B03902062 資工二 董文捷

Problem 1

(1)

(a)

1.

Assume $f(n) = O(n^2) \leq c * n^2 - d * n$

Take c = 800, d = 200

$f(n) = O(n^2) \leq$ $\mathbf{800 * n^2 - 200 * n}$

2.

Boundary

$n = 1$ $f(1) = 514 < 800 * 1 - 200 * 1 = 600$

$n = 2$ $f(2) = 1028 < 800 * 4 - 200 * 2 = 2800$

$n = 3$ $f(3) = 1542 < 800 * 9 - 200 * 3 = 6600$

3.

If $f(n) = O(n^2) \leq$ $\mathbf{800 * n^2 - 200 * n}$ holds for m < n

take $m = \frac{n}{4}$

$f(n) = 16f(\frac{n}{4}) + 514n$

$\leq 16 * (800 * \frac{n^2}{16} - 200 * \frac{n}{4}) + 514 * n$

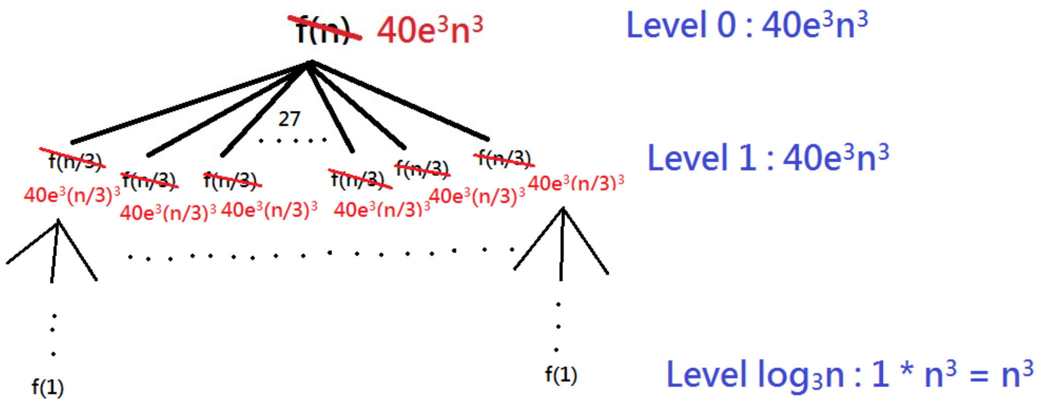
$= 800 * n^2 - 286 * n$

$\leq\ 800 * n^2 - 200 * n$

The equation also holds

By mathematical induction, f(n) $\leq$ **$800 * n^2 - 200 * n$** holds for all n, which indicates f(n) is $O(n^2)$

(b)



$f(n) = 40e^3n^3 * \log_3 n + n^3$ $\qquad O(n^3\log n)$

Proof :

1.

Assume $f(n) = O(n^3\log n) \leq c * n^3\log n$

Take c = 1000

$f(n) = O(n^3\log n) \leq$ **$1000 * n^3\log n$**

2.

Boundary

Take $n = 2, 3, 4, 5$ as boundary (Since $n = 1$ doesn't hold)

$n = 2$ $f(2) = 27 + 40e^3 * 2^3 \leq 1000 * 2^3 * \log 2$

$n = 3$ $f(3) = 27 + 40e^3 * 3^3 \leq 1000 * 3^3 * \log 3$

$n = 4$ $f(4) = 27 + 40e^3 * 4^3 \leq 1000 * 4^3 * \log 4$

$n = 5$ $f(5) = 27 + 40e^3 * 5^3 \leq 1000 * 5^3 * \log 5$

3.

