1. Input : Unsorted array of n distinct number
$A[a_1, a_2, \ldots a_n]$
Output : Index i, where a_i is the smallest numbar
in $A[a_1, a_2, \ldots a_n]$

Divide and conquer algorithm

Divide: Divide the n-element array into two array $A[a_1, a_2 \ldots a_{n/2}]$ and $A[a_{\lfloor n/2+1 \rfloor}, \ldots a_n]$ where each of them has size $n/2$. Now we have two smaller sub problem from the original one.

Conquer: Solve the sub problems recursively. The base case will be the subproblem with size 1 and index of that element will be returned

Combine ; from the two subproblem solved individually, we get the index s_1 & s_2 of the smallest element of both. We then compare $A[s_1]$ and $A[s_2]$ and return the index of the smallest one.

Pseudo code

```
GetMinIndex (A, i, j)
    if (i >= j) return i;
    else   x = GetMinIndex (A, i, (i+j)/2)
           y = GetMinIndex (A, (i+j)/2 + 1, j)
           if (A[x] < A[y]) return x;
           else             return y;
```

Recurrence Relation

$$T(n) = \begin{cases} 1, & \text{if } n = 1 \\ 2T(n/2) + 1, & \text{otherwise} \end{cases}$$

Proof by induction

Guess algorithm takes $\theta(n)$ time

At first to prove $T(n) = O(n)$

$T(n) \leq c(n-1)$ for some constant $c > 0$
and $n \geq n_0$ for some $n_0 > 0$

Assume $T(k) \leq c(k-1)$ for $k < n$

$$\begin{aligned} \text{Now } T(n) &= 2T(n/2) + 1 \\ &\leq 2 \cdot c\left(\frac{n}{2} - 1\right) + 1 \\ &= cn - 2c + 1 \\ &= c(n-1) - c + 1 \\ &= c(n-1) + (1-c) \\ &\leq c(n-1) \quad \text{for } 1 - c < 0 \Rightarrow c > 1 \end{aligned}$$

$\therefore T(n) = O(n)$ ——— (1)

Now to prove $T(n) = \Omega(n)$

$T(n) \geq cn$ for some constant $c > 0$
and $n \geq n_0$ for some $n_0 > 0$

Assume $T(k) \geq ck$ for $k < n$

$$\begin{aligned} \text{Now } T(n) &= 2T(n/2) + 1 \\ &\geq 2 \cdot c \cdot \frac{n}{2} + 1 \\ &= cn + 1 \\ &\geq cn \end{aligned}$$

$\therefore T(n) = \Omega(n)$ ——— (2)

from (1) & (2) $T(n) = \theta(n)$

2. (a)

$T(n) = O(n^2 \log_5 n)$

<u>proof by induction</u> : $T(n) \leq c n^2 \log_5 n$

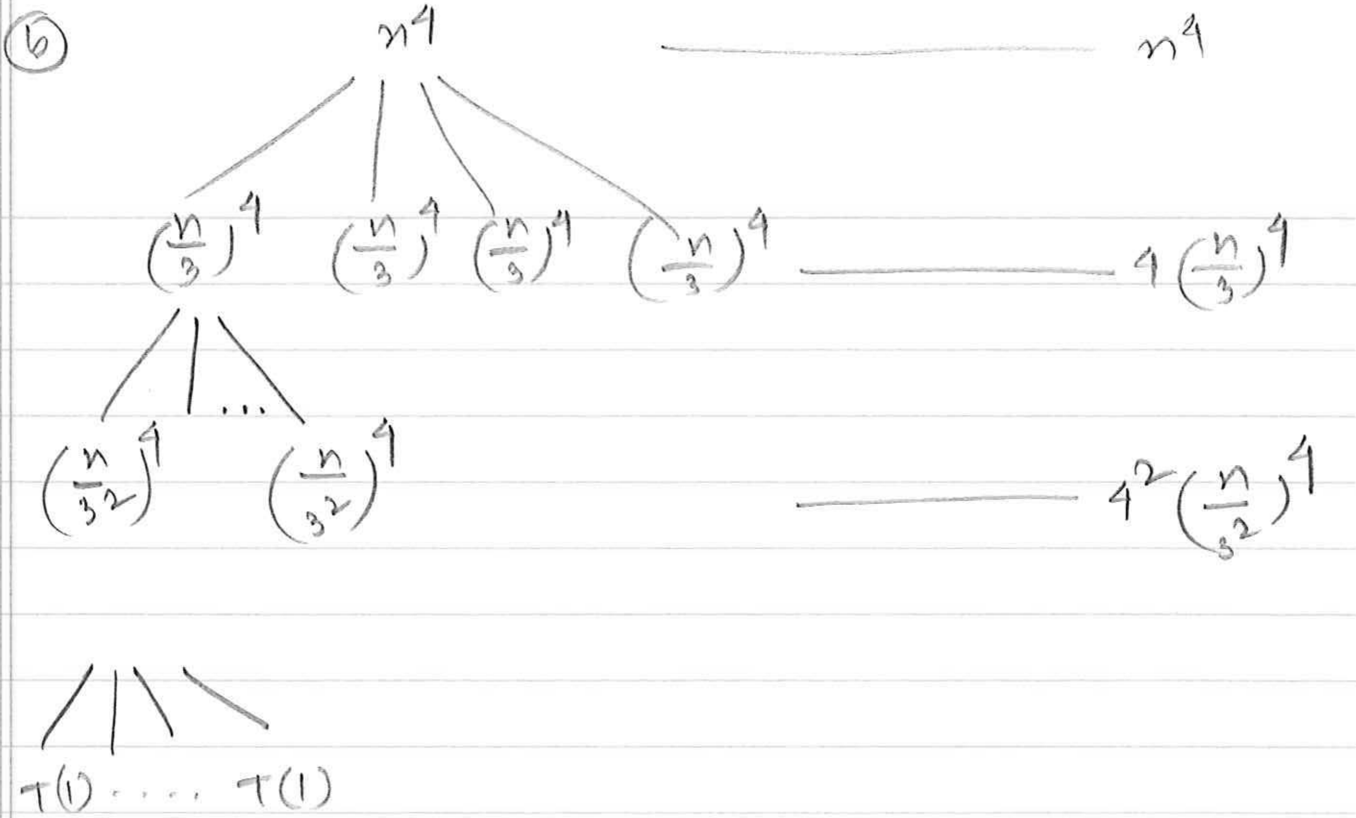
Assume $T(k) \leq c k^2 \log_5 k$ for $k < n$

