#HW1

姓名：宋子維 系級：資工二 學號：宋子維

Problem 1

(1) Solve the recurrences.

a. $f(n) = 16f(\frac{n}{4}) + 514n,\ prove\ that\ f(n) = O(n^2)$.

Suppose f(n)≤cn^2-dn where c is a positive constant and d>0, and we assume the bound above holds when m < n. Hence, it also holds when m = $\lfloor \frac{n}{4} \rfloor$. That is, $f(\lfloor \frac{n}{4} \rfloor) = f(\frac{n}{4}) \le c\frac{n^2}{16} - dn$.

$$\begin{aligned} \Rightarrow f(n) &= 16f(\tfrac{n}{4}) + 514n \\ &\le 16c\tfrac{n^2}{16} - 16dn + 514n \\ &= cn^2 - 16dn + 514n \\ &\le cn^2 - dn,\ if\ d\ \ge \tfrac{514}{15} \end{aligned}$$

Check boundary:

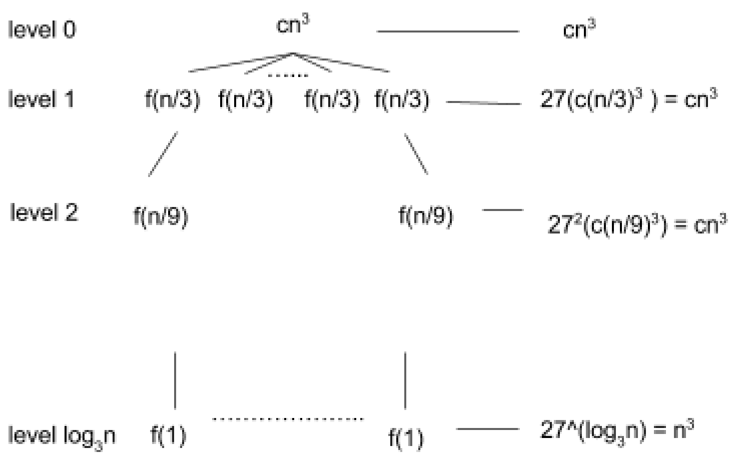
$f(1) = 16f(\frac{1}{4}) + 514 = 530$, $f(2) = 16f(\frac{2}{4}) + 1028 = 1044$

$\Rightarrow f(1) \le c - \frac{514}{15}\ and\ f(2) \le 4c - \frac{1028}{15}\ by\ taking\ c = 565$

$\Rightarrow f(n) \le cn^2 - dn\ holds\ for\ n \ge 1\ by\ induction \Rightarrow f(n) = O(n^2)$

b. $f(n) = 27f(\frac{n}{3}) + 40e^3n^3,\ prove\ that\ f(n) = O(n^3 log\ n)$

$let\ c = 40e^3$



$$f(n) = \left(\sum_{i=0}^{log_3 n-1} 27^i c\left(\frac{n}{3^i}\right)^3\right) + n^3 = \left(\sum_{i=0}^{log_3 n-1} cn^3\right) + n^3$$

$$= cn^3(log_3 n) + n^3 \le cn^3 log\, n + n^3 \in O(n^3 log\, n)$$

$Proof\ that\ f(n) \le dn^3 log\ n,\ d\ is\ a\ positive\ constant$

$f(2) = 27f(\frac{2}{3}) + 320e^3 = 27 + 320e^3\ \&\ f(3) = 27f(\frac{3}{3}) + 1080e^3$

$As\ loog\ as\ d\ is\ large\ enough,\ f(2), f(3) \le dn^3 log\ n.$

$Suppose\ f(m) \le dm^3 log\ m\ holds\ when\ m < n.\ Then\ it\ also\ holds\ when\ m = \frac{n}{3}.$

$That\ is,\ f(\frac{n}{3}) \le d(\frac{n}{3})^3 log(\frac{n}{3})$

$f(n) = 27f(\frac{n}{3}) + 40e^3 n^3 \le 27d(\frac{n}{3})^3 log(\frac{n}{3}) = dn^3 log(\frac{n}{3}) \le dn^3 log\ n$