If $f(n) = O(n^3 \log n) \leq$ **$1000 * n^3 \log n$** holds for $m < n$

take $m = \frac{n}{3}$

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

$\leqq 27 * 1000 * (\frac{n}{3})^3 * \log(\frac{n}{3}) + 40e^3 n^3$

$= 1000 * n^3 \log n - (1000 \log 3 - 40e^3) n^3$

$\leqq 1000 * n^3 \log n$

The equation also holds

By mathematical induction, $f(n) \leq$ **$1000 * n^3 \log n$** holds for all n, which indicates $f(n)$ is $O(n^3 \log n)$

(2)

References :

https://en.wikipedia.org/wiki/Stirling%27s_approximation

http://math2.org/math/expansion/power.htm

1. $e^5n^3 - 10n^2 + e^{1000}$

$\mathbf{n^3}$

2. f(n) = ef(n/2) $e^{log_2 n} = n^{log_2 e}$

$\boldsymbol{n^{log_2 e}}$

3. f(n) = f(n − 1) + n^e $1^e + 2^e + \ldots\ldots + n^e$

$\mathbf{n^{e+1}}$

4. f(n) = √nf(√n) + n

recursion tree, take m > 1 as base case $n\, log_m n$

n ln n

5. f(n) = f(n − 1) + $\frac{1}{n}$ Integral test ln n

ln n

6. f(n) = f(n − 1) + f(n − 2) Fibonacci $\frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n \right)$

$\boldsymbol{\left(\frac{1+\sqrt{5}}{2}\right)^n}$

7. en lg (ln n)

n ln (ln n)

8. $\sum_{i=1}^{n} \frac{1}{i}$ Integral test

ln n

9. $\frac{n}{\ln n}$

$\boldsymbol{\frac{n}{\ln n}}$

10. n lg n

n ln n

11. n log n

n ln n

12. n! Stirling's approximation $\sqrt{2\pi n}(\frac{n}{e})^n$

$\boldsymbol{\sqrt{n}(\frac{n}{e})^n}$

13. 2147483647

1

14. $n^{\frac{1}{\lg n}}$ take lg : lg y = lg n * $\frac{1}{\lg n}$ y = 10

1

15. $(\sqrt{2})^{\ln n}$

$\boldsymbol{(\sqrt{2})^{\ln n}}$

16. ln n! Stirling's approximation

n ln n

17. $e^{\ln n}$ $n^{\ln e} = n$

n

18. $\frac{10\,n}{e}$

n

19. lg ln n lg ln n = lg e * ln ln n

ln ln n

20. 2^{10000}

1

21. $(\lg n)^{\ln n}$

$\mathbf{n^{\ln \lg n}}$

22. $n^{3/2}$

$\mathbf{n^{3/2}}$

23. $n^{\lg \lg n}$

$\mathbf{n^{\lg \lg n}}$

24.$n^{\lg \ln n}$

$\mathbf{n^{\lg \ln n}}$

Equivalence class

C_1 1

13. 2147483647

14. $n^{\frac{1}{\lg n}}$

20. 2^{10000}

C_2 ln ln n

19. lg ln n

C_3 ln n

5. f(n) = f(n − 1) + $\frac{1}{n}$

8. $\sum_{i=1}^{n} \frac{1}{i}$

C_4 $\frac{n}{\ln n}$

9. $\frac{n}{\ln n}$

C_5 n

17. $e^{\ln n}$

18. $\frac{10\,n}{e}$

C_6 n ln (ln n)

7. en lg (ln n)

C_7 n ln n

4. f(n) = √nf(√n) + n

10. n lg n

11. n log n

16. $\ln n!$

C_8 $n^{log_2 e}$

$f(n) = ef(n/2)$

C_9 $n^{3/2}$

22. $n^{3/2}$

C_{10} n^3

1. $e^5n^3 - 10n^2 + e^{1000}$

C_{11} n^{e+1}

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

C_{12} $(\sqrt{2})^{\ln n}$

15. $(\sqrt{2})^{\ln n}$

C_{13} $(\frac{1+\sqrt{5}}{2})^n$

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

C_{14} $n^{\lg \lg n}$

23. $n^{\lg \lg n}$

C_{15} $n^{\lg \ln n}$

24.$n^{\lg \ln n}$

C_{16} $n^{\ln \lg n}$

21. $(\lg n)^{\ln n}$

C17 $\sqrt{n}\left(\frac{n}{e}\right)^{n}$

12. n!

Problem 2

(1)

References :

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

(a)

Split the array A into two arrays A1 and A2 of half size, we can find the majority element of A by checking if majority_A_1 or majority_A_2 is the majority element of A if A_1 or A_2 has majority element. If both of them aren't, A has no majority element.

