Approximation Algorithms III

Lecture 10
September 24, 2018
10.1: Subset Sum
Subset Sum

**Instance:** $X = \{x_1, \ldots, x_n\}$ – $n$ integer positive numbers, $t$ - target number

**Question:** $\exists$ subset of $X$ s.t. sum of its elements is $t$?

Assume $x_1, \ldots, x_n$ are all $\leq n$. Then this problem can be solved in

(A) The problem is still **NP-Hard**, so probably exponential time.
(B) $O(n^3)$.
(C) $2^{O(\log^2 n)}$.
(D) $O(n \log n)$.

(E) None of the above.
Subset Sum

**Instance:** \( X = \{x_1, \ldots, x_n\} \) \( n \) integer positive numbers, \( t \) - target number

**Question:** \( \exists \) subset of \( X \) s.t. sum of its elements is \( t \)?

**SolveSubsetSum** \((X, t, M)\)

\[
\begin{align*}
  b[0 & \ldots Mn] \leftarrow \text{false} \\
  & \quad \text{// } b[x] \text{ is true if } x \text{ can be} \\
  & \quad \text{// realized by subset of } X. \\
  b[0] & \leftarrow \text{true.} \\
  \text{for } i = 1, \ldots, n \text{ do} \\
  \quad \text{for } j = Mn \text{ down to } x_i \text{ do} \\
\end{align*}
\]
Subset Sum

**Instance:** \( X = \{ x_1, \ldots, x_n \} \) – \( n \) integer positive numbers, \( t \) - target number

**Question:** \( \exists \) subset of \( X \) s.t. sum of its elements is \( t \)?

\[
\text{SolveSubsetSum} (X, t, M)
\]

\[
b[0 \ldots Mn] \leftarrow \text{false}
\]

// \( b[x] \) is true if \( x \) can be

// realized by subset of \( X \).

\[
b[0] \leftarrow \text{true}.
\]

for \( i = 1, \ldots, n \) do

for \( j = Mn \) down to \( x_i \) do

//
Subset Sum

**Subset Sum**

**Instance:** \( X = \{x_1, \ldots, x_n\} \) \( n \) integer positive numbers, \( t \) - target number

**Question:** \( \exists \) subset of \( X \) s.t. sum of its elements is \( t \)?

\[
\text{SolveSubsetSum} \ (X, t, M)
\]

\[
b[0 \ldots Mn] \leftarrow \text{false}
\]

\[
// \ b[x] \text{ is } \text{true} \text{ if } x \text{ can be}
\]

\[
// \text{realized by subset of } X.
\]

\[
b[0] \leftarrow \text{true}.
\]

\[
\text{for } i = 1, \ldots, n \ \text{do}
\]

\[
\text{for } j = Mn \ \text{down to } x_i \ \text{do}
\]
Subset Sum

**Instance:** \( X = \{x_1, \ldots, x_n\} \) – \( n \) integer positive numbers, \( t \) - target number

**Question:** \( \exists \) subset of \( X \) s.t. sum of its elements is \( t \)?

**SolveSubsetSum** \((X, t, M)\)

\[
\begin{align*}
&M: \text{Max value input numbers.} \\
&R.T. \quad O(Mn^2) \\
&b[0 \ldots Mn] \leftarrow \text{false} \\
&\quad // b[x] \text{ is true if } x \text{ can be} \\
&\quad // \text{realized by subset of } X. \\
&b[0] \leftarrow \text{true}.
\end{align*}
\]

for \( i = 1, \ldots, n \) do

for \( j = Mn \) down to \( x_i \) do

...
Subset Sum

Efficient algorithm???

1. Algorithm solving Subset Sum in $O(Mn^2)$.
2. $M$ might be prohibitly large...
3. if $M = 2^n$ $\implies$ algorithm is not polynomial time.
4. Subset Sum is \textbf{NPC}.
5. Still want to solve quickly even if $M$ huge.
6. Optimization version:

\textbf{Subset Sum Optimization}

\textbf{Instance}: $(X, t)$: A set $X$ of $n$ positive integers, and a target number $t$.

\textbf{Question}: The largest number $\gamma_{opt}$ one can represent as a subset sum of $X$ which is smaller or equal to $t$. 
Subset Sum

Efficient algorithm???

1. Algorithm solving Subset Sum in $O(Mn^2)$.
2. $M$ might be prohibitly large...
3. if $M = 2^n \implies$ algorithm is not polynomial time.
4. Subset Sum is NPC.
5. Still want to solve quickly even if $M$ huge.
6. Optimization version:

Subset Sum Optimization

Instance: $(X, t)$: A set $X$ of $n$ positive integers, and a target number $t$.

Question: The largest number $\gamma_{opt}$ one can represent as a subset sum of $X$ which is smaller or equal to $t$. 
Subset Sum

Efficient algorithm???

1. Algorithm solving Subset Sum in $O(Mn^2)$.
2. $M$ might be prohibitly large...
3. if $M = 2^n \implies$ algorithm is not polynomial time.
4. Subset Sum is NPC.
5. Still want to solve quickly even if $M$ huge.
6. Optimization version:

**Subset Sum Optimization**

**Instance**: $(X, t)$: A set $X$ of $n$ positive integers, and a target number $t$.

**Question**: The largest number $\gamma_{opt}$ one can represent as a subset sum of $X$ which is smaller than or equal to $t$. 
Subset Sum

2-approximation

Lemma

1. \((X, t)\); Given instance of Subset Sum. \(\gamma_{opt} \leq t\): Opt.
2. \(\implies\) Compute legal subset with sum \(\geq \gamma_{opt}/2\).
3. Running time \(O(n \log n)\).

Proof.

1. Sort numbers in \(X\) in decreasing order.
2. Greedily - add numbers from largest to smallest (if possible).
3. \(s\): Generates sum.
Subset Sum

2-approximation

Lemma

1. \((X, t)\); Given instance of **Subset Sum**. \(\gamma_{opt} \leq t\):
   - Opt.