$Hence,\ f(n) \le dn^3 log\ n\ holds\ for\ n \ge 1\ by\ induction.$

$\Rightarrow f(n) = O(n^3 log\ n)$

(2) Discuss with B03902024

reference:

https://en.wikipedia.org/wiki/Fibonacci_number

http://www.wolframalpha.com/input/?i=f%28n%29%3Dsqrt%28n%29f%28sqrt%28n%29%29%2Bn

http://stackoverflow.com/questions/25905118/finding-big-o-of-the-harmonic-series

http://www.av8n.com/physics/stirling-factorial.htm

$a.\ e^5 n^3 - 10n^2 + e^{1000} = \Theta(n^3)$

$b.\ f(n) = ef(\frac{n}{2}) = \Theta(n^{log\ e})$

$c.\ f(n) = f(n-1) + n^e$

$= f(n-2) + (n-1)^e + n^e$

$= f(n-3) + (n-2)^e + (n-1) + n^e$

$= \ ...$

$= f(1) + \sum_{i=2}^{n} i^e$

$And\ \frac{1}{e+1}(n^{e+1} - 1) = \int_1^n x^e dx \le \sum_{i=2}^{n} i^e \le \int_2^{n+1} x^e dx = \frac{1}{e+1}((n+1)^{e+1} - 2^{e+1}) = \frac{1}{e+1}(n^{e+1} + (e+1)n^e ...)$

$Thus,\ f(n) = f(1) + \sum_{i=2}^{n} i^e = \Theta(n^{e+1})$

$d.\ f(n) = \sqrt{n} f(\sqrt{n}) + n,\ prove\ f(n) \le cnlog(log\ n) + dn = O(nlog(log\ n)),\ c > 0,\ d > 0$

$f(3) = \sqrt{3}f(\lfloor\sqrt{3}\rfloor) + 3 = 3 + \sqrt{3} \le c3log(log\ 3) + 3d$ *by taking* $c \ge \frac{3+\sqrt{3}}{3log(log\ 3)},\ d > 0$

Suppose $f(m) \le cmlog(log\ m) + dm$ *holds when* $m < n$.

Since $\sqrt{n} < n$, *then* $f(\sqrt{n}) \le c\sqrt{n}log(log\sqrt{n}) + d\sqrt{n}$ *alse holds.*

$$f(n) = \sqrt{n}f(\sqrt{n}) + n \le \sqrt{n}(c\sqrt{n}log(log\sqrt{n}) + d\sqrt{n}) + n = cnlog(log\sqrt{n}) + (d+1)n$$

$$= cnlog(\tfrac{1}{2}logn) + (d+1)n = cn(log\tfrac{1}{2} + log(logn)) + (d+1)n$$

$$= cnlog(logn) + (d+1-c)n \le cnlog(logn) + dn \text{ by taking } c \ge 1$$

$\Rightarrow$ $f(n) \le cnlog(logn) = O(nlog(logn))$ *by induction.*

$f(n) = \Omega(nlog(logn))$ *can be proved similarly.*

$\Rightarrow$ $f(n) = \Theta(nlog(logn))$

$e.\ ln\ n = \int_1^n \frac{1}{x}dx < \sum_{i=1}^{n} \frac{1}{i} \le \int_1^{n+1} \frac{1}{x}dx = ln(n+1),\ \ \sum_{i=1}^{n} \frac{1}{i} = \Theta(logn)$

$f.\ f(n) = f(n-1) + \frac{1}{n}$

$$= f(n-2) + \frac{1}{n-1} + \frac{1}{n}$$

$$= f(n-3) + \frac{1}{n-2} + \frac{1}{n-1} + \frac{1}{n}$$

$$= \ldots$$

$$= f(1) + \sum_{i=2}^{n} \frac{1}{i} = \Theta(log\ n) \text{ by the result above}$$

$g.\ f(n) = f(n-1) + f(n-2) = \frac{\varphi^n - (1-\varphi)^n}{\sqrt{5}} = \Theta(\varphi^n),\ where\ \varphi = \frac{1+\sqrt{5}}{2}\ (Fibonacci\ sequence)$

$h.\ enlg(ln\ n) = en\frac{log(\frac{log\ n}{log\ e})}{log\ 10} = \Theta(nloglogn)$

