Second Exam

(1) n^2 , $10n$

$n^2 \geq 10n$ IF $n = 10$

1	1	10
2	4	20
3	9	30
4	16	40
5	25	50
6	36	60
7	49	70
8	64	80
9	81	90
10	100	100
11	121	110

2^n , $2n^3$

$2^n \geq 2n^3$ IF $n = 12$

	$2n^3$	2^n
1	2	2
2	16	4
3	54	8
4	128	16
5	250	32
6	432	64
7	686	128
8	1024	256
9	1458	512
10	2000	1024
11	2662	2048
12	3456	4096

(2) $T(n) = 4T(n/2) + n$

$= 4(4T(n/2^2) + n/2) + n$

$= 4.4T(n/2^2) + 2n + n$

$= 4.4.(4T(n/2^3) + n/2^2) + 2n + n$

$= 4^3 T(n/2^3) + 4n + 2n + n$

$= 4^3(4T(n/2^4) + n/2^3) + 4n + 2n + n$

$= 4^4 T(n/2^4) + 8n + 4n + 2n + n$

repeat until $T(1)$, assume you repeat until K times

$T(n) = 4^K T(n/2^K) + n\sum_{i=0}^{K} 2^i$

$\because n/2^K = 1 \quad \therefore K = \log_2 n$

$\therefore T(n) = 2^{2K} T(n/2^K) + n\sum_{i=0}^{K} 2^i$

$\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$

$\therefore T(n) = 2^{2\log_2 n} * 1 + n(2^{K+1} - 1)$

$= n^2 + n(2^K * 2 - 1) = n^2 + n(2^{\log_2 n} * 2 - 1)$

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

$T(n) = 3n^2 - n$

$\therefore T(n) = O(n^2)$ #

③ O-notation ⇒ ✱ It represents the upper bounds of the number of an algorithm

✱ Big O Notation's role is to calculate the longest time an algorithm can take for its execution

✱ It is used for calculating the worst-case time complexity of an algorithm

what is the meaning of $n^2 + O(n) = O(n^2)$

$\because n^2 = O(n^2)$, $O(n) = O(n)$

$\therefore n^2 + O(n) = max(O(n^2), O(n))$

$= O(n^2)$

④ Solved before

⑤ $T(n) = a_m n^m + a_{m-1} n^{m-1} + \dots + a_1 n^1 + a_0 n^0$

$T(n) \le |a_m n^m| + |a_{m-1} n^{m-1}| + \dots + |a_1 n^1| + |a_0 n^0|$

$= n^m \cdot \left(|a_m| + \left|\frac{a_{m-1}}{n}\right| + \dots + \left|\frac{a_1}{n^{m-1}}\right| + \left|\frac{a_0}{n^m}\right|\right)$

$\le n^m \cdot (|a_m| + |a_{m-1}| + \dots + |a_1| + |a_0|) = c \cdot n^m$

when selecting $c = |a_m| + |a_{m-1}| + \dots + |a_1| + |a_0|$

$\Rightarrow T(n) = o(n^m)$

⑥ Solved before

⑦ ALGORITHM Fib(n)

// Input: Enter n you want to get Fibonacci for it

// OUTPUT: result of n Fibonacci $\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$

F ← (1 1 / 1 0) // Define matrix 2*2

If n = 0 do

return 0

Power (F, n-1)

return F[0][0] // return F_{n+1} which is index F[0][0]

ALGORITHM Power (F, n)

// Input: Matrix and exponential

// output: the result of calculation F^n

If n = 0 or n = 1 THEN return

M ← (1 1 / 1 0)

Power (F, ⌊n/2⌋)

Multiply (F, F)

If n % 2 ≠ 0 do

Multiply (F, M)

ALGORITHM Multiply(F, M)
// Input: two matrix
// output: Result of multiplication of two matrix
x ← (F[0][0] * M[0][0] + F[0][1] * M[1][0]);
y ← (F[0][0] * M[0][1] + F[0][1] * M[1][1]);
z ← (F[1][0] * M[0][0] + F[1][1] * M[1][0]);
w ← (F[1][0] * M[0][1] + F[1][1] * M[1][1]);
F[0][0] ← x; F[0][1] ← y; F[1][0] ← z; F[1][1] ← w

