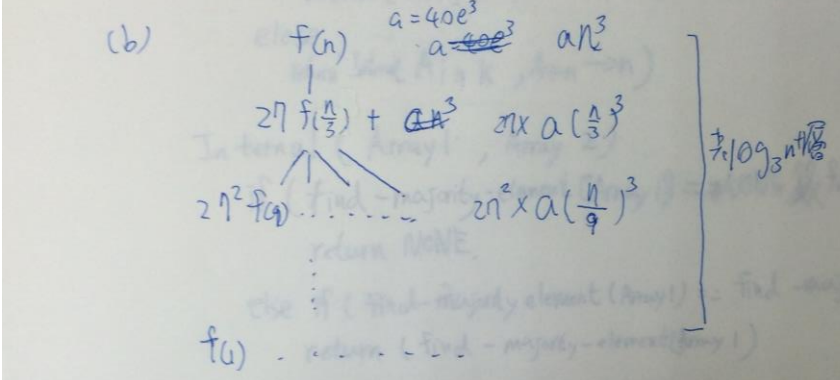
Problem 1.

(1) (a) $f(1) = 1 \leq cn^2 \Rightarrow c \geq 1$,

$f(n) = 16(\frac{n}{4}) + 514n < 16c(\frac{n}{4})^2 + 514n \leq c'n^2$

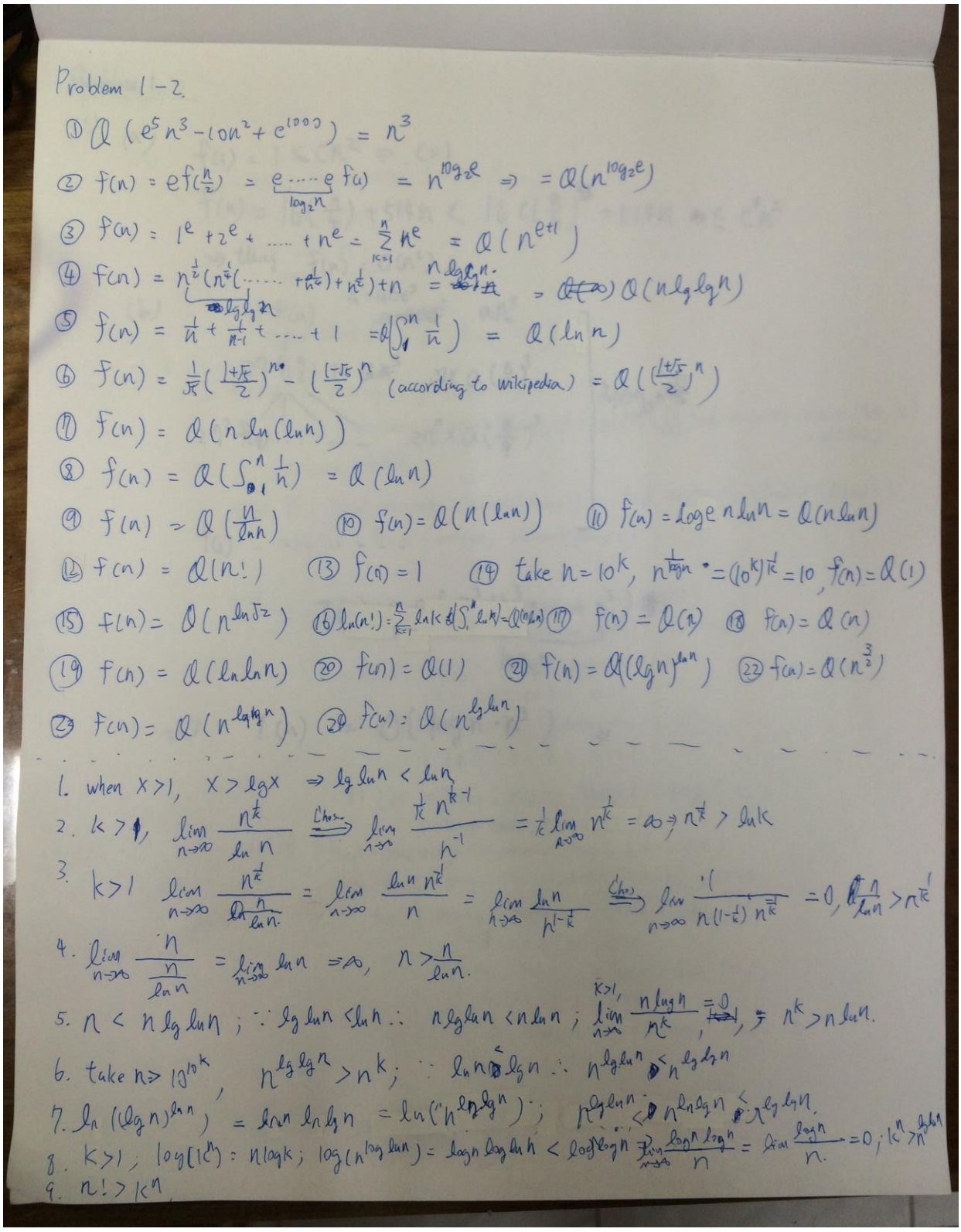
$\Rightarrow$ thus $f(n) = O(n^2)$ ※

(b)