$i.\ \frac{n}{ln\ n} = \frac{n}{(\frac{log\ n}{log\ e})} = \frac{(log\ e)n}{log\ n} = \Theta(\frac{n}{logn})$

$j.\ nlg\ n = \Theta(nlog\ n)$

$k.\ nlog\ n = \Theta(nlog\ n)$

$l.\ n! = \Theta(n!)$

$m.\ 2147483647 = \Theta(1)$

$n.\ let\ n^{\frac{1}{lgn}} = x,\ then\ 1 = (lgn)(\frac{1}{lgn}) = lgx,\ x = 2 = n^{\frac{1}{lgn}} = \Theta(1)$

$o.\ (\sqrt{2})^{ln\ n} = n^{ln\sqrt{2}} = n^{\frac{ln\ 2}{2}} = \Theta(n^{\frac{ln\ 2}{2}})$

$p.\ nln\ n - n + 1 = \int_1^n lnxdx < ln\ n! = \sum_{i=1}^{n} ln\ i < \int_1^{n+1} lnxdx = (n+1)ln\ n - (n+1) + 1$

$\Rightarrow$ $ln\ n! = \Theta(nln\ n) = \Theta(nlogn)$

$q.\ e^{ln\ n} = n^{ln\ e} = n = \Theta(n)$

$r.\ \frac{10n}{e} = \Theta(n)$

$s.\ lgln\ n = \Theta(log(ln\ n))$

$t.\ 2^{10000} = \Theta(1)$

$u.\ (lgn)^{ln\ n} = n^{lnlgn} = \Theta(n^{\ lnlogn})$

$v.\ n^{\frac{3}{2}} = \Theta(n^{\frac{3}{2}})$

$w.\ n^{lglg\ n} = \Theta(n^{loglog\ n})$

$x.\ n^{lgln\ n} = \Theta(n^{logln\ n})$

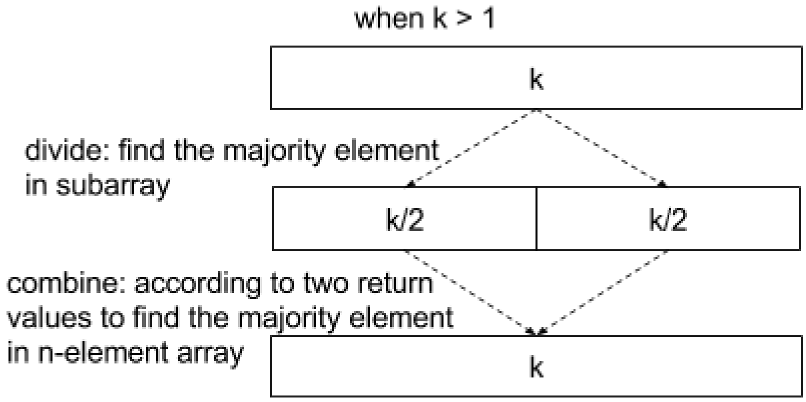
	Complexity		Complexity
$C_1 = \{n^{\frac{1}{lgn}},\ 2147483647,\ 2^{10000}\}$	1	$C_{10} = \{n^{\frac{3}{2}}\}$	$n^{\frac{3}{2}}$
$C_2 = \{lgln\ n\}$	$log(ln\ n)$	$C_{11} = \{e^5 n^3 - 10n^2 + e^{1000}\}$	n^3
$C_3 = \{f(n) = f(n-1) + \frac{1}{n},\ \sum_{i=1}^{n} \frac{1}{i}\}$	$logn$	$C_{12} = \{f(n) = f(n-1) + n^e\}$	n^{e+1}
$C_4 = \{(\sqrt{2}\ )^{ln\ n}\}$	$n^{\frac{ln\,2}{2}}$	$C_{13} = \{(lgn)^{ln\ n}\}$	$n^{lnlog\ n}$
$C_5 = \{\frac{n}{ln\ n}\}$	$\frac{n}{logn}$	$C_{14} = \{n^{lgln\ n}\}$	$n^{logln\ n}$
$C_6 = \{\frac{10n}{e},\ e^{ln\ n}\}$	n	$C_{15} = \{n^{lglg\ n}\}$	$n^{loglogn}$
$C_7 = \{f(n) = \sqrt{n}f(\sqrt{n}) + n,\ enlg(ln\ n)\}$	$nlog(logn)$	$C_{16} = \{f(n) = f(n-1) + f(n-2)\}$	$(\frac{1+\sqrt{5}}{2})^n$
$C_8 = \{nlogn,\ nlgn,\ ln(n!)\}$	$nlogn$	$C_{17} = \{n!\}$	$n!$
$C_9 = \{f(n) = ef(\frac{n}{2})\}$	$n^{log\ e}$		