We can compute nth power of M = (1 1 / 1 0) using recursive squaring. That is if n is even, we recursively compute $A^{n/2}$. This sub-problem takes T(n/2) time. Cach this result in temp variable. The desired result is

⑧ Solved before

⑨ ALGORITHM lcs(X, Y, m, n)
// Input: two sequences
// OUTPUT: calculate the length of the longest match
if m = 0 or n = 0 do
return 0;
else if X[m-1] = Y[n-1] do
return 1 + lcs(X, Y, m-1, n-1);
else
return max(lcs(X, Y, m, n-1), lcs(X, Y, m-1, n));

Recursion

$A^n = A^{n/2} \times A^{n/2}$. Since we have already compute $A^{n/2}$, it's the same subproblem, we don't have to repeat this work. So, all we do is temp * temp. Multiplying two 2x2 matrices takes constant time - O(1). So, the total running time is T(n) = T(n/2) + O(1) which upper bound O(lg n)

ALGORITHM LCS(X, Y, m, n)
// input: two subsequence
// output: longest common subsequence
IF (X[m] = Y[n]) do
    LCS(X, Y, m, n) = 1 + LCS(X, Y, m-1, n-1)
else
    LCS(X, Y, m, n) = max {LCS(X, Y, m-1, n), LCS(X, Y, m, n-1)}

Dynamic programming

X	A	B	C	B	D	A	B
	1	2	3	4	5	6	7

Y	B	D	C	A	B	A
	1	2	3	4	5	6

		A 1	B 2	C 3	B 4	D 5	A 6	B 7
	0	0	0	0	0	0	0	0
B	1	0	0	①	1	1	1	1	1
D	2	0	0	1	1	1	2	2	2
C	3	0	0	1	②	2	2	2	2
A	4	0	1	1	2	2	2	3	3
B	5	0	1	2	2	③←3	3	4	
A	6	0	1	2	2	3	3	④←4	

The length is 4
BCBA

LCS: subproblem
L(i, J) = LCS (X[i:], Y[J:])
: (for 0 ≤ i ≤ |X|, 0 ≤ J ≤ |Y|)
Relate: L(i, J) = IF X[i] = Y[J]:
    1 + L(i+1, J+1)
else:
    max {L(i+1, J), L(i, J+1)}
⩾ one of X[i] & Y[J] ∉ LCS
Topo: For i = 0, ..., |X|: for j = 0, ..., |Y|
Base: L(|X|, j) = 0 = L(i, |Y|)
original: L(0, 0)
Time: θ(|X|.|Y|)

THIRD Exam

① @ - outer loop iterations = n times
- First inner loop: it's dependent on the outer loop for one of its factors, the number of iterations in the body of the inner loop is $1+2+3+\dots+10 = 55$, the average of this loop is 5.5 (55/10), this can be written as $(N+1)/2$
- second inner loop: it's dependent on the first inner loop for one of its factor, the number of iterations in the body of the inner loop is $1+3+6+10+\dots+55 = 220$, the average of this loop is 22 (220/10), this can be written as $(N+1)*2$