Now, $T(n) = 25\, T(n/5) + n^2$

$$\begin{aligned}
&\leq 25 \cdot c\left(\frac{n}{5}\right)^2 \log_5\left(\frac{n}{5}\right) + n^2 \\
&= cn^2(\log_5 n - \log_5 5) + n^2 \\
&= cn^2 \log_5 n - cn^2 + n^2 \\
&= cn^2 \log_5 n - (cn^2 - n^2) \\
&\leq cn^2 \log_5 n \quad \text{when } cn^2 - n^2 > 0 \\
&\qquad \Rightarrow c > 1
\end{aligned}$$

$\therefore T(n) = O(n^2 \log_5 n)$

(b)



height of the tree $= \log_3 n$

so running time $= \sum_{i=0}^{\log_3 n} 4^i \left(\frac{n}{3^i}\right)^4$

$= n^4 \sum_{i=1}^{\log_3 n} \left(\frac{4}{3^4}\right)^i$

$= n^4 \sum_{i=1}^{\infty} \left(\frac{4}{3^4}\right)^i$

$= n^4 \cdot \frac{1}{1 - \frac{4}{3^4}}$

$= O(n^4)$

proof by induction

$T(n) \le cn^4$ for some constan c

and $n \geqslant n_0$, $n_0 > 0$

Assume $T(k) \leqslant ck^4$ for all $k < n$

so $T(n) \leqslant 4c\left(\frac{n}{3}\right)^4 + n^4$

$$= \frac{4cn^4}{81} + n^4$$

$$= cn^4 + \left(n^4 + \frac{4cn^4}{81} - cn^4\right)$$

$$= cn^4 + \left(n^4 + \frac{4cn^4 - 81cn^4}{81}\right)$$

$$= cn^4 + \left(n^4 - \frac{77cn^4}{81}\right)$$

$$= cn^4 + n^4\left(1 - \frac{77}{81}c\right)$$

$$\leqslant cn^4 \quad \text{if} \quad n^4\left(1 - \frac{77}{81}c\right) < 0$$

$$\Rightarrow 1 - \frac{77}{81}c < 0$$

$$\Rightarrow 1 < \frac{77}{81}c$$

$$\Rightarrow c > \frac{81}{77}$$

$\therefore T(n) = O(n^4)$

(4) We want to find the number of subproblem x such that running time

$$T(n) = x\,T\left(\frac{n}{3}\right) + O(\log n)$$

Here $a = x$, $b = 3$, $f(n) = \log n$

As $T(n) = O(n^2)$, $f(n) = \log n$ is not going to dominate $T(n)$. Its bound is going to be determined by $O(n^2)$.

So $n^{\log_b a} = O(n^2)$ (master method applied)

$\therefore n^{\log_3 x} = O(n^2)$

$\Rightarrow \log_3 x < 2$

$\Rightarrow x < 9$

So maximum 8 subproblems of size $n/3$ can be taken.