2. \(\Rightarrow\) Compute legal subset with sum \(\geq \gamma_{opt}/2\).

3. Running time \(O(n \log n)\).

Proof.

1. Sort numbers in \(X\) in decreasing order.
2. Greedily - add numbers from largest to smallest (if possible).
3. \(s\): Generates sum.
Subset Sum

2-approximation

Lemma

1. \((X, t)\); Given instance of \textit{Subset Sum}. \(\gamma_{opt} \leq t\): Opt.
2. \(\implies\) Compute legal subset with sum \(\geq \gamma_{opt}/2\).
3. Running time \(O(n \log n)\).

Proof.

1. Sort numbers in \(X\) in decreasing order.
2. Greedily - add numbers from largest to smallest (if possible).
3. \(s\): Generates sum.
Subset Sum

2-approximation

Lemma

1. \((X, t)\); Given instance of Subset Sum. \(\gamma_{\text{opt}} \leq t\):
   Opt.
2. \(\implies\) Compute legal subset with sum \(\geq \gamma_{\text{opt}}/2\).
3. Running time \(O(n \log n)\).

Proof.

1. Sort numbers in \(X\) in decreasing order.
2. Greedily - add numbers from largest to smallest (if possible).
3. \(s\): Generates sum.
Subset Sum

2-approximation

Lemma

1. \((X, t)\); Given instance of Subset Sum. \(\gamma_{\text{opt}} \leq t\): Opt.
2. \(\implies\) Compute legal subset with sum \(\geq \frac{\gamma_{\text{opt}}}{2}\).
3. Running time \(O(n \log n)\).

Proof.

1. Sort numbers in \(X\) in decreasing order.
2. Greedily - add numbers from largest to smallest (if possible).
3. \(s\): Generates sum.
10.1.1: On the complexity of $\varepsilon$-approximation algorithms
Polynomial Time Approximation Schemes

Definition (PTAS)

**PROB**: Maximization problem.

\( \varepsilon > 0 \): approximation parameter.

\( \mathcal{A}(I, \varepsilon) \) is a **polynomial time approximation scheme (PTAS)** for **PROB**:

1. \( \forall I: (1 - \varepsilon) |\text{opt}(I)| \leq |\mathcal{A}(I, \varepsilon)| \leq |\text{opt}(I)|, \)
2. \( |\text{opt}(I)| \): opt price,
3. \( |\mathcal{A}(I, \varepsilon)| \): price of solution of \( \mathcal{A} \).
4. \( \mathcal{A} \) running time polynomial in \( n \) for fixed \( \varepsilon \).

For minimization problem:

\( |\text{opt}(I)| \leq |\mathcal{A}(I, \varepsilon)| \leq (1 + \varepsilon)|\text{opt}(I)|. \)
Polynomial Time Approximation Schemes

**Definition (PTAS)**

**PROB:** Maximization problem.

\( \varepsilon > 0 \): approximation parameter.

\( A(I, \varepsilon) \) is a *polynomial time approximation scheme* (PTAS) for **PROB**:

1. \( \forall I: (1 - \varepsilon) \mid \text{opt}(I) \mid \leq \mid A(I, \varepsilon) \mid \leq \mid \text{opt}(I) \mid \),

2. \( \mid \text{opt}(I) \mid \): opt price,

3. \( \mid A(I, \varepsilon) \mid \): price of solution of \( A \).

4. \( A \) running time polynomial in \( n \) for fixed \( \varepsilon \).

For minimization problem:

\( \mid \text{opt}(I) \mid \leq \mid A(I, \varepsilon) \mid \leq (1 + \varepsilon) \mid \text{opt}(I) \mid \).
Polynomial Time Approximation Schemes

1. Example: Approximation algorithm with running time $O(n^{1/\varepsilon})$ is a PTAS. Algorithm with running time $O(1/\varepsilon^n)$ is not.

2. Fully polynomial...

Definition (FPTAS)

An approximation algorithm is fully polynomial time approximation scheme (FPTAS) if it is a PTAS, and its running time is polynomial both in $n$ and $1/\varepsilon$.

3. Example: PTAS with running time $O(n^{1/\varepsilon})$ is not a FPTAS.

4. Example: PTAS with running time $O(n^2/\varepsilon^3)$ is a FPTAS.
Polynomial Time Approximation Schemes

1. Example: Approximation algorithm with running time $O(n^{1/\varepsilon})$ is a PTAS.
   Algorithm with running time $O(1/\varepsilon^n)$ is not.

2. Fully polynomial...

Definition (FPTAS)

An approximation algorithm is **fully polynomial time approximation scheme** (FPTAS) if it is a PTAS, and its running time is polynomial both in $n$ and $1/\varepsilon$.

3. Example: PTAS with running time $O(n^{1/\varepsilon})$ is not a FPTAS.

4. Example: PTAS with running time $O(n^2/\varepsilon^3)$ is a FPTAS.
Polynomial Time Approximation Schemes

1. Example: Approximation algorithm with running time $O(n^{1/\varepsilon})$ is a PTAS.
   Algorithm with running time $O(1/\varepsilon^n)$ is not.

2. Fully polynomial...

Definition (FPTAS)

An approximation algorithm is **fully polynomial time approximation scheme** (FPTAS) if it is a PTAS, and its running time is polynomial both in $n$ and $1/\varepsilon$.

3. Example: PTAS with running time $O(n^{1/\varepsilon})$ is not a FPTAS.

4. Example: PTAS with running time $O(n^2/\varepsilon^3)$ is a FPTAS.
Polynomial Time Approximation Schemes

1. Example: Approximation algorithm with running time $O(n^{1/\varepsilon})$ is a PTAS. Algorithm with running time $O(1/\varepsilon^n)$ is not.

2. Fully polynomial...

Definition (FPTAS)

An approximation algorithm is fully polynomial time approximation scheme (FPTAS) if it is a PTAS, and its running time is polynomial both in $n$ and $1/\varepsilon$.

