Let's write $T(n)$.

In the past, we have referred to $T(n)$ as actual runtime, now we're interested in expected run time.

$T(n)$ = expected time to run Rand-Select on list $A$ of size $n$.

\[
\text{exp. time to partition } A + \text{expected time to run RS on larger of } \text{L and G.} \\
\leq n \leftrightarrow + \frac{7}{6} (\text{let's figure this out}).
\]

Actually worst case (not just expected)

Expected time to run RS on sublist?

We saw that if we can partition the list roughly in $\frac{1}{2}$ each time, then we get linear time.

The closer we get to all even split the better.

Let's classify the partition based on this:

Partition is:

- Lucky - if $L, G$ both have $\leq \frac{3n}{4}$ elements.

  \[\text{ex: } |L| = \frac{n}{2}, \quad |G| = \frac{n}{2}\]

  \[|L| = \frac{n}{3}, \quad |G| = \frac{2n}{3}\]

- Unlucky - if either $L$ or $G$ has $\geq \frac{3n}{4}$ elements.

  \[\text{ex: } |L| = \frac{4n}{5}, \quad |G| = \frac{n}{5}\]

$Pr(\text{lucky})$? - Depends on where the pivot ends up after partition.

Consider list split into 4:

Where does pivot have to end up for "lucky" $Pr(\text{lucky}) = \frac{1}{4}$
\[ \Pr(\text{lucky}) = \Pr(\text{unlucky}) = \frac{1}{2} \]

⇒ Back to \( T(n) \).

2 possibilities for processing \( L \) or \( G \) (lucky or unlucky)

\[
T(n) \leq n + (\text{time for lucky split}) \cdot \Pr(\text{lucky}) + (\text{time for unlucky split}) \cdot \Pr(\text{unlucky split})
\]

\[
\leq n + (\text{worst case ?}) \cdot T(3n/4) \left( \frac{1}{2} \right) + (\text{worst case ?}) \cdot T(n-1) \left( \frac{1}{2} \right)
\]

\[
T(n) \leq n + \underbrace{T(3n/4)}_{\text{(recurse on } G\text{)}} \left( \frac{1}{2} \right) + \underbrace{T(n)}_{\text{(recurse on } L\text{)}} \left( \frac{1}{2} \right)
\]

Multiply both sides by 2:

\[
2T(n) \leq 2n + T(3n/4) + T(n)
\]

\[
T(n) \leq 2n + T(3n/4)
\]

Now apply Master's Theorem:

\[
T(n) = aT(n/b) + cn^i
\]

\[ a = 1, \quad b = 4/3, \quad i = 1 \]

\[ a < b^i \]

Root dominates

⇒ \( \Theta(n) \) \( = \Theta(n) \).

**Important Note:**

We showed average case \( \Theta(n) \).

Can solve in worst case \( \Theta(n) \) ch. 9.3

- Better worst case time but
- Much more complicated algorithm
- Almost always slower \( \Rightarrow \) average case \( \Theta(cn) \)
  - in practice but \( c \) is very high!
Assembly Line Problem

- Cars produced in factory with 2 assembly lines.
- Each line has n stations (that add car parts).
- Stations $S_{1,j}$, $S_{2,j}$ for $j = 1, \ldots, n$.
- Stations $S_{1,j}$ and $S_{2,j}$ perform same task but possibly at different speeds.
- $a_{1,j}$ = time required at $S_{1,j}$
- $a_{2,j}$ = time required at $S_{2,j}$

Cars usually stay on one line (time from $S_{1,j}$ to $S_{1,j+1}$ and $S_{2,j}$ to $S_{2,j+1}$ negligible).

"Special" Rush order cars can switch from line 1 to line 2 to speed up.

$t_{1,j}$ = time to switch from $S_{1,j}$ to $S_{2,j+1}$
$t_{2,j}$ = time to switch from $S_{2,j}$ to $S_{1,j+1}$
Given values for $a_{ij}$, $b_{ij}$, $t_{ij}$, $c_{ij}$, find set of stations to visit that will minimize overall time.

Optimization problem: goal is to minimize or maximizes a specified value.

Other optimization problems: coin changing, shortest path, not searching, sorting.

Ex.:

```
7  ->  9  ->  8  ->  4  ->  8
|   |   |   |   |
|___|___|___|___|
1   2   3   5
```

Note: Greedy doesn't work! $7 + (2) + 5 + (1) + 3 + 4 + 8 = 30$ (optimal is 27)

Brute Force: Consider all possible combinations.
For each of $n$ pairs of stations, $2$ possibilities $\Rightarrow O(2^n)$

Better: Consider smaller versions of the problem, use solutions to those to solve larger + larger versions.

Best time through last station depends on best times through prev stations.
Consider some station \( j \).

Want to know best time through: \( S_{1,j}, S_{2,j} \).

Let:

\[
\begin{align*}
    c(S_{1,j}) & : \text{fastest time through } S_{1,j} \\
    c(S_{2,j}) & : \text{" " " " } S_{2,j} \\
\end{align*}
\]

(If we knew these, could find best time through station \( j \) by finding min of the two.)

How to express \( c(S_{1,j}), c(S_{2,j}) \) in terms of the previous subproblem: \( c(S_{1,j-1}), c(S_{2,j-1}) \)?

\[
C(S_{1,j}) = \begin{cases} 
\left( 3 \right. \text{best time through prev station} + \left. \text{transfer time} \right) + \left. \text{time through this station} \right) \\
\min \left\{ \begin{array}{l}
\text{if prev station on line 1} \\
\text{" " " " " } 2 \left( C(S_{2,j-1}) + t_{2,j-1} + a_{1,j} \right) \\
\text{" " " " " } 0 + a_{1,j} \text{ if } j>1 \\
\text{" " " " " } 1 \left( C(S_{1,j-1}) + t_{1,j-1} + a_{1,j} \right) \\
\end{array} \right. 
\end{cases}
\]

True for which values of \( j \)? \( j \geq 1 \)

What about \( j=1 \)?

\[
C(S_{1,1}) = a_{1,1} \text{ if } j=1
\]
Similarly:

$$c(S_{2,j}) = \begin{cases} a_{2,j} & \text{if } j = 1 \\ \min \left\{ c(S_{2,j-1}) + a_{2,j} \right. \\
\left. c(S_{1,j-1}) + t_{1,j-1} + a_{2,j} \right\} & \text{if } j > 1 \end{cases}$$

**Dynamic Programming:** solving a problem using solutions to sub-problems

When can we use dynamic programming?

*If the problem exhibits:*

- Optimal sub-structure property: optimal solution to the main problem depends on optimal solutions to sub-problems.

Main problem: $c(S_{1,j})$, $c(S_{2,j})$

Sub problems: $c(S_{1,j-1})$, $c(S_{1,j-2})$, $c(S_{1,j-3})$, ..., $c(S_{2,j-1})$, ...

Another problem that has this property? Coin changing!

Optimal number of coins for $n=15$ depends on optimal number of coins for $15-12$

- $15 - 12$
- $15 - 5$
- $15 - 1$

Assembly Line vs. Coin Change (Tabulation vs. Memoization D.P.)