Problem 2

(1)

a. reference: https://classes.soe.ucsc.edu/cmps102/Fall01/solutions4.pdf

The algorithm applies Divide and Conquer method and it is similar to merge sort algorithm. First, split the array into left and right repeatedly and call itself on two subarrays. When the size of array is just one, the only element is returned as majority element (base case). As for recursive case, it is shown as below.



There are four condictions of two return values when combining:

1. Both left and right subarrays **return NONE**. There is no element appearing k/4 times in the left subarray, neither is the right subarray. Therefore, after combining two subarrays into one, there is no element appear k/2 times in k-element array, that is, there is no majority element in n-element array. ⇒ **return NONE**
2. The left subarray **returns majority element**, but the right one **returns NONE**: Since left subarray has element appearing k/4 times and the right one doesn't, the only possibility to majority value in k-element array is majority element in the left subarray. Therefore, just check each element in the combined array and count how many times do the majority element in left subarray appear. If it appears more than k/2 times, then **return its values**; otherwise, **return NONE**.
3. The left subarray **returns NONE**, but the right one **returns majority element**: Similar to 2., just count the number of right majority in combined array. if it appears more than k/2 times, then **return its value**; otherwise, **return NONE**.
4. Both left and right subarrays **returns its majority element**: Count the number of those two majority elements from left and right in combined array. If one of them appears more then k/2 times, then **return it**; otherwise, **return NONE**.

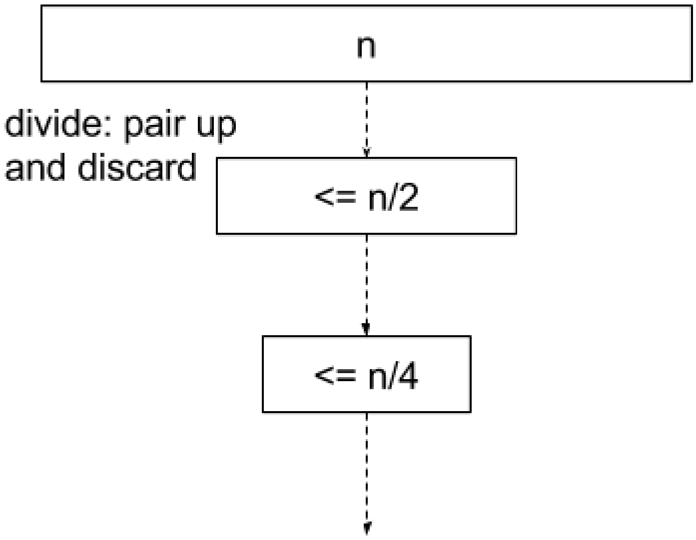
For non-recursive cost, it takes O(n) to count (and split) the elements. And in recursive case, it takes 2T(n/2) to call itself.

⇒ T(n) = 2T(n/2) + O(n)

⇒ T(n) = O(nlog n) by Master Theorem.

b. reference: http://www.ece.northwestern.edu/~dda902/336/hw4-sol.pdf

The algorithm also applies Divide and Conquer method. The main idea is to pair up two elements in the array to get n/2 pairs. Then how to deal with these pairs? Suppose there is a pair (p, q). If p is equal to q, then keep p and discard q. If p is not equal to q, then dicard both of them. And then, the majority element (if it exists) in the array formed by remaining elements is the same as the one in the origin array. Hence, it calls itself again.