3. Example: PTAS with running time $O(n^{1/\varepsilon})$ is not a FPTAS.

4. Example: PTAS with running time $O(n^2/\varepsilon^3)$ is a FPTAS.
Approximating Subset Sum

**Subset Sum Approx**

**Instance:** \((X, t, \varepsilon)\): A set \(X\) of \(n\) positive integers, a target number \(t\), and parameter \(\varepsilon > 0\).

**Question:** A number \(z\) that one can represent as a subset sum of \(X\), such that \((1 - \varepsilon)\gamma_{\text{opt}} \leq z \leq \gamma_{\text{opt}} \leq t\).
Approximating Subset Sum
Looking again at the exact algorithm

`ExactSubsetSum(S, t)`

1. $n \leftarrow |S|$
2. $P_0 \leftarrow \{0\}$
3. for $i = 1 \ldots n$ do
   1. $P_i \leftarrow P_{i-1} \cup (P_{i-1} + x_i)$
   2. Remove from $P_i$ all elements $> t$

return largest element in $P_n$

1. $S = \{a_1, \ldots, a_n\}$
   $x + S = \{a_1 + x, a_2 + x, \ldots a_n + x\}$
2. Lists might explode in size.
Trim the lists...

$L'$: Inc. sorted list of numbers

\[
\text{\textbf{Trim}}(L', \delta) \\
L = \langle y_1 \ldots y_m \rangle \\
curr \leftarrow y_1 \\
L_{\text{out}} \leftarrow \{ y_1 \} \\
\text{for } i = 2 \ldots m \text{ do} \\
\quad \text{if } y_i > curr \cdot (1 + \delta) \\
\quad \quad \text{Append } y_i \text{ to } L_{\text{out}} \\
\quad curr \leftarrow y_i \\
\text{return } L_{\text{out}}
\]

**Definition**

For two positive real numbers $z \leq y$, the number $y$ is a $\delta$-approximation to $z$ if

\[
\frac{y}{1 + \delta} \leq z \leq y.
\]

**Observation**

If $x \in L'$ then there exists a number $y \in L_{\text{out}}$ such that $y \leq x \leq y(1 + \delta)$, where
Trim the lists...

**ApproxSubsetSum**($S$, $t$)

```
// $S = \{x_1, \ldots, x_n\}$,
// $x_1 \leq x_2 \leq \ldots \leq x_n$

n ← $|S|$, $L_0 ← \{0\}$,
$\delta = \varepsilon/2n$

for $i = 1 \ldots n$ do
  $E_i ← L_{i-1} \cup (L_{i-1} + x_i)$
  $L_i ← \text{Trim}(E_i, \delta)$
  Remove from $L_i$ elems $> t$.

return largest element in $L_n$
```

$E_i$: Computed by merging two sorted lists in linear time.
Understanding trimming
Understanding trimming
Understanding trimming
Understanding trimming
Understanding trimming
Understanding trimming
Understanding trimming
Understanding trimming
Understanding trimming
Understanding trimming
Understanding trimming
Understanding trimming
Understanding trimming
Understanding trimming
Understanding trimming
Understanding trimming
Understanding trimming
Understanding trimming
Understanding trimming
Understanding trimming
Understanding trimming
Understanding trimming
Understanding trimming
Remark

1. Can assume that trimmed lists $L_i$ are sorted...
2. Algorithm: $E_i \leftarrow L_{i-1} \cup (L_{i-1} + x_i)$
3. So, this is just copy, shift, and merge of two sorted lists.
4. ... resulting in a sorted list.
5. takes linear time in size of lists.
Remark

1. Can assume that trimmed lists $L_i$ are sorted...
2. Algorithm: $E_i \leftarrow L_{i-1} \cup (L_{i-1} + x_i)$
3. So, this is just copy, shift, and merge of two sorted lists.
4. ... resulting in a sorted list.
5. takes linear time in size of lists.
Remark

1. Can assume that trimmed lists \( L_i \) are sorted...
2. Algorithm: \( E_i \leftarrow L_{i-1} \cup (L_{i-1} + x_i) \)
3. So, this is just copy, shift, and merge of two sorted lists.
4. ... resulting in a sorted list.
5. takes linear time in size of lists.
Remark

1. Can assume that trimmed lists $L_i$ are sorted...
2. Algorithm: $E_i \leftarrow L_{i-1} \cup (L_{i-1} + x_i)$
3. So, this is just copy, shift, and merge of two sorted lists.
4. ... resulting in a sorted list.
5. takes linear time in size of lists.
Remark

1. Can assume that trimmed lists $L_i$ are sorted...
2. Algorithm: $E_i \leftarrow L_{i-1} \cup (L_{i-1} + x_i)$
3. So, this is just copy, shift, and merge of two sorted lists.
4. ... resulting in a sorted list.
5. takes linear time in size of lists.
Analysis

1. $E_i$ list generated by algorithm in $i$th iteration.
2. $P_i$: list of numbers (no trimming).

Claim

For any $x \in P_i$ there exists $y \in L_i$ such that $y \leq x \leq (1 + \delta)^i y$.

Proof

1. If $x \in P_1$ then follows by observation above.
2. If $x \in P_{i-1}$ implies (induction) $\exists y' \in L_{i-1}$ s.t. $y' \leq x \leq (1 + \delta)^{i-1} y'$.
3. By observation $\exists y \in L_i$ s.t. $y \leq y' \leq (1 + \delta) y$, As such,
Analysis

1. $E_i$: list generated by algorithm in $i$th iteration.
2. $P_i$: list of numbers (no trimming).

Claim

For any $x \in P_i$ there exists $y \in L_i$ such that
$y \leq x \leq (1 + \delta)^i y$.

Proof

1. If $x \in P_1$ then follows by observation above.
2. If $x \in P_{i-1} \implies$ (induction) $\exists y' \in L_{i-1}$ s.t.
   $y' \leq x \leq (1 + \delta)^{i-1} y'$.
3. By observation $\exists y \in L_i$ s.t. $y \leq y' \leq (1 + \delta) y$,
   As such,
Analysis

1. $E_i$: list generated by algorithm in $i$th iteration.
2. $P_i$: list of numbers (no trimming).