iterations = # loop iterations * # inner loop iterations

$T(N) = N * (N+1)/2 * (N+1)*2$

$= (N^2+N) * (N+1)$

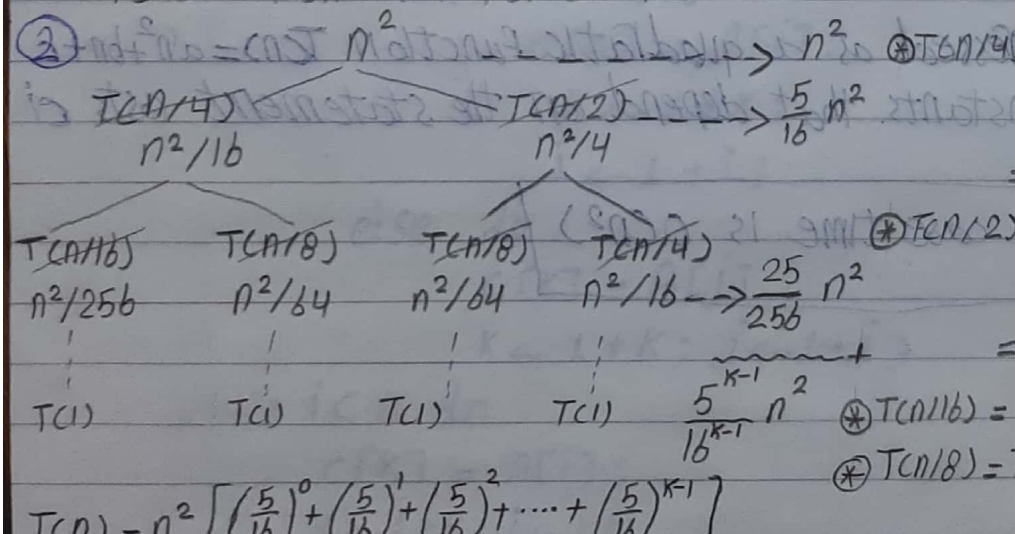
$= N^3 + N^2 + N^2 + N$

$= N^3 + N^2 + 2N$

⑥ $T(N)$

$T(n) = T(n/4) + T(n/2) + n^2$

②



$T(n) = n^2\left[\left(\frac{5}{16}\right)^0 + \left(\frac{5}{16}\right)^1 + \left(\frac{5}{16}\right)^2 + \dots + \left(\frac{5}{16}\right)^{k-1}\right]$

assume that $r = \frac{5}{16} \longrightarrow n^2 \cdot \frac{a_1}{1-r} = \frac{1}{1-\frac{5}{16}} \cdot n^2 = \frac{16}{11}n^2 = T(n)$

$\therefore T(n) = \Theta(n^2)$

$T(n) = T(n/4) + T(n/2) + n^2$

$T(n/4) = T(n/4^2) + T(n/2.4) + (n/4)^2$

$= T(n/16) + T(n/8) + \frac{n^2}{16}$

$T(n/2) = T(n/2.4) + T(n/2.2) + (n/2)^2$

$= T(n/8) + T(n/4) + \frac{n^2}{4}$

$T(n/16) = T(n/64) + T(n/32) + \frac{n^2}{256}$

$T(n/8) = T(n/32) + T(n/16) + \frac{n^2}{64}$

③ solved before

④ Insertion sort (A, n)

for $j \leftarrow 2$ to n do	c_1	n
key $\leftarrow A[j]$	c_2	$n-1$
$i \leftarrow j-1$	c_4	$n-1$
while $i>0$ and $A[i] > key$ do	c_5	$\sum_{j=2}^{n} t_j$
$A[i+1] \leftarrow A[i]$	c_6	$\sum_{j=2}^{n} t_j - 1$
$i \leftarrow i-1$	c_7	$\sum_{j=2}^{n} t_j - 1$
$A[i+1] \leftarrow key$	c_8	$n-1$

