Given a set $S$ of positive integers $\{x_1, x_2, \ldots, x_n\}$ and a target value $t$, and we are asked whether there exists a subset $S' \subseteq S$ that sums exactly to $t$.

The optimization problem is to determine the subset whose sum is as large as possible but not larger than $t$.

Suppose we are given $0 < \epsilon < 1$. Let $z^* \leq t$ denote the optimum sum.

Approximation Problem: Return a value $z \leq t$ such that $z \geq z^*(1 - \epsilon)$.

If we think of this as a knapsack problem, we want our knapsack to be within a factor of $(1 - \epsilon)$ of being as full as possible. So, if $\epsilon = 0.1$, then the knapsack should be at least 90% as full as the best possible.
For example, if $S = \{1, 4, 6\}$ and $t = 8$ then the successive lists would be

$L_0 =<0>$
$L_1 =<0> \cup <0 + 1> = <0, 1>$
$L_2 =<0, 1> \cup <0 + 4, 1 + 4> = <0, 1, 4, 5>$
$L_3 =<0, 1, 4, 5> \cup <0 + 6, 1 + 6, 4 + 6, 5 + 6> = <0, 1, 4, 5, 6, 7, 10, 11>$

The last list would have the elements 10 and 11 removed, and the final answer would be 7.

The algorithm runs in $\Omega(2^n)$ time in the worst case, because this is the number of sums that are generated if there are no duplicates, and no items are removed.
To convert this into an approximation algorithm, we will:

• Introduce a way to “trim” the lists to decrease their sizes.
  – The idea is that if the list L contains two numbers that are very close to one another, e.g. 91, 048 and 91, 050, then we should not need to keep both of these numbers in the list. One of them is good enough for future approximations.

How much trimming can we allow and still keep our approximation bound?

Furthermore, will we be able to reduce the list sizes from exponential to polynomial?
The trimming must also depend on $\epsilon$. We select $\delta = \epsilon/n$.

Note that $0 < \delta < 1$. Assume that the elements of $L$ are sorted. We walk through the list. Let $z$ denote the last untrimmed element in $L$, and let $y \geq z$ be the next element to be considered. If $(y - z) / y \leq \delta$ then we trim $y$ from the list.

Equivalently, this means that the final trimmed list cannot contain two value $y$ and $z$ such that $(1 - \delta) \leq y \leq z$.

We can think of $z$ as representing $y$ in the list.

Example:
Given $\delta = 0.1$ and $L = <10, 11, 12, 15, 20, 21, 22, 23, 24, 29>$
trimmed list $L' = <10, 12, 15, 20, 23, 29>$
Another way to visualize trimming is to break the interval from \([1, t]\) into a set of *buckets* of exponentially increasing size. Let \(d = 1/(1 - \delta)\). Note that \(d > 1\). Consider the intervals

\([1, d], [d, d^2], [d^2, d^3], \ldots, [d^{k-1}, d^k]\),

where \(d^k \geq t\). If \(z \leq y\) are in the same interval \([d^{i-1}, d^i]\) then

\[
\frac{y - z}{y} \leq \frac{d^i - d^{i-1}}{d^i} = 1 - \frac{1}{d} = \delta.
\]

Thus, we cannot have more than one item within each bucket.
Proof

Claim: The number of distinct items in a trimmed list is $O((n \log t)/\varepsilon)$, which is polynomial in input size and $1/\varepsilon$.

Proof: We know that each pair of consecutive elements in a trimmed list differ by a ratio of at least $d = 1/(1 - \delta) > 1$. Let $k$ denote the number of elements in the trimmed list, ignoring the element of value $0$. Thus, the smallest nonzero value and maximum value in the trimmed list differ by a ratio of at least $d^{k-1}$. Since the smallest (nonzero) element is at least as large as 1, and the largest is no larger than $t$, then it follows that $d^{k-1} \leq t/1 = t$. Taking the natural log of both sides we have $(k - 1) \ln d \leq \ln t$. Using the facts that $\delta = \varepsilon/n$ and the log identity that $\ln(1 + x) \leq x$, we have

$$k - 1 \leq \frac{\ln t}{\ln d} = \frac{\ln t}{-\ln(1 - \delta)} \leq \frac{\ln t}{\delta} = \frac{n \ln t}{\varepsilon}$$

$$k = O\left(\frac{n \log t}{\varepsilon}\right).$$

Observe that the input size is at least as large as $n$ (since there are $n$ numbers) and at least as large as $\log t$ (since it takes $\log t$ digits to write down $t$ on the input). Thus, this function is polynomial in the input size and $1/\varepsilon$. 
Approximate Subset Sum

The running time of the procedure is $O(n|L|)$ which is $O(n^2 \ln t/\epsilon)$ by the earlier claim.
Approximate Subset Sum Example

For example, consider the set $S = \{104, 102, 201, 101\}$ and $t = 308$ and $\epsilon = 0.20$. We have $\delta = \epsilon/4 = 0.05$.