Claim

For any $x \in P_i$ there exists $y \in L_i$ such that

$y \leq x \leq (1 + \delta)^i y$.

Proof

1. If $x \in P_1$ then follows by observation above.
2. If $x \in P_{i-1} \implies$ (induction) $\exists y' \in L_{i-1}$ s.t.
   
   $y' \leq x \leq (1 + \delta)^{i-1} y'$.
3. By observation $\exists y \in L_i$ s.t. $y \leq y' \leq (1 + \delta)y$,

As such,
Proof continued

1. If $x \in P_i \setminus P_{i-1}$ $\implies$ $x = \alpha + x_i$, for some $\alpha \in P_{i-1}$.

2. By induction, $\exists \alpha' \in L_{i-1}$ s.t.
   \[ \alpha' \leq \alpha \leq (1 + \delta)^{i-1}\alpha'. \]

3. Thus, $\alpha' + x_i \in E_i$.

4. $\exists x' \in L_i$ s.t. $x' \leq \alpha' + x_i \leq (1 + \delta)x'$.

5. Thus,
   \[ x' \leq \alpha' + x_i \leq \alpha + x_i = x \leq (1 + \delta)^{i-1}\alpha' + x_i \leq (1 + \delta)^{i-1}(\alpha' + x_i) \leq (1 + \delta)^{i}x'. \]

$\blacksquare$
Proof continued

1. If \( x \in P_i \setminus P_{i-1} \implies x = \alpha + x_i \), for some \( \alpha \in P_{i-1} \).

2. By induction, \( \exists \alpha' \in L_{i-1} \) s.t. \( \alpha' \leq \alpha \leq (1 + \delta)^{i-1} \alpha' \).

3. Thus, \( \alpha' + x_i \in E_i \).

4. \( \exists x' \in L_i \) s.t. \( x' \leq \alpha' + x_i \leq (1 + \delta)x' \).

5. Thus,
\[
x' \leq \alpha' + x_i \leq \alpha + x_i = x \leq (1+\delta)^{i-1} \alpha' + x_i \leq (1 + \delta)^{i-1}(\alpha' + x_i) \leq (1 + \delta)^i x'.
\]
Proof continued

1. If \( x \in P_i \setminus P_{i-1} \implies x = \alpha + x_i \), for some \( \alpha \in P_{i-1} \).
2. By induction, \( \exists \alpha' \in L_{i-1} \) s.t.
   \( \alpha' \leq \alpha \leq (1 + \delta)^{i-1}\alpha' \).
3. Thus, \( \alpha' + x_i \in E_i \).
4. \( \exists x' \in L_i \) s.t. \( x' \leq \alpha' + x_i \leq (1 + \delta)x' \).
5. Thus,
   \( x' \leq \alpha' + x_i \leq \alpha + x_i = x \leq (1 + \delta)^{i-1}\alpha' + x_i \leq (1 + \delta)^i x' \).
Proof continued

1. If $x \in P_i \setminus P_{i-1}$ $\implies$ $x = \alpha + x_i$, for some $\alpha \in P_{i-1}$.

2. By induction, $\exists \alpha' \in L_{i-1}$ s.t. $\alpha' \leq \alpha \leq (1 + \delta)^{i-1}\alpha'$.

3. Thus, $\alpha' + x_i \in E_i$.

4. $\exists x' \in L_i$ s.t. $x' \leq \alpha' + x_i \leq (1 + \delta)x'$.

5. Thus,

$$x' \leq \alpha' + x_i \leq \alpha + x_i = x \leq (1+\delta)^{i-1}\alpha' + x_i \leq (1 + \delta)^i x'.$$
Proof continued

1. If \( x \in P_i \setminus P_{i-1} \implies x = \alpha + x_i \), for some \( \alpha \in P_{i-1} \).
2. By induction, \( \exists \alpha' \in L_{i-1} \) s.t. \( \alpha' \leq \alpha \leq (1 + \delta)^{i-1} \alpha' \).
3. Thus, \( \alpha' + x_i \in E_i \).
4. \( \exists x' \in L_i \) s.t. \( x' \leq \alpha' + x_i \leq (1 + \delta) x' \).
5. Thus,
\[
x' \leq \alpha' + x_i \leq \alpha + x_i = x \leq (1 + \delta)^{i-1} \alpha' + x_i \leq (1 + \delta)^i x'.
\]

1. If \( x \in P_i \setminus P_{i-1} \implies x = \alpha + x_i \), for some \( \alpha \in P_{i-1} \).

2. By induction, \( \exists \alpha' \in L_{i-1} \) s.t. \( \alpha' \leq \alpha \leq (1 + \delta)^{i-1}\alpha' \).

3. Thus, \( \alpha' + x_i \in E_i \).

4. \( \exists x' \in L_i \) s.t. \( x' \leq \alpha' + x_i \leq (1 + \delta)x' \).

5. Thus,
\[ x' \leq \alpha' + x_i \leq \alpha + x_i = x \leq (1 + \delta)^{i-1}\alpha' + x_i \leq (1 + \delta)^i x'. \]
Proof continued

1. If $x \in P_i \setminus P_{i-1} \implies x = \alpha + x_i$, for some $\alpha \in P_{i-1}$.

2. By induction, $\exists \alpha' \in L_{i-1}$ s.t.
   $$\alpha' \leq \alpha \leq (1 + \delta)^{i-1}\alpha'.$$

3. Thus, $\alpha' + x_i \in E_i$.

4. $\exists x' \in L_i$ s.t. $x' \leq \alpha' + x_i \leq (1 + \delta)x'$.