The base case occurs when size = 1 and size = 0. The first one **returns the only element** and the second one **returns NONE**. Check correctness: Suppose m is the majority element in the array. After the first round of pairing up, there are p * (m, m) and q * (m, a) where a≠ m. Hence, we can find the two relations below

$\Rightarrow 2p+q>n/2 \qquad -(1)\ and$

$2p+2q=n \qquad -(2)$

$\Rightarrow 2p>0 \qquad -2*(1)-(2)$

$\Rightarrow p>0$

$remaining\ numbers: n-p-2q \qquad -(3)$

$remaining\ m : p \qquad -(4)$

$2p-(n-p-2q)=3p+2q-n \qquad -2*(4)-(3)$

⇒ $3p+2q-n=p+(2p+2q)-n=p+n-n=p>0$

⇒ $2p>n-p-2q$

⇒ $p>(n-p-2q)/2$

That is, m is still the majority of the remaining numbers.

However, it holds only if the array has a majority element. Therefore, it is necessary to take O(n) time to go through the array to check whether the element we find is the majority array.

$Recursive\ cost \le T(n/2),\ and\ non-recursive\ cost = O(n)$

⇒ $T(n) \le T(n/2) + O(n)$

⇒ $T(n) = O(n)$ by Master Theorem

(2)

a. It take $O(n_1+n_2)$ time complexity to merge two sorted arrays with size n_1 and n_2 respectively. If merging the first two arrays with size n, the array with size 2n is created. Then merge in the third, the array with size 3n is created, and so on. Hence, the time complexity is

⇒ (n+n) + (2n+n) + (3n+n) + …… + ((k-2)n+n) + ((k-1)n+n)

= 2n + 3n + 4n + …… + (k-1)n + kn

= $\frac{(k+2)*(k-1)n}{2}$

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

= $O(k^2n)$

b. Recursively divide k-sorted arrays into two parts, each of them has k/2 arrays. If k = 2, then merge two arrays with size n into one array with size 2n. And then recursively merge two arrays with size 2n into one array with size 4n, and so on. In the last step, merge two kn/2 arrays into a single sorted array of size kn. In each level, there are $\frac{k}{2^i}$ - sorted arrays with size $2^i n$, and we need to merge two of them respectively to get $\frac{k}{2^{i+1}}$ - sorted arrays with size $2^{i+1}n$. Therefore, it takes

$\frac{k}{2^{i+1}} 2^i n \ = O(nk)$ time complexity in each level. Hence, the recurrence relation can be written as

$T(k) = 2T(\frac{k}{2}) + O(nk)$

$Solve\ T(k) = O(nk\ log\ k)\ by\ Master\ Theorem$

Problem 3

(1) Discuss with B03902048.

reference:

http://math.stackexchange.com/questions/296532/connect-n-white-and-n-black-points

Prove that the match with minimum total segment length is a good-word match:

Assume that the match M with minimum total segment length is not a good-word match, that is, it is a miserable-word match. Therefore, there must exist at least one intersection in M. Arbitrarily choose one of them, in which AB intersects CD. (A and D are white, B and C are black.) By the hint, we can find

$$AB + CD > AC + BD$$

Hence, M is not the match with minimum total segment length. It is a contraction to the premise. Therefore, the match with total segment length is a good-word match.

And then consider all N! matches to connect each black and white points. Since N! is a finite number, there always exists at least one match with minimum total segment length in N! matches.

Therefore, good-word match always exists since the match with minimum total segment length which is also a good-word match always exists.

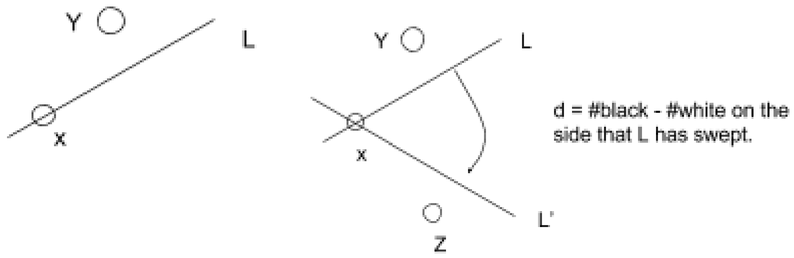
(2) Discuss with B03902048 and Instructor Hsiao.

