Everyone of you who ever tries to design algorithms in the real world is likely to encounter NP-hard problems. What would you do?

1) Maybe the problem you actually want to solve is not as general, e.g., the input satisfies special properties (bounded degree, planarity or other geometric structure, density/sparsity, ...)

2) Maybe you can settle for an approximation today!

3) Maybe exponential algo. are not that bad — there might be a $C^n$ algo. for $C \approx 1$. 
Def. An algorithm $A$ gives $\ell$-approximation for an optimization problem $P$, if for every input $x$,

\[
\min P(x) \leq A(x) \leq \ell \max P(x) \quad \text{for maximization $P$} \\
\min P(x) \leq A(x) \leq \max P(x) \quad \text{for minimization $P$}
\]

Remark. The definition is for multiplicative approximation. Sometimes we'll be interested in additive approximation

\[
\min P(x) - \beta \leq A(x) \leq \max P(x) + \beta
\]
or in a combination of the two.
Today we'll see three approximation algorithms for NP-hard problems (Vertex-Cover, Set-Cover, Partition). Each has a different approximation factor (2, ln(n)+1, 1+ɛ for every ɛ>0, respectively).

"PTAS": poly-time approximation scheme.

Interestingly, all three are conjectured to be optimal.

Proving optimality, i.e., better approximation is NP-hard, is via the "Probabilistically Checkable Proofs Thm" (PCP)
**Vertex-Cover**

Given an undirected graph $G=(V,E)$, find $V' \subseteq V$, such that for every $(u,v) \in E$, have $u \in V'$ or $v \in V'$. $V'$ should be of min size.

**Approx. Algo.**

While not all edges covered -

Pick $(u,v) \in E$

add both $u,v$ to $V'$

remove $u,v$ and all the edges that touch them

**Note** The algo. always returns a vertex-cover. The run-time is linear.

**Theme:** "Be generous"
Claim: The size of the cover returned by algo is at most twice the size of an optimal cover.

Proof: Observe—the edges the algo considers don't have vertices in common, so every vertex cover has to take at least one vertex from every edge.

Set-Cover

Given $U, S_1, \ldots, S_m \subseteq U, \bigcup_{i=1}^{m} S_i = U$, find $I \subseteq \{1, \ldots, m\}$ such that $\bigcup_{i \in I} S_i = U$.

Minimize $|I|$. 
Approximation algorithm

While not all universe covered

Pick $S_i$: largest

Remove all of the elements in $S_i$.

Note always returns a set-cover.

Run time = $\Theta(\sum_{i=1}^{m} |S_i|)$.

Claim: Gives $(1+\ln(U_1+1))$-approximation.

PF: Assume there’s a cover of size $k$.
Let $U_i =$ universe in iteration $i$.

$\forall i$, $U_i$ can be covered by $k$ sets
$\Rightarrow$ one of them covers at least $\frac{|U_i|}{k}$ elements
$\Rightarrow$ also picks a set of size $\geq \frac{|U_i|}{k}$.
$\Rightarrow \forall i$, $|U_{i+1}| \leq (1 - \frac{1}{k})|U_i|$

$\forall i$, $|U_i| \leq (1 - \frac{1}{k})^i n \leq e^{-\frac{1}{k}} n$ for $n = |U_0|$

$\Rightarrow \forall i \geq k(\ln n + 1)$, $|U_i| < 1$, i.e., $|U_i| = 0$. $\square$
Partition given numbers $s_1 \geq s_2 \geq \cdots \geq s_n$, partition them into two sets $A \cup B = \{s_1, \ldots, s_n\}$, such that $\max\{\sum_{i \in A} s_i, \sum_{i \in B} s_i\}$ minimized.

$$\sum_{i \in A} s_i = 24$$

$$\sum_{i \in B} s_i = 23$$

Approximation algorithm

1) Set $m = \lceil \sqrt[3]{n} \rceil$. Find optimal partition $A', B'$ for $s_{i-m} \leq s_i$.

2) $A \leftarrow A'$; $B \leftarrow B'$

3) For $i < m + \lceil \sqrt[3]{n} \rceil$

   - If $\sum_{i \in A} s_i \leq \sum_{i \in B} s_i$, add $i$ to $A$

   - Otherwise, add $i$ to $B$.

Theme: optimization for sub-problem sometimes gives approx for the whole problem.
Note. The algorithm always returns a partition.

Run-time = $\Theta(2^n + n) = \Theta(n)$

$\varepsilon$ is constant

Claim. The algorithm gives a $(1+\varepsilon)$-approximation.

Proof. Without loss of generality, assume $\sum_{i \in A'} s_i \geq \sum_{i \in B'} s_i$.

Case 1. $\sum_{i \in A'} s_i \geq \frac{1}{2} \sum_{i = 1}^n s_i$.

$\Rightarrow A = A'$ and this has to be optimal:
partition of cost $\leq \sum_{i \in A'} s_i$ would induce better
partition for $\sum_{i = 1}^n s_i$.

Case 2. $\sum_{i \in A'} s_i \leq \frac{1}{2} \sum_{i = 1}^n s_i$.

Note: $|\sum_{i \in A} s_i - \sum_{i \in B} s_i| \leq s_{m+1}$

$\Rightarrow$ The algorithm output a partition of cost $\leq \frac{1}{2} \sum_{i = 1}^n s_i + s_{m+1}$
while $\text{opt} \geq \sum_{i = 1}^n s_i \Rightarrow$ ratio $\leq 1 + \frac{s_{m+1}}{(m-1)s_{m+1}} \leq 1 + \varepsilon$. 

□