5. Thus,
   $$x' \leq \alpha' + x_i \leq \alpha + x_i = x \leq (1 + \delta)^{i-1}\alpha' + x_i \leq (1 + \delta)^{i-1}(\alpha' + x_i) \leq (1 + \delta)^i x'.$$
Proof continued

1. If $x \in P_i \setminus P_{i-1} \implies x = \alpha + x_i$, for some $\alpha \in P_{i-1}$.

2. By induction, $\exists \alpha' \in L_{i-1}$ s.t. $\alpha' \leq \alpha \leq (1 + \delta)^{i-1} \alpha'$.

3. Thus, $\alpha' + x_i \in E_i$.

4. $\exists x' \in L_i$ s.t. $x' \leq \alpha' + x_i \leq (1 + \delta)x'$.

5. Thus,
   \[ x' \leq \alpha' + x_i \leq \alpha + x_i = x \leq (1+\delta)^{i-1} \alpha' + x_i \leq (1 + \delta)^{i-1}(\alpha' + x_i) \leq (1 + \delta)^i x'. \]
10.1.1.1: Running time
Running time of ApproxSubsetSum

Lemma
For $x \in [0, 1]$, it holds $\exp(x/2) \leq (1 + x)$.

Lemma
For $0 < \delta < 1$, and $x \geq 1$, we have

$$\log_{1+\delta} x \leq \frac{2 \ln x}{\delta} = O\left(\frac{\ln x}{\delta}\right).$$

See notes for a proof of lemmas.
Running time of ApproxSubsetSum

Observation
In a list generated by Trim, for any number $x$, there are no two numbers in the trimmed list between $x$ and $(1 + \delta)x$.

Lemma
$|L_i| = O\left(\frac{n}{\varepsilon} \log n\right)$, for $i = 1, \ldots, n$. 
Running time of ApproxSubsetSum

Proof.

1. $L_{i-1} + x_i \subseteq [x_i, ix_i]$.

2. Trimming $L_{i-1} + x_i$ results in list of size

$$\log_{1+\delta} \frac{ix_i}{x_i} = O\left(\frac{\ln i}{\delta}\right) = O\left(\frac{\ln n}{\delta}\right),$$

3. Now, $\delta = \varepsilon/2n$, and

$$|L_i| \leq |L_{i-1}| + O\left(\frac{\ln n}{\delta}\right) \leq |L_{i-1}| + O\left(\frac{n \ln n}{\varepsilon}\right) = O\left(\frac{n^2 \log n}{\varepsilon}\right).$$
Running time of \textbf{ApproxSubsetSum}

Proof.

1. $L_{i-1} + x_i \subseteq [x_i, ix_i]$.

2. Trimming $L_{i-1} + x_i$ results in list of size

   \[
   \log_{1+\delta} \frac{ix_i}{x_i} = O\left(\frac{\ln i}{\delta}\right) = O\left(\frac{\ln n}{\delta}\right),
   \]

3. Now, $\delta = \epsilon / 2n$, and

   \[
   |L_i| \leq |L_{i-1}| + O\left(\frac{\ln n}{\delta}\right) \leq |L_{i-1}| + O\left(\frac{n \ln n}{\epsilon}\right) \\
   = O\left(\frac{n^2 \log n}{\epsilon}\right)
   \]
Lemma

The running time of \textbf{ApproxSubsetSum} is \( O\left(\frac{n^3}{\epsilon} \log n\right) \).

Proof.

1. Running time of \textbf{ApproxSubsetSum} dominated by total length of \( L_1, \ldots, L_n \).

2. Above lemma implies

\[
\sum_i \left| L_i \right| = O \left( n \times \frac{n^2}{\epsilon} \log n \right) = O \left( \frac{n^3}{\epsilon} \log n \right)
\]

3. \textbf{Trim} runs in time proportional to size of lists.

4. Overall, \( \text{R.T.} = O\left( \frac{n^3}{\epsilon} \log n \right) \).
ApproxSubsetSum

Theorem

ApproxSubsetSum returns $u \leq t$, s.t.

\[
\frac{\gamma_{opt}}{1+\varepsilon} \leq u \leq \gamma_{opt} \leq t,
\]

$\gamma_{opt}$: opt solution.

Running time is $O(\left(\frac{n^3}{\varepsilon}\right) \log n)$.

Proof.

1. Running time from above.
2. $\gamma_{opt} \in P_n$: optimal solution.
3. $\exists z \in L_n$, such that $z \leq \text{opt} \leq (1 + \delta)^n z$
4. $(1 + \delta)^n = (1 + \varepsilon/2n)^n \leq \exp\left(\frac{\varepsilon}{2}\right) \leq 1 + \varepsilon,$
   since $1 + x \leq e^x$ for $x \geq 0$.
5. $\gamma_{opt}/(1+\varepsilon) \leq z \leq \text{opt} \leq t$.
ApproxSubsetSum

**Theorem**

*ApproxSubsetSum* returns $u \leq t$, s.t.

$$\frac{\gamma_{opt}}{1+\varepsilon} \leq u \leq \gamma_{opt} \leq t,$$

$\gamma_{opt}$: opt solution.

*Running time is* $O((n^3/\varepsilon) \log n)$.

**Proof.**

1. Running time from above.

2. $\gamma_{opt} \in P_n$: optimal solution.

3. $\exists z \in L_n$, such that $z \leq \text{opt} \leq (1 + \delta)^n z$

4. $(1 + \delta)^n = (1 + \varepsilon/2n)^n \leq \exp\left(\frac{\varepsilon}{2}\right) \leq 1 + \varepsilon$, since $1 + x \leq e^x$ for $x \geq 0$.

5. $\gamma_{opt}/(1 + \varepsilon) \leq z \leq \text{opt} \leq t$.
ApproxSubsetSum

Theorem

ApproxSubsetSum returns $u \leq t$, s.t.
\[ \frac{\gamma_{\text{opt}}}{1+\varepsilon} \leq u \leq \gamma_{\text{opt}} \leq t, \]

$\gamma_{\text{opt}}$: opt solution.

Running time is $O((n^3/\varepsilon) \log n)$.

Proof.