$T(n) \leq a(n^3 + n^3 + \cdots + n^3)$ ($\log_3 n$ 項)

$\Rightarrow T(n) = O(\log n \cdot n^3)$ ※



As consequence:

$$[\,2^{10000}\,,2147483647\,,\,n^{1/\lg n}\,]<\ \lg\ln n\ <[\,f(n)=f(n-1)+\tfrac{1}{n}\,,\ \textstyle\sum_{i=1}^{n}\tfrac{1}{i}\,]<$$

$$(\sqrt{2})^{\ln n}\ <\ \tfrac{n}{\ln n}<[\,e^{\ln n}\,,\,\tfrac{10n}{e}\,]<\ \ en\lg(\ln n)\ <f(n)=\sqrt{n}f(\sqrt{n})+n\ <[\,\ln n!\,,\,n\lg n\,,$$

$n \ln n$]< $f(n) = ef(n/2)$ < $n^{3/2}$ < $e^5 n^3 - 10n^2 + e^{1000}$ < $f(n) = f(n-1) + n^{\varepsilon}$ <

$(\lg n)^{\ln n}$ < $n^{\lg \ln n}$ < $n^{\lg \lg n}$ < $f(n) = f(n-1) + f(n-2)$ < $n!$

Problem 2.

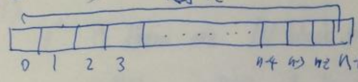
(1) (a)

```
find-majority-element ( array Ai→n)
    K=(i+n)/2.
    if(i == n)
        return ( A[i])
    elese
        return Internal( Ai→k , Ak+1→n)

Internal ( Array1 , Array 2)
    if (find-majority-element (Array 1) = NONE && find-majority-element (Array 2) = NONE)
        return NONE.
    else if ( find-majority element (Array1) == find-majority-element (Array2))
        return ( find-majority-element(Array 1)

    else
        if ( find-majority-element(Array1) != NONE)
            K = find-majority-element (Array1) ; n=0
            for (i = Array1.begin; i< Array1.end, i++)
                if ( A[i] == k), n++;
            for( i = Array2.begin, i <Array2.size, i++)
                if ( A2[i] = k) n++;
            if (n >(Array1.size+ Array2.size)/2.)
                return K
        else if (find-majority-element(Array2 != NONE)
            K= find-majority-element (Array1) ; n=0
            for ( i =Array1.begin , i<Array1.size, i++)
                if (A1[i] ==k) n++
            for ( i = Array2.begin, i<Array2.size, i++)
                if (A2[i] =k )  n++
            if (n > (Array1.size + Array 2.size) /2)
                return k;

    else.
        return false.
```

2-1-(b)

```
find_majority_element(Array A)
  //if two element is different, they can pair up. If
    there is majority_element, it couldn't be pair up by our rule.
  int, blumber=, non_pair_num = 0;
  for (i=0 ⟶ i<n , i++)
      if (non_pair_num = 0)
          blumber = A[i]
          non_pair_num ++.
      else
          if (blumber = A[i])
              non_pair_num ++
          else.
              non_pair_num --.
  check blumber.
      For(i=0, j=0, i<n, i++)
          if (A[i] == blumber)
              j++
          if(j > n/2)
              return j.
  return NONE.
```

2-2-(a)



sequential_merge : 1st [0] 1

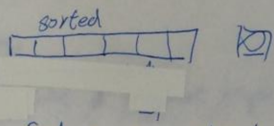
2nd [0 1] [2] + 2 (sort)

3rd [0 1 2] [3] + 3

⋮

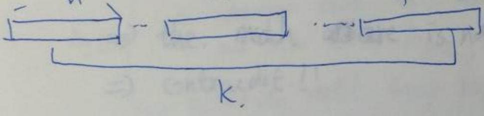
nth [0 1 ... n-2] [n] + n

=

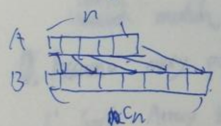


find a proper place to insert for a sorted array with size k, take at most $O(k)$

2-2-(a)



① 1° merge a n-size-array A into cn-size-array B, A, B were sorted.