\[
\begin{align*}
\text{init:} & \quad L_0 = \langle 0 \rangle \\
\text{merge:} & \quad L_1 = \langle 0, 104 \rangle \\
\text{trim:} & \quad L_1 = \langle 0, 104 \rangle \\
\text{remove:} & \quad L_1 = \langle 0, 104 \rangle \\
\text{merge:} & \quad L_2 = \langle 0, 102, 104, 206 \rangle \\
\text{trim:} & \quad L_2 = \langle 0, 102, 206 \rangle \\
\text{remove:} & \quad L_2 = \langle 0, 102, 206 \rangle \\
\text{merge:} & \quad L_3 = \langle 0, 102, 201, 206, 303, 407 \rangle \\
\text{trim:} & \quad L_3 = \langle 0, 102, 201, 303, 407 \rangle \\
\text{remove:} & \quad L_3 = \langle 0, 102, 201, 303 \rangle \\
\text{merge:} & \quad L_4 = \langle 0, 101, 102, 201, 203, 302, 303, 404 \rangle \\
\text{trim:} & \quad L_4 = \langle 0, 101, 201, 302, 404 \rangle \\
\text{remove:} & \quad L_4 = \langle 0, 101, 201, 302 \rangle 
\end{align*}
\]

The final output is 302.
The optimum is $307 = 104 + 102 + 101$.
Actual relative error in this case is within 2%.
Another well-known NP-Complete problem.

**Bin Packing:** Given a set of $n$ objects, let $s_i$ denote the size of the $i$th object (assume that the sizes have been normalized so that $0 < s_i < 1$). We want to put these objects into a set of bins (each bin can hold a subset of objects whose total size is at most 1) such that we use the fewest possible bins.

Many of these applications involve not only the size of the object but their geometric shape as well.

- Packing boxes into a truck
- Cutting the maximum number of pieces of certain shapes out of a piece of sheet metal.

However, even if we ignore the geometry, and just consider the sizes of the objects, the decision problem is still NP-complete. (The reduction is from the knapsack problem.)
First-fit heuristic: We start with an unlimited number of empty bins. We take each object in turn, and find the first bin that has space to hold this object. We put this object in this bin.

We claim that first-fit uses at most twice as many bins as the optimum. That is, if the optimal solution uses $b_{\text{opt}}$ bins, and first-fit uses $b_{\text{ff}}$ bins, then we show below that:

$$\frac{b_{\text{ff}}}{b_{\text{opt}}} \leq 2.$$
Theorem: The first-fit heuristic achieves a ratio bound of 2.

Proof: Consider an instance \( \{s_1, \ldots, s_n\} \) of the bin packing problem. Let \( S = \sum_i s_i \) denote the sum of all the object sizes. Let \( b_{\text{opt}} \) denote the optimal number of bins, and \( b_{\text{FF}} \) denote the number of bins used by first-fit.

First, observe that since no bin can hold more than one unit’s worth of items, and we have a total of \( S \) units to be stored, it follows that we need a minimum of \( S \) bins to store everything. (And this would be achieved only if every bin were filled exactly to the top.) Thus, \( b_{\text{opt}} \geq S \).

Next, we claim that \( b_{\text{FF}} \leq 2S \). To see this, let \( t_i \) denote the total size of the objects that first-fit puts into bin \( i \). There cannot be two bins \( i < j \) such that \( t_i + t_j < 1 \). The reason is that any item we decided to put into bin \( j \) must be small enough to fit into bin \( i \). Thus, the first-fit algorithm would never put such an item into bin \( j \). In particular, this implies that for all \( i \), \( t_i + t_{i+1} \geq 1 \) (where indices are taken circularly modulo the number of bins). Thus we have

\[
b_{\text{FF}} = \sum_{i=1}^{b_{\text{FF}}} 1 \leq \sum_{i=1}^{b_{\text{FF}}}(t_i + t_{i+1}) = \sum_{i=1}^{b_{\text{FF}}} t_i + \sum_{i=1}^{b_{\text{FF}}} t_{i+1} = S + S = 2S \leq 2b_{\text{opt}},
\]

which completes the proof.
**Other Heuristics**

**Best-fit:** attempts to put the object into the bin in which it fits most closely with the available space (assuming that there is sufficient space).

**First-fit-decreasing:** objects are first sorted in decreasing order of size.

**Note:**
A more careful (and complicated) proof establishes that *first-fit* has a approximation ratio that is a bit smaller than 2, and in fact $17/10 = 1.7$ is possible.

*Best-fit* has a very similar bound.

It can be shown that *first-fit-decreasing* has a significantly better bound than either of these. In particular, it achieves a ratio bound of $11/9 \approx 1.222$. 

Rhodes College
Facility Location: Imagine that Blockbuster Video wants to open 50 stores in some city.

- The company asks you to determine the best locations for these stores.
- The condition is that you are to minimize the maximum distance that any resident of the city must drive in order to arrive at the nearest store.

K-center Problem: Given an undirected graph $G = (V, E)$ with nonnegative edge weights, and an integer $k$, compute a subset of $k$ vertices $C \subseteq V$, called centers, such that the maximum distance between any vertex in $V$ and its nearest center in $C$ is minimized.
Let $G = (V,E)$ denote the graph, and let $w(u, v)$ denote the weight of edge $(u, v)$. ($w(u, v) = w(v, u)$ because $G$ is undirected.)

For each pair of vertices, $u, v \in V$, let $d(u, v) = d(u, v)$ denote the distance between $u$ to $v$, that is, the length of the shortest path from $u$ to $v$.

- **Note**: the shortest path distance satisfies the triangle inequality.

Consider a subset $C \subseteq V$ of vertices, the centers. For each vertex $v \in V$ we can associate it with its nearest center in $C$.

For each center $c_i \in C$ we define its neighborhood to be the subset of vertices for which $c_i$ is the closest center

$$V(c_i) = \{v \in V \mid d(v, c_i) \leq d(v, c_j), \text{ for } i \neq j\}.$$
The k-center Problem

**Assumption:** no ties for distances to the closest center

$V(c_1), V(c_2), \ldots, V(c_k)$ forms a *partition* of the vertex set of $G$

The *bottleneck distance* associated with each center is the distance to its farthest vertex in $V(c_i)$, that is, $\Delta(c_i) = \max_{v \in V(c_i)} d(v, c_i)$

**Overall bottleneck distance:** $\Delta(C) = \max_{c_i \in C} \Delta(c_i)$

**k-center problem:** Given a weighted undirected graph $G = (V, E)$, and an integer $k \leq |V|$, find a subset $C \subseteq V$ of size $k$ such that $\Delta(C)$ is minimized.

Decision-problem formulation of the k-center problem is NP-complete (reduction from dominating set).
Greedy Approximation Algorithm

Greedy approximation to k-center.

```
KCenterApprox(G, k) {
    C = empty_set
    for each u in V do  // initialize distances
        d[u] = INFINITY
    for i = 1 to k do {  // main loop
        Find the vertex u such that d[u] is maximum
        Add u to C  // u is current bottleneck vertex
        // update distances
        Compute the distance from each vertex v to its closest vertex in C, denoted d[v]
    }
    return C  // final centers
}
```

Overall running time is $O(kE \log V)$. 
How bad could greedy be?

We want to show that this approximation algorithm always produces a final distance $\Delta(C)$ that is within a factor of 2 of the distance of the optimal solution.

Let $O = \{o_1, o_2, \ldots, o_k\}$ denote the centers of the optimal solution. 
Let $\Delta^* = \Delta(O)$ be the optimal bottleneck distance.
Let $G = \{g_1, g_2, \ldots, g_k\}$ be the centers found by the greedy approximation. 
Also, let $g_{k+1}$ denote the next center that would have been added next, that is, the bottleneck vertex for $G$.
Let $\Delta_G$ denote the bottleneck distance for $G$.
Notice that the distance from $g_{k+1}$ to its nearest center is equal $\Delta_G$.  

An example showing that greedy can be a factor 2 from optimal. Here $k = 2$. 

Approximation Bound
Theorem: The greedy approximation has a ratio bound of 2, that is $\Delta_G/\Delta_* \leq 2$.

Proof: Let $G' = \{g_1, g_2, \ldots, g_k, g_{k+1}\}$ be the $(k + 1)$-element set consisting of the greedy centers together with the next greedy center $g_{k+1}$. First observe that for $i \neq j$, $d(g_i, g_j) \geq \Delta_G$. This follows as a result of our greedy selection strategy. As each center is selected, it is selected to be at the maximum (bottleneck) distance from all the previous centers. As we add more centers, the maximum distance between any pair of centers decreases. Since the final bottleneck distance is $\Delta_G$, all the centers are at least this far apart from one another.

![Diagram showing greedy centers and optimal centers with analysis for $k = 5$.](image)

Each $g_i \in G'$ is associated with its closest center in the optimal solution, that is, each $g_i$ belongs to $V(o_m)$ for some $m$. Because there are $k$ centers in $O$, and $k + 1$ elements in $G'$, it follows from the pigeon-hole principal, that at least two centers of $G'$ are in the same set $V(o_m)$ for some $m$. (In the figure, the greedy centers $g_4$ and $g_5$ are both in $V(o_2)$). Let these be denoted $g_i$ and $g_j$.

Since $\Delta_*$ is the bottleneck distance for $O$, we know that the distance from $g_i$ to $o_k$ is of length at most $\Delta_*$ and similarly the distance from $o_k$ to $g_j$ is at most $\Delta_*$. By concatenating these two paths and the triangle inequality, it follows that there exists a path of length at most $2\Delta_*$ from $g_i$ to $g_j$, and hence we have $d(g_i, g_j) \leq 2\Delta_*$. But from the comments above we have $d(g_i, g_j) \geq \Delta_G$. Therefore, $\Delta_G \leq d(g_i, g_j) \leq 2\Delta_*$. Therefore $\Delta_G/\Delta_* \leq 2$, as desired.
What do *expert* programmers do when faced by an intractable problem?

**Brute-Force Solution:** $O(n!)$

**Dynamic Programming Algorithms:** $O(n^2 2^n)$

**Selling on Ebay:** $O(1)$

Still working on your route?

Shut the hell up.