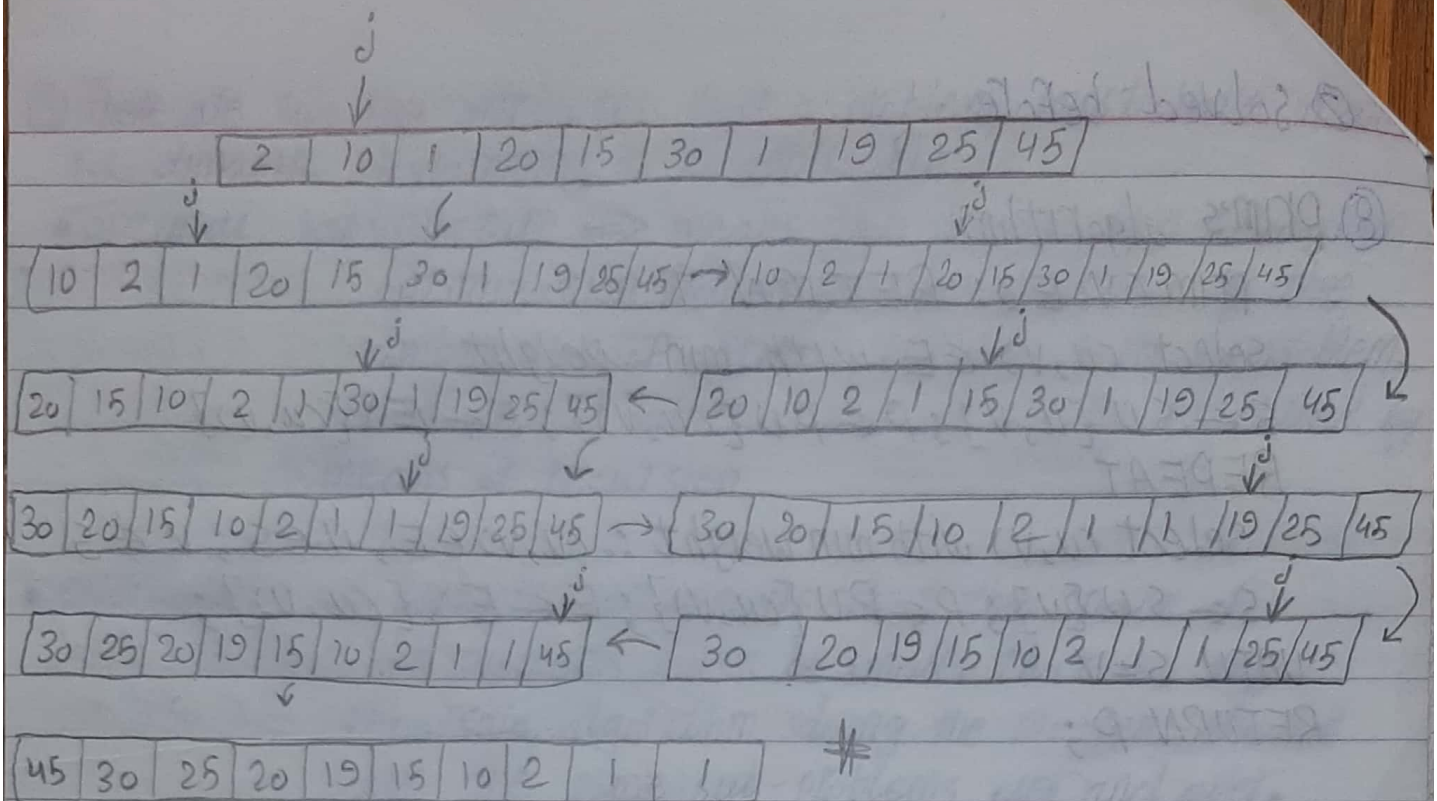
Worst case: The worst case of insertion sort occurs if the array is the reverse order

$$T(n) = c_1(n) + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1)$$

$$= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)n - (c_2 + c_4 + c_5 + c_8)$$

T(n) can be expressed as a quadratic function $T(n) = an^2 + bn + c$ where a, b, c constants that depend on the statement costs ci

So, the worst case time is $O(n^2)$



⑤ Solved before

⑥ Merge sorted array (A, B, C, n, m)

```
i ← 0, j ← 0, k ← 0;
while i<n and j<m do
    IF A[i] < B[j] do
        C[k] = A[i];
        k ← 1+k;
        i ← 1+i;
    else do
        C[k] = B[j]
        k ← 1+k; j ← 1+j;
while i<n do
    C[k] = A[i];
    k ← 1+k; i ← 1+i
while j<m do
    C[k] = B[j]; k ← 1+k; j ← j+1;
```

