Basic Algorithms - see C.

Divide and Conquer.

Two steps:

1) Divide: break the problem into parts, which can be solved independently.

2) Conquer: put together solutions from each part to get back full solution.

Merge Sort.

Given [a₁, ⋯, an] need to sort them in increasing order.

Break into two lists:

[a₁, ⋯, an] = [a₁, ⋯, an/2], [an/2+1, ⋯, an]

Sort them recursively

→ [b₁, ⋯, bn], [bn+1, ⋯, bn+2]

2) merge them to get

[c₁, ⋯, cn].
How do you merge two sorted arrays.

- We can do it in $O(n)$.
- Keep comparing first elements.
- Remove the smaller and continue.

```
      b1
     ├──
     │   └── bw2
     └──
         └── bn
```

Complexity of merge = $O(n)$
as each element gets removed only once.

How to analyze time complexity.

\[
T(n) = 2T(n/2) + O(n).
\]

\[
\begin{array}{c}
\text{At root} \quad \frac{n}{2} \quad - \quad O(n) \\
\end{array}
\]

\[
\text{At level} 2 \quad \frac{n}{4} \quad - \quad 2 \cdot O(n/2).
\]

\[
\text{At level} 3 \quad \frac{n}{8} \quad - \quad 4 \cdot O(n/4).
\]

\[
\text{At level} 4 \quad \frac{n}{16} \quad - \quad 8 \cdot O(n/8).
\]

\[
\text{At level} k \quad \frac{n}{2^k} \quad - \quad 2^k \cdot O(n/2^k).
\]

\[
\text{Total} \quad \sum_{i=1}^{k} 2^i \cdot O(n/2^i) \quad = \quad O(n).
\]
if in general $n=2^k$

at each step you do a total of:

- # nodes at height $h$ x work at each node depth.

$$= 2^h \times O\left(\frac{n}{2^h}\right)$$

$$= O(n)$$.

& maximum depth = $\log n$

$\Rightarrow$ total work = height x work at each level

$$= O(n \log n)$$.

---

General recursion tree analysis.

say we break the given problem into "a"

- problems of size = $b$ by each.

& take $O(n^e)$ time to reconstruct the solution

$$\Rightarrow T(n) = aT\left(\frac{n}{b}\right) + O(n^e)$$.

$$\|$$

$C \cdot n^e$
So height = \[ \lceil \log_b n \rceil = h. \]

-> $n$ at depth $k$ \# nodes = $a^k$

work done per node = $(\frac{n}{b^k})^e$.

$\Rightarrow$ work done per level

$= a^k \cdot c \cdot (\frac{n}{b^k})^e$

$= c \cdot n^e \cdot (\frac{a}{b^e})^k$

$\Rightarrow$ total work

$= \sum_{k=0}^{h} c \cdot n^e \cdot (\frac{a}{b^e})^k = c \cdot n^e \cdot \frac{1}{1 - (\frac{a}{b^e})}$

case analysis

$s = \frac{a}{b^e}$. 
1) $g \leq 1$

$\Rightarrow \quad \sum_{k=1}^{\infty} s_k \leq \sum_{k=1}^{\infty} \frac{a}{s_k} = \frac{1}{1-g}$.

$\Rightarrow \quad T(n) \leq \frac{c.n^e}{1-g} = O(n^e)$.

2) $g = 1$

$\Rightarrow \quad T(n) = c.n^e.h = O(n^e \log n)$.

3) $g > 1$

$\Rightarrow \quad \sum_{k=1}^{\infty} s_k = \frac{\frac{h+1}{h-1}}{s^* - s} \leq s^{h+1}$

$\Rightarrow \quad T(n) \leq c.n^e \cdot s^{h+1}$

$\leq c.(\frac{n}{b^{h+1}})^{h+1} \cdot a.b^{h+1}$

$\leq c.a^h \log b.n$

$= c.a^{\log a} \log b.n$

$= c.n^{\log b.a}$

$= O(n^{\log b})$.  

as $n = \log b$
\[
\Rightarrow \\
1) \ e > \log_b a \quad \rightarrow \quad O(n^e) \\
2) \ e = \log_b a \quad \rightarrow \quad O(n \log n) \\
3) \ e < \log_b a \quad \rightarrow \quad O(n \log^a b) \\
\]

"Master Theorem"

For merge sort: \[ T(n) = 2T(m/2) + O(n) \]
\[ \Rightarrow T(n) = O(n \log n) \]

---

**Counting Inversions.**

Given a list \( a_1, \ldots, a_n \), an inversion is \( i < j \), \( a_i > a_j \).

How to count total number of inversions?

**Idea:**
1) Break into two lists.
2) Calculate inversions inside.
3) Merge & count inversions while merging.
\[ a_1, \ldots, a_n \]

\[ a_1, \ldots, a_{n/2}, a_{n/2+1}, \ldots, a_n. \]

While we can count inversions in subarrays, there can be cross inversions.

How to get those?

\( \rightarrow \) the merging step, when the head of the list 2 is smaller than head of list 1, we get 1 \( h, i \) number of inversions.

Example:

\( h \rightarrow 10, 12, 17, 19, 25 \rightarrow 29. \n\)
\( h_2 \rightarrow 5, 11, 13, 18, 20, 21. \)

Because 5 is smaller than 10

\( \rightarrow \) \( \exists \) 6 elements in \( h \)

\( \rightarrow \) there are \( \exists \) 6 inversions with 5.

Hence in merging we can add up the inversions.
Quick sort.

\[ T(n) = O(n \log n) \]

Takes the same time as merge sort.

1. Split so that you are going to have small elements on the left & large ones on the right.
2. Choose pivot \( a_i \).
3. Split to have all elements less than or equal to \( a_i \) to the left & greater than \( a_i \) to the right.

Solve left & right recursively.
Time taken

\[ T(n) = T(k) + T(n-1-k) + O(n) \]

\[ \downarrow \]

splitting.

This could potentially be bad. \( \Rightarrow k=1 \Rightarrow T(n) = \Theta(n^2) \)

depending on \( k \), so we should choose a good \( k \).

If \( k = \frac{n}{2} \) (\( a_i \) is the median).

Then left & right are of size \( \frac{n-1}{2} \) each.

Then we get the recursion

\[ T(n) = 2T\left(\frac{n-1}{2}\right) + O(n). \]

\[ \Rightarrow T(n) = \Theta(n \log n). \]

Hence we need to find the median.
Finding the median is a bit hard.

1) Sort the list
2) Find \( \frac{\l}{2} \)th element

\[ T(n) = O(n\log n) \]

but we need median to sort

This is circular!

\[ \implies \text{use merge sort then} \]

But we can do faster! (Given \( a_1, \ldots, a_n \), find \( K^{th} \) element)

Median of \( n \) medians. (Quick select (K))

Take \( a_1, \ldots, a_n \) & break into groups of 5.

\[
\begin{align*}
a_1 & \quad a_2 & \quad a_3 & \quad a_4 & \quad a_5 \\
a_6 & \quad a_7 & \quad a_8 & \quad a_9 & \quad a_{10} \\
\vdots
\end{align*}
\]

\[
\begin{align*}
a_{n-4} & \quad a_{n-3} & \quad a_{n-2} & \quad a_{n-1} & \quad a_n
\end{align*}
\]

Find median of each 5 tuple

Which assume that they are

\[ a_3, a_8, \ldots, a_{n-2} \]
Find median of $a_3, a_8, \ldots, a_{n-2}$ recursively. \( \rightarrow \mu_1 \)

\( \rightarrow \) take this element & split $a_1, \ldots, a_n$

around $\mu_1$

\[ \mu_1 \]

\[ l_1 \quad l_2. \]

\( \mu_1 \) is not the median for $a_1, \ldots, a_n$

But, $\mu_1$ splits the list a bit nicely.

\[
\begin{array}{c|c|c|c|c}
\mu_1 & a_4 & a_2 & a_3 & a_{n-4} \\
\mu_1 & a_5 & a_7 & a_8 & a_{n-3} \\
\mu_1 & a_9 & a_{10} & a_{11} & a_n \\
\end{array}
\]

\( B_1 \leftarrow \)

\( B_2 \rightarrow \)

We know that everything in $B_1 \leq \mu_1$,

& $B_2 \geq \mu_1$,

\( |B_1| = |B_2| = \frac{8n}{10}. \)

\( \Rightarrow \) $l_1, l_2 \leq \frac{7n}{10}$ as there

at least $\frac{3n}{10}$ elements in blocks $B_1, B_2$. 
To select the \( k \)-th element of an array

\[
T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + O(n).
\]

Proof that this is linear time in homework.

\( \Rightarrow \) We can find any element given its position in \( O(n) \).

\( \Rightarrow \) For quick sort, call quickSelect\((\frac{n}{2})\)

\( \Rightarrow \) Gives a good pivot & solve further.

\( \Rightarrow \) \( T(n) = 2T\left(\frac{n}{2}\right) + O(n) \)

\( \Rightarrow \) \( T(n) = O(n \log n) \)
**Lower Bound for Sorting**

In general, if we only compare elements we cannot do faster than O(n log n).

**Proof.**

First look at what kinds of queries we can do on the array.

- We can only check if \( a_i \preceq a_j \).

After we get this answer, we can choose how to proceed, based on this answer.

- Decision tree model of computation.

The leaves represent a sorted list based on the path chosen.
Now this a binary tree with \( n! \) (at least) leaves, representing each sorted permutation.

\[
\#	ext{ leaves in a binary tree of height } h \\
= 2^h \\
\Rightarrow 2^h \geq n!
\]

\[
\Rightarrow h \geq \log n!
\]

\[
\Rightarrow \sum_{i=1}^{2^h} \log i \\
\quad \Rightarrow \sum_{i=1}^{n} \log i \\
\quad \Rightarrow \sum_{i=\frac{n}{2}}^{n} \log \frac{n}{2} \\
\quad \Rightarrow \frac{n}{2} \log \frac{n}{2} = O(n \log n).
\]

\[
\Rightarrow h \geq O(n \log n).
\]

\[
\Rightarrow \exists \text{ a path from the root to the leaf with length } O(n \log n).
\]