Proof :

If x is not a majority element in A_1 or A_2, then $x_{A1} \leqq \lfloor n_{A1}/2 \rfloor$

$x_{A2} \leqq \lfloor n_{A2}/2 \rfloor \quad x_A = x_{A1} + x_{A2} = \lfloor n_{A1}/2 \rfloor + \lfloor n_{A2}/2 \rfloor \leqq \lfloor n_A/2 \rfloor$

is not a majority element of A

Time complexity :

T(n) = 2 * T(n / 2) + 2 * n

T(n) = n log n

My code in C++ :

```
bool check_majority(int left, int right, int *A, int candidate)
{
    if(candidate == -1) // -1 represent NONE
        return 0;
    int i, num = 0, n = (right - left + 1);
    for(i = left; i <= right; i++)
        if(A[i] == candidate)
            num++;
    if(num > (n / 2))
        return 1;
    return 0;
}
int find_majority(int left, int right, int *A)
{
    if(left >= right)
        return A[left];
    else
    {
        int mid = (left + right) / 2;
        int majority_A1 = find_majority(left, mid, A);
        int majority_A2 = find_majority(mid + 1, right, A);
        if(check_majority(left, right, A, majority_A1))
            return majority_A1;
        else if(check_majority(left, right, A, majority_A2))
            return majority_A2;
        else
            return -1; // no majority element
    }
}
```

(b)

Pair up the elements in A arbitrarily to get n / 2 pairs. In each pair, if two elements are different, we discard both of them. Otherwise, we keep one of them. A variable tie-breaker is kept and updated with the unpaired element each time when n is odd. Repeat the process until there are only two elements left, if one of them equals to the tiebreaker or they are the same element, return the element. That element is the candidate of majority element, check the whole array to see if it is really the majority element. Otherwise, return NONE.

If there is a majority element m, it will still be a majority element after one process. Suppose d is the difference of number of m and other elements. If m is a majority element, d > 0, and d has to > 0 after the process.

Pair of different element

Both ! = m   d = d + 2

One of them = m   d = d

So d will not decrease due to the discard. If d > 0 for the pairs with same elements, we can combine them with d / 2 still > 0

Therefore m is still a majority element after one process.

(2)

(a)

Merging two arrays of size a and b takes O(a + b) time

If we merge arrays one by one, we will take 2n + 3n + 4n

+ …… + kn (First time n + n, second time 2n + n ……)

$= \left(\frac{k^2+k}{2} - 1 \right) n$

$O(k^2n)$

(b)

We can divide t arrays into two sides with each side having $\frac{t}{2}$ or $\frac{t}{2}$ + 1 arrays. Merge each side respectively. After each side finishing merging, we can merge the two merged arrays of size $(\frac{t}{2})$ n or $(\frac{t}{2} + 1)$ n together, taking O(n t) times, and return the final array.

Time complexity :

$T(t) = 2 * T(\frac{t}{2}) + O(n\ t)$

T(t) = O(n t log t)

T(k) = **O(n k log k)**

Problem 3

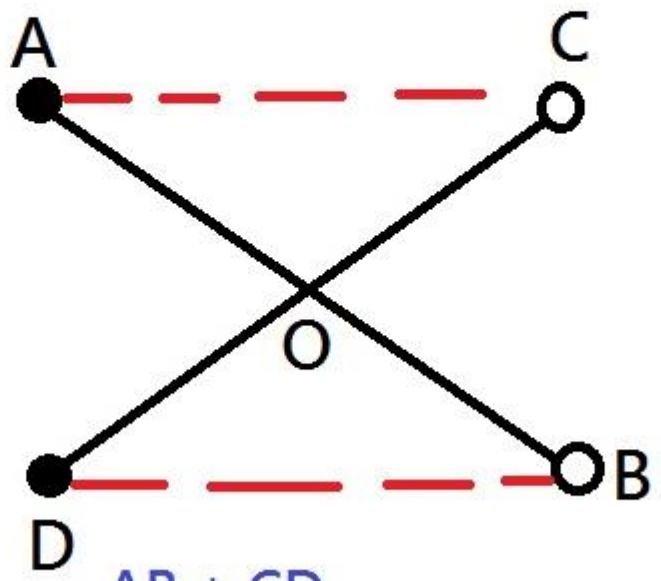
References : Discussed with JoJorge (b03902061)

(1) The match with minimum total segment length must be a good-word match.