⑦ Solved before

⑧ prim's algorithm

```
prim(V, E): RETURN T
select (u,v) ∈ E with min weight
S ← S ∪ {u, v}; P ← P ∪ {(u,v)}; E ← E \ {(u,v)}
REPEAT
  select (u,v) with min weight :. (u,v) ∈ E, u ∈ S, v ∉ S
  S ← S ∪ {v}; P ← P ∪ {(u,v)}; E ← E \ {(u,v)};
UNTIL S = V
RETURN P;
```

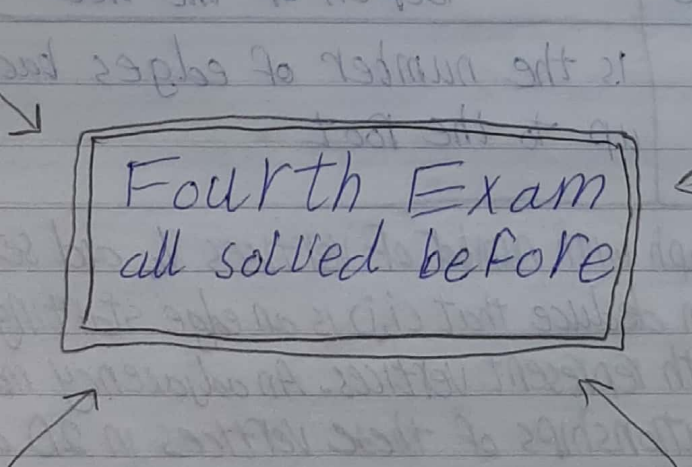
⑨ There are two key attributes that a problem must have in order for dynamic programming to be applicable:

- optimal substructure ⇒ means that the solution to a given optimization problem can be obtained by the combination of optimal solutions to its sub-problem. Such optimal substructures are usually described by means of recursion

- overlapping sub-problems ⇒ means that the space of sub-problems must be small, that is, any recursive algorithm solving the problem should solve the same sub-problems over and over, rather than generating new sub-problems

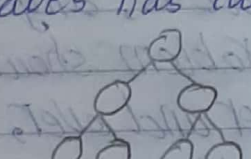
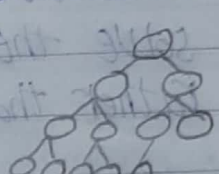
If a problem can be solved by combining optimal solutions to non-overlapping sub-problems, the strategy is called "divide and conquer" instead. This is why merge sort and quick sort aren't classified as dynamic programming problems.

⑩ solved before



Fifth Exam

① @ optimization problem ⇒ is the problem of finding the best solution from all feasible solutions.
It can be divided into two categories:
- An optimization with discrete variables is known
- A problem with continuous variables is known

Full binary tree	Complete binary tree
A full binary tree is a tree in which every node other than the leaves has two children 	A complete binary tree is a binary tree in which every level, except possibly the last, is complete filled, and all nodes are as far left as possible 

spanning tree ⇒ is a subset of Graph G, which has all the vertices covered with minimum possible number of edges. A spanning tree doesn't have cycles and cannot be disconnected.

Height of the tree	Depth of the tree
is the number of edges to its most distant leaf node	is the number of edges back up to the root

Adjacency matrix ⇒ The graph with number of vertices V_G and set number of edges E_G. From this we can deduce that c(i,j) is an edge starting at i and ending at j, if i and j both represent vertices. An adjacency matrix is a way of representing the relationships of these vertices in 2D array.
⇒ For unweighted graphs if there is connection between vertex i and j, (b) solved before then the value of the cell [i,j] will equal 1, if there isn't connection, it will equal 0
⇒ For weighted graph we will add value rather than 1 or 0

Adjacency matrix => of $G = (V_G, E_G)$ is the $n \times n$ matrix A indexed by V, whose (u,v)-entry is defined as

$$A_{uv} = \begin{cases} 1 & \text{if } uv \in E, \text{ undefined} \\ 0 & \text{if } uv \notin E \end{cases}$$

② Binary tree => is a tree data structure composed of nodes, each of which has at most, two children, referred to as left and right nodes. The tree starts off with a single node known as the root.