First, choose the point with minimum x-coordinate as x. If there are two points with minumum x-coordinate, choose the one with smaller y-coordinate. Then connect x to other points p_i , and calculate the slope m_i of line xp_i. After calculating, p_i can be sorted according to m_i. And then, first draw the line L such that one side of L has only one point. Slowly rotate L such that L only sweep one point per rotation. (This can be done since we have the slope of each line segment.) **I claim that L satisfies the condiction in this problem can be found in the process mentioned above certainly.**

Proof:

Suppose not. If x is white, there are two cases.

Case 1: L splits the plane such that one side of L has only one black point before rotations. (Y is black) This is in contradiction to the premise since there are (N-1) black and (N-1) white points on the right plane of L.



Case 2: L splits the plane such that one side of L has only one white point before rotations. (Y is white) Consider that there is only point (Z) have not been swept by L. Since #black $\neq$ white when L rotates to L' and the initial d = -1, d will be smaller than 0 during the rotation. If z is black, $d_{L'}$ = (N-1)-(N-2) = 1 > 0. If z is white, $d_{L'}$ = N - (N-3) = 3 > 0. It's a contradiction too!

Similarly, when x is black, a contradiction also occurs.

Hence, the assumption is wrong. L satisfies the condiction in this promblem can be found during rotation.

Analysis:

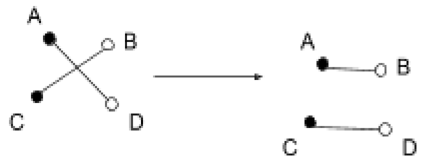
It takes O(2N) to find the point with minimum x-coordinate. Calculating the slope of xp_i also takes O(2N). Sorting can be done in O(2Nlog 2N). As for rotation, it takes O(2N) to rotate L.

Hence, the total complexity is

$$O(2N) + O(2N) + O(2Nlog2N) + O(2N) = O(NlogN)$$

(3) Discuss with B03902048

Use the result of (2) to divide 2N points. If one side of L has the number of black points equal to the number of white points, then the other side including x will also has the property #black = #white. Then dividing the points into two sides repeatedly until there are only one black and one white point. If there are only two points, then connect them and that is a good-word match. When combining two good-word matches to one match, the intersection won't happen. Consider the situation below.



It is a contradiction since A and B will been together when dividing. Same as other cases.

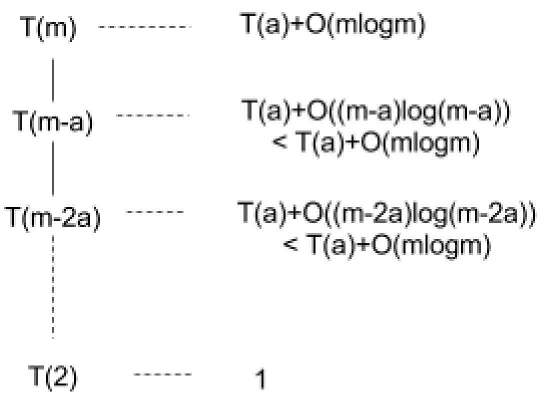
Hence, the recurrence relation can be expressed as

$T(2N) = T(2N - a) + T(a) + O(NlogN),\ 2N - 2 \geq a \geq 2$

$let\ 2N = m,\ then\ the\ relation\ can\ be\ rewritten\ as$

$T(m) = T(m - a) + T(a) + O(\frac{m}{2} log\ \frac{m}{2}) = T(m - a) + T(a) + O(mlog\ m)$

$Then\ the\ recursive\ tree\ can\ be\ drawn\ as\ below$



$$\Rightarrow\ T(m) = 1 + \sum_{i=0}^{\frac{m-3}{a}} (T(a) + O((m - ia)log(m - ia))) < 1 + \sum_{i=0}^{\frac{m-3}{a}} (T(a) + O(mlogm))$$

$$\Rightarrow\ T(m) < 1 + \frac{m}{a} T(a) + \frac{m}{a} O(mlogm)$$

$$\Rightarrow\ T(m) = 1 + O(m) + O(m^2 logm) = O(m^2 logm)\ since\ a\ is\ a\ constant.$$

$\Rightarrow$ *The time complexity is* $O(N^2 logN)$.