⇒ take $O(cn)$ #.

Sequential merge : n n n ... n ⇒ take time: n + 2n + ... + (k-1)n

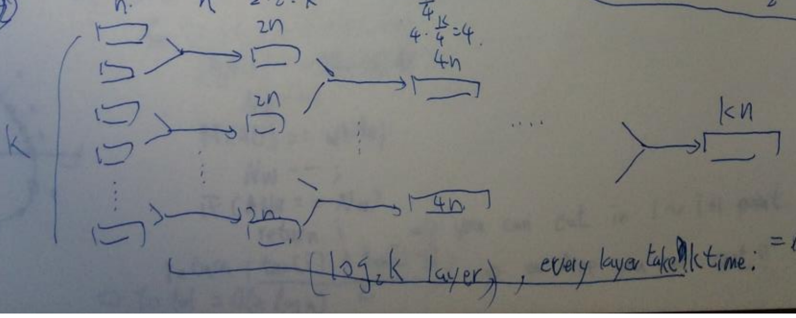
1st, 2nd, ..., (k-1)th

$= \frac{1}{2}k(k-1)n = O(k^2)$ #

(b)

② k

n. n $\frac{k}{2}$. $2 \cdot \frac{k}{2} = k$ 2n; $\frac{k}{4}$ $4 \cdot \frac{k}{4} = 4$. 4n



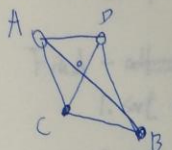
($\log_2 k$ layer), every layer take k time: $= O(k\log k)$ #

3. (1)

1° If there are N black points and N white points, they can have ~~this~~ N! different kinds of connections.

And ~~one of them the~~ the minium total segement length must exist. in the N! different connections.

2° if ~~the exist~~ good-word match doesn't exist. At least one interection. (the shortest match has)



for $\overline{OA}+\overline{OC} > \overline{AC}$, $\overline{OD}+\overline{OB} > \overline{DB}$

$\overline{OA}+\overline{OC}+\overline{OD}+\overline{OB} = \overline{AB}+\overline{CD} > \overline{AC}+\overline{DB}$

⇒ if ~~interection~~ intersection exist, there is at least another connection way that ~~have~~ has smaller segment length sum.

⇒ the match are is not the shortest.

⇒ contracdit.!!

3° thus, there must has a minium total segement length sum, and it is a good word ~~world~~ match.

(2)

0. Assume every point ~~exis~~ store is struct Point(x,y), All point store in an Array Point[N]

1' Sort Array by it's x value (Merge-sort, O(nlogn)) (small to big)

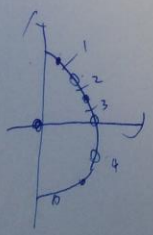
Array.Sort(). ~~(assume it's black)~~,

2° take A[0] as basic point., O(1) (assume it's black; N_B & N_W for black point & white point nu

3° count the tan[] of A[0]A[1] in array[1]~array[n], put into A array tan[n-1]

```
for(i=1, i<n ; i++)
tan[i-1] = (A[i].y - A[i].y) / (A[i].x - A[0].x)   (if A[i].x - A[0].x = 0 ↗ if A[i].x-A[0].y >0, tan[i-1]=
                                                    ↘ if A[i].x-A[0].y >0, tan[i-1] =
```

4° sort A[1]~A[n] with tan[n-1] (Merge sort) O(nlogn)




```
for(i=0 ; i<n-1; i++)
  if(A[i] == Black)
    N_B --;
  if(A[i] == White)
    N_W --;
  if(N_B == N_W)
    return i        ⇒ you can cut in i~i+1 point.
return (tan[i] + tan[i-1]) / 2   ⇒ a line you can cut!!
```

⇒ total = O(nlogn)

(3)

according to (1) (2):

~~when you find a point~~

1° take one point as basic point, we can always find another point if we line up with it, the left and right planes are both another subset of points, and they can make two good-word situations.

2° take all points ~~in right~~ from left-most, we take $\frac{n}{2}$ times, we can find all

Find-~~all point~~ a-good-word (Points)

1. sort points with x from small to big ($O(\log n)$)

$\frac{n}{2}$ times:

2. use (2) to find a link. ($O(n \log n)$)
3. ~~take~~ take this two point away, others put into array A_{n-2}, with the origin-x-sort order.
4. redo until Array has no point.

$\Rightarrow$ big O is $O(n \cdot n \log n)$ ∝