1. Running time from above.
2. $\gamma_{\text{opt}} \in P_n$: optimal solution.
3. $\exists z \in L_n$, such that $z \leq \text{opt} \leq (1 + \delta)^n z$
4. $(1 + \delta)^n = (1 + \varepsilon/2n)^n \leq \exp\left(\frac{\varepsilon}{2}\right) \leq 1 + \varepsilon$, since $1 + x \leq e^x$ for $x \geq 0$.
5. $\gamma_{\text{opt}}/(1 + \varepsilon) \leq z \leq \text{opt} \leq t$. 
**ApproxSubsetSum**

**Theorem**

*ApproxSubsetSum* returns $u \leq t$, s.t.

\[
\frac{\gamma_{\text{opt}}}{1+\varepsilon} \leq u \leq \gamma_{\text{opt}} \leq t,
\]

$\gamma_{\text{opt}}$: opt solution.

*Running time* is $O((n^3 / \varepsilon) \log n)$.

**Proof.**

1. Running time from above.
2. $\gamma_{\text{opt}} \in P_n$: optimal solution.
3. $\exists z \in L_n$, such that $z \leq \text{opt} \leq (1 + \delta)^n z$
4. $(1 + \delta)^n = (1 + \varepsilon/2n)^n \leq \exp\left(\frac{\varepsilon}{2}\right) \leq 1 + \varepsilon$, since $1 + x \leq e^x$ for $x \geq 0$. 
ApproxSubsetSum

Theorem

**ApproxSubsetSum** returns \( u \leq t \), s.t.
\[
\frac{\gamma_{opt}}{1 + \varepsilon} \leq u \leq \gamma_{opt} \leq t,
\]
\(
\gamma_{opt}: \text{opt solution.}
\)

Running time is \( O\left(\frac{n^3}{\varepsilon} \log n\right) \).

Proof.

1. Running time from above.
2. \( \gamma_{opt} \in P_n \): optimal solution.
3. \( \exists z \in L_n, \text{such that} z \leq \text{opt} \leq (1 + \delta)^n z \)
4. \( (1 + \delta)^n = (1 + \varepsilon / 2n)^n \leq \exp\left(\frac{\varepsilon}{2}\right) \leq 1 + \varepsilon \), since \( 1 + x \leq e^x \) for \( x \geq 0 \).
5. \( \gamma_{opt} / (1 + \varepsilon) \leq z \leq \text{opt} \leq t \).
10.2: Maximal matching
Maximal matching

1. $G = (V, E)$
2. Compute maximal matching...
3. $X \subseteq E$ which is maximal and independent.
4. Maximal = can not improved by adding an edge.
5. Maximum = largest possible set among all possible sets.
6. Computing the maximum is hard then computing maximal solution.
7. Q: Find maximal matching quickly and of large size...
An example of the greedy algorithm...
An example of the greedy algorithm...
An example of the greedy algorithm...
An example of the greedy algorithm...
An example of the greedy algorithm...
An example of the greedy algorithm...
An example of the greedy algorithm...
An example of the greedy algorithm...
An example of the greedy algorithm...
An example of the greedy algorithm...
An example of the greedy algorithm...
An example of the greedy algorithm...
An example of the greedy algorithm...
An example of the greedy algorithm...
An example of the greedy algorithm...
An example of the greedy algorithm...
An example of the greedy algorithm...
An example of the greedy algorithm...
An example of the greedy algorithm...
An example of the greedy algorithm...
Maximal matching: Algorithm

1. Algorithm: Repeatedly pick an arbitrary edge and remove it.
3. Clearly a maximal matching...
4. This is a $2$-approximation to the maximum matching.
5. Because...
6. Every edge in $M$ “kills” two edges of $X$ in the worst case.
Maximal matching: Algorithm

1. Algorithm: Repeatedly pick an arbitrary edge and remove it.
3. Clearly a maximal matching...
4. This is a 2-approximation to the maximum matching.
5. Because...
6. Every edge in $M$ “kills” two edges of $X$ in the worst case.
Maximal matching: Algorithm

1. Algorithm: Repeatedly pick an arbitrary edge and remove it.
3. Clearly a maximal matching...
4. This is a $2$-approximation to the maximum matching.
5. Because...
6. Every edge in $M$ “kills” two edges of $X$ in the worst case.
Maximal matching: Algorithm

1. Algorithm: Repeatedly pick an arbitrary edge and remove it.
3. Clearly a maximal matching...
4. This is a 2-approximation to the maximum matching.
5. Because...
6. Every edge in $M$ “kills” two edges of $X$ in the worst case.
Maximal matching: Algorithm

1. Algorithm: Repeatedly pick an arbitrary edge and remove it.
3. Clearly a maximal matching...
4. This is a 2-approximation to the maximum matching.
5. Because...
6. Every edge in $M$ “kills” two edges of $X$ in the worst case.
Maximal matching: Result

**Theorem**

Given a graph $G$ one can compute in $O(n + m)$ time, a maximal matching with at least $|X|/2$ edges, where $X$ is the size of the maximum (optimal) matching.
10.2.1: Bin packing
Bin packing

Problem definition

**Bin Packing**

**Instance:** \( v \): Bin size. \( S = \{\alpha_1, \ldots, \alpha_n\} \): \( n \) items
\( \alpha_i \): size of \( i \)th item.

**Target:** Find min \# \( B \), and a decomposition \( S_1, \ldots, S_B \) of \( S \), such that \( \forall j \sum_{x \in S_j} \leq v \).

1. \( \bigcup_i S_i = S \) and \( \forall i \neq j \quad S_i \cap S_j = \emptyset \).
2. **NP-Hard** from Partition.
3. **NP-Hard** to approximate within \( 3/2 \).
4. Natural problem...
5. How to approximate?
6. First fit. Have a row of bins, insert items greedily, like...
Bin packing

Problem definition

Bin Packing

Instance: \( v \): Bin size. \( S = \{\alpha_1, \ldots, \alpha_n\} \): \( n \) items
\( \alpha_i \): size of \( i \)th item.