ⓐ solved before

ⓑ Base case: For $H=0$. A tree of height 0 has exactly one node and it's in level one "$i=1$"

$2^{i-1} = 2^{1-1} = 1$ maximum number of node

Induction hypothesis: For tree of height H has $H+1$ number of level "$i = H+1$"

③ MergeSort(A[i,j])

```
IF i<j then
    k:= (i+j)/2;                    O(1)
    MergeSort(A[i,k]);              T(N/2)
    MergeSort(A[k+1,j]);            T(N/2)
    merge(A[i,j],k);                C.N
```

```
Merge(A[i,j],k)
    l←i; m←k+1; t←i;
    while(l≤k) or (m≤j)
        IF l>k then                 B[t]←A[m]; m←m+1;
        else IF m>j then            B[t]←A[l]; l←l+1;
        else IF A[l]<A[m] then      B[t]←A[m]; m←m+1;
        else                        B[t]←A[l]; l←l+1;
        t←t+1;
    for t←i to j do A[t]←B[t]
```

$T_{merge}(n) = o(7n+3) = o(n)$

$T(n) = 2T(n/2) + 1 + T_{merge} = 2T(n/2) + 7n + 4$

$= 2(2T(n/2^2) + 7n/2 + 4) + 7n + 4$

$= 2.2\,T(n/2^2) + 7n + 7n + 4 + 4.2$

$= 2.2(2T(n/2^3) + 7n/2^2 + 4) + 7n + 7n + 4 + 4.2$

$= 2^3 T(n/2^3) + 3.7n + 4(1 + 2 + 2^2)$

$= 2^{\log_2 n} T(n/2^{\log_2 n}) + \log_2 n . 7n + 4\sum_{i=0}^{\log n} 2^i$

$= n + \log_2 n . 7n + 8n - 4$

$= \log_2 n . 7n + 9n - 4$

$4\sum_{i=0}^{\log n} 2^i \rightarrow 4(2^{\log_2 n + 1} - 1)$

$4(2^{\log_2 n} . 2 - 1)$

$4(n.2 - 1)$

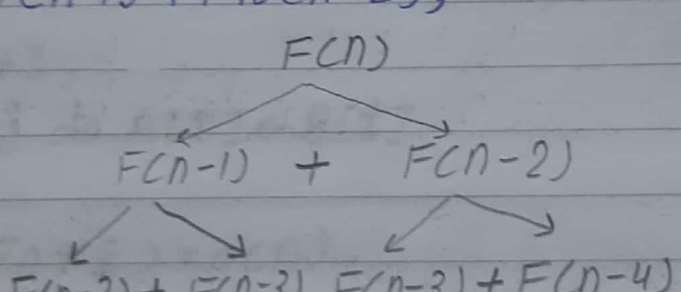
$(8n - 4)$ #

$T(n) = o(n \log n)$

SIXth Exam

1) The Fibonacci numbers ⇒ are defined as the sequence beginning with 0 and two 1's, and where each succeeding number in the sequence is the sum of the two preceeding numbers.

ALGORITHM Fib(n)
//input: nth Fibonacci number
//output: the result of adding preceeding numbers
```
if (n≤1)
    return 1;
else
    return Fib(n-1) + Fib(n-2);
```



④ ALGORITHM search(A, n, x):
// input: number to search for in the array
// output: the position of number in array if it exist
// else return 0
For j ← 1 to n do
    IF x = A[j] do
        return j;
return 0;

⑥ optimization problems:
is the problem of finding the best solution from all feasible solutions.
It can be divided into two categories:
⊕ An optimization problem with discrete variables is known
⊗ An problem with continous variables is known

Show greedy technique
prim's algorithm

```
prim(V,E) RETURN T;
select (u,v) ∈ E with min weight
S ← SU {u,v}; P ← PU {(u,v)}; E ← E \ {(u,v)};
REPEAT
  select (u,v) with min weight : (u,v) ∈ E, u ∈ S, v ∉ S
  S ← SU{v}; P ← PU{(u,v)}; E ← E \ {(u,v)};
UNTIL S=V
```