Proof :

(Proof by contradiction)

Assume the match with minimum total segment length is M_{min}. If M_{min} is not a good-word match, we can find two intersected line segments AB and CD. Replace pair AB and CD with pair AC and BD, forming a new match M'. M' has a smaller total segment length than M_{min} since $AB + CD > AC + BD$, which is a contradiction to the assumption that M_{min} has a minimum total segment length. So we can conclude that M_{min} must be a good-word match.



$AB + CD$

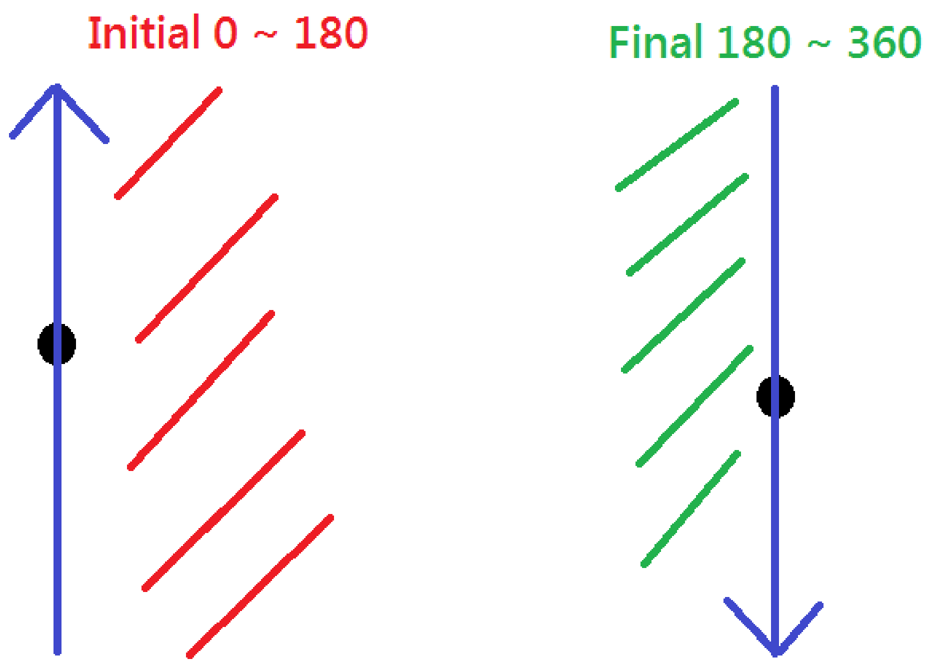
$= AO + OB + CO + OD$

$= AO + OC + DO + OB$

$> AC + BD$

(2) Choose an arbitrary center point x. Sort all points by their angles relative to x (+y for 0 degree and +x for 90 degree). Keep one variable to store now angle rotating from 0 degree and two variables for now position of L_first and L_last in the array. The numbers between L_first and L_last are points in the chosen side. Since there are no two points coincide at the same location and no three points on the same line, we can choose now_angle carefully to include one more point or exclude one point each time (L_first ++ or L_last ++). Assume for the initial situation, d = black – white != 0, and the final situation's d also != 0.(If either of them = 0 we can simply choose it as answer). Since initial_d + final_d = +1 or -1 (The total difference in whole plane), and initial_d != 0 fianl_d != 0, there must be a negative integer and a positive integer in initial_d and final_d. During the rotation, d will either +1 or -1 (continuous function). By Intermediate Value Theorem, during the change of d between positive and negative, there must be one step having d = 0. If there is a d = 0 with one side empty, we can rotate L between the non-empty side and get another d = 0 with neither

side empty.



Time Complexity :

Sort : O(N log N)

Find L : O(N)

Total : O(N log N)

(3)We can use the result of (2) to divide and conquer, each time divide points into two sides(x is included in the side with unequal black and white), each side can find their own good-word match (By (1), the good-word match must exist)

Time complexity :

Each time will divide at least one pair from original group, at most need N times to finish, and each time takes $O(n_{element} \log n_{element})$ to divide, which is smaller than $O(N\log N)$

So $T(N) \leqq N * O(N\log N) = O(N^2\log N)$