Target: Find min \( \# B \), and a decomposition \( S_1, \ldots, S_B \) of \( S \), such that \( \forall j \sum_{x \in S_j} \leq v \).

1. \( \bigcup_i S_i = S \) and \( \forall i \neq j \ S_i \cap S_j = \emptyset \).
2. \( \textbf{NP-Hard} \) from Partition.
3. \( \textbf{NP-Hard} \) to approximate within \( 3/2 \).
4. Natural problem...
5. How to approximate?
6. First fit: Have a row of bins, insert items greedily, like...
Bin packing

Problem definition

**Bin Packing**

<table>
<thead>
<tr>
<th>Instance: v: Bin size. S = {α₁, ..., αₙ}: n items</th>
</tr>
</thead>
<tbody>
<tr>
<td>αᵢ: size of i-th item.</td>
</tr>
<tr>
<td><strong>Target:</strong> Find min ≠ B, and a decomposition S₁, ..., Sₘ of S, such that ∀j ( \sum_{x \in S_j} \leq v ).</td>
</tr>
</tbody>
</table>

1. \( \bigcup_i S_i = S \) and ∀\( i \neq j \), \( S_i \cap S_j = \emptyset \).
2. **NP-Hard** from Partition.
3. **NP-Hard** to approximate within \( 3/2 \).
4. Natural problem...
5. How to approximate?
6. First fit: Have a row of bins, insert items greedily into the first bin that fits them.
7. First fit decreasing: Sort the elements first...
Bin packing

Problem definition

Bin Packing

Instance: \( v \): Bin size. \( S = \{\alpha_1, \ldots, \alpha_n\} \): \( n \) items
\( \alpha_i \): size of \( i \)th item.

Target: Find \( \min \) \( \# B \), and a decomposition \( S_1, \ldots, S_B \) of \( S \), such that \( \forall j \sum_{x \in S_j} \leq v \).

1. \( \bigcup_i S_i = S \) and \( \forall i \neq j \quad S_i \cap S_j = \emptyset \).
2. \textbf{NP-Hard} from \textbf{Partition}.
3. \textbf{NP-Hard} to approximate within \( 3/2 \).
4. Natural problem...
5. How to approximate?
6. First fit: Have a row of bins, insert items one by one, like
Bin packing

Problem definition

Bin Packing

Instance: $v$: Bin size. $S = \{\alpha_1, \ldots, \alpha_n\}$: $n$ items
$
\alpha_i$: size of $i$th item.

Target: Find $\min \# B$, and a decomposition $S_1, \ldots, S_B$ of $S$, such that $\forall j \sum_{x \in S_j} x \leq v$.

1. $\bigcup_i S_i = S$ and $\forall i \neq j$ $S_i \cap S_j = \emptyset$.
2. **NP-Hard** from **Partition**.
3. **NP-Hard** to approximate within $3/2$.
4. Natural problem...
5. How to approximate?
6. First fit: Have a row of bins, insert items greedily into the first bin that fits them.
Bin packing

Problem definition

Bin Packing

Instance: $v$: Bin size. $S = \{\alpha_1, \ldots, \alpha_n\}$: $n$ items

$\alpha_i$: size of $i$th item.

Target: Find min $\# B$, and a decomposition $S_1, \ldots, S_B$ of $S$, such that $\forall j \sum_{x \in S_j} \leq v$.

1. $\cup_i S_i = S$ and $\forall i \neq j \ S_i \cap S_j = \emptyset$.
2. **NP-Hard** from **Partition**.
3. **NP-Hard** to approximate within $3/2$.
4. Natural problem...
5. How to approximate?

6. First fit: Have a row of bins, insert items greedily into the first bin that fits them.
7. First fit decreasing: Sort the elements first...
Bin packing

Problem definition

**Bin Packing**

**Instance:** \( v \): Bin size. \( S = \{\alpha_1, \ldots, \alpha_n\} \): \( n \) items
\( \alpha_i \): size of \( i \)th item.

**Target:** Find \( \min \# B \), and a decomposition \( S_1, \ldots, S_B \) of \( S \), such that \( \forall j \sum_{x \in S_j} \leq v \).

1. \( \bigcup_i S_i = S \) and \( \forall i \neq j \quad S_i \cap S_j = \emptyset \).
2. **NP-Hard** from **Partition**.
3. **NP-Hard** to approximate within \( 3/2 \).
4. Natural problem...
5. How to approximate?
6. First fit: Have a row of bins, insert items greedily.

30/41112
Bin packing

Problem definition

Bin Packing

Instance: \( v \): Bin size. \( S = \{\alpha_1, \ldots, \alpha_n\} \): \( n \) items
\( \alpha_i \): size of \( i \)th item.

Target: Find min \( \# B \), and a decomposition \( S_1, \ldots, S_B \) of \( S \), such that \( \forall j \sum_{x \in S_j} \leq v \).

1. \( \bigcup_i S_i = S \) and \( \forall i \neq j \quad S_i \cap S_j = \emptyset \).
2. \textbf{NP-Hard} from Partition.
3. \textbf{NP-Hard} to approximate within \( 3/2 \).
4. Natural problem...
5. How to approximate?
6. First fit: Have a row of bins, insert items greedily, like...
Bin packing: First fit

Analysis

Lemma

First fit is a $2$-approximation.

Proof.

Observe that only one bin can have less than $\frac{v}{2}$ content in it...
10.3: Independent set of axis-parallel rectangles
An example

Input
Assume: Open rectangles.

Independent set of rectangles.
An example

Input
Assume: Open rectangles.

Independent set of rectangles.
Given $n$ intervals on the real line, computing the largest independent set of intervals on the real line, can be done in:

(A) $O(n)$ time.
(B) $O(n \log n)$ time.
(C) $O(n^{3/2})$ time.
(D) $O(n^2)$ time.
(E) NP-Hard.
Independent set of rectangles
Algorithm: Divide & Conquer
Independent set of rectangles

Algorithm: Divide & Conquer
Independent set of rectangles

Algorithm: Divide & Conquer
Independent set of rectangles

Algorithm: Divide & Conquer
Independent set of rectangles

Algorithm: Divide & Conquer
Independent set of rectangles

Algorithm: Divide & Conquer
Independent set of rectangles

Algorithm: Divide & Conquer
Independent set of rectangles

Algorithm: Divide & Conquer
Independent set of rectangles

Algorithm: Divide & Conquer

\( \mathcal{R} \): A set of axis parallel rectangles.

**RectIndep***(\( \mathcal{R} \)):

\[
\text{if } |\mathcal{R}| \leq 10 \text{ then}
\]

Solve by brute force

\text{return size of solution}

\( x_M \): Median of right \( x \)-coordinate of rects in \( \mathcal{R} \)

\( \ell \): Vertical line through \( x_M \).

\( \mathcal{R}_M \): Rects of \( \mathcal{R} \) intersecting \( \ell \)

\( \mathcal{R}_L, \mathcal{R}_R \): Rectangles in \( \mathcal{R} \) left/ right of \( \ell \).

\( S_L \leftarrow \text{RectIndep}(\mathcal{R}_L) \)

\( S_R \leftarrow \text{RectIndep}(\mathcal{R}_R) \)

\( S_M \leftarrow \text{compute opt solution for } \mathcal{R}_M \text{ (intervals!)} \)

\text{return max}(S_M, S_L + S_R)
Analysis

1. If $S_M \geq \text{Opt} / (2 \log n)$... done.
2. $\text{Opt}_L + \text{Opt}_R \geq (1 - 1/(2 \log n)) \text{Opt}$.
3. By induction: $S_L \geq \text{Opt}_L/(2 \log(n/2))$ and $S_R \geq \text{Opt}_R/(2 \log(n/2))$.
4. $S_L + S_R \geq \frac{(1 - 1/(2 \log n)) \text{Opt}}{2 \log(n/2)}$
5. $\frac{1}{2 \log(n/2)}$ =
   \[
   \frac{1}{2 \log n - 2} - \frac{1}{(2 \log n)(2 \log n - 2)}
   \geq \frac{2 \log n - 1}{(2 \log n)(2 \log n - 2)} \geq
   \frac{1}{(2 \log n)(2 \log n - 2)} \geq \frac{1}{2 \log n}.
   \]
Analysis

1. If $S_M \geq \text{Opt} / (2 \lg n)$... done.
2. $\text{Opt}_L + \text{Opt}_R \geq (1 - 1/(2 \lg n))\text{Opt}$.
3. By induction: $S_L \geq \text{Opt}_L / (2 \lg (n/2))$ and $S_R \geq \text{Opt}_R / (2 \lg (n/2))$.
4. $S_L + S_R \geq \frac{(1 - 1/(2 \lg n))\text{Opt}}{2 \lg (n/2)}$

5. \[
\frac{1}{2 \lg (n/2)} = \frac{1}{2 \lg n - 2} - \frac{(2 \lg n)(2 \lg n - 2)}{2 \lg n - 1} \geq \frac{1}{(2 \lg n)(2 \lg n - 2)} \geq \frac{1}{2 \lg n}.
\]
Analysis

1. If $S_M \geq \text{Opt} / (2 \log n)$... done.
2. $\text{Opt}_L + \text{Opt}_R \geq (1 - 1 / (2 \log n)) \text{Opt}$.
3. By induction: $S_L \geq \text{Opt}_L / (2 \log (n/2))$ and $S_R \geq \text{Opt}_R / (2 \log (n/2))$.
4. $S_L + S_R \geq \frac{(1 - 1 / (2 \log n)) \text{Opt}}{2 \log (n/2)}$
5. $\frac{1}{2 \log (n/2)} = \frac{1}{2 \log n - 2} - \frac{(2 \log n)(2 \log n - 2)}{2 \log n - 1} \geq \frac{(2 \log n)(2 \log n - 2)}{2 \log n - 2} \geq \frac{1}{2 \log n}$. 
Analysis

1. If $S_M \geq \text{Opt}/(2 \lg n)$... done.
2. $\text{Opt}_L + \text{Opt}_R \geq (1 - 1/(2 \lg n))\text{Opt}$.
3. By induction: $S_L \geq \text{Opt}_L/(2 \lg(n/2))$ and $S_R \geq \text{Opt}_R/(2 \lg(n/2))$.
4. $S_L + S_R \geq \frac{(1-1/(2 \lg n))\text{Opt}}{2 \lg(n/2)}$
5. $\frac{1}{2 \lg(n/2)} = 1 - \frac{2 \lg n - 2}{2 \lg n - 1}$
   $\geq \frac{(2 \lg n)(2 \lg n - 2)}{(2 \lg n)(2 \lg n - 2)} \geq \frac{1}{2 \lg n}$.  
7. Algorithm is $2 \lg n$ approximation.
Analysis

1. If $S_M \geq \text{Opt}/(2 \log n)$... done.
2. $\text{Opt}_L + \text{Opt}_R \geq (1 - 1/(2 \log n))\text{Opt}$.
3. By induction: $S_L \geq \text{Opt}_L/(2 \log(n/2))$ and $S_R \geq \text{Opt}_R/(2 \log(n/2))$.
4. $S_L + S_R \geq \frac{(1 - 1/(2 \log n))\text{Opt}}{2 \log(n/2)}$
5. $\frac{2 \log(n/2)}{2 \log n - 2} \geq \frac{2 \log n - 1}{(2 \log n)(2 \log n - 2)} \geq \frac{1}{2 \log n}$. 

Conclusion: If $S_M \leq \text{Opt}/(2 \log n)$, then $S_L + S_R \geq \text{Opt}/(2 \log n)$.

Algorithm is $2 \log n$ approximation.
Analysis

1. If $S_M \geq \text{Opt}/(2 \lg n)$... done.
2. $\text{Opt}_L + \text{Opt}_R \geq (1 - 1/(2 \lg n))\text{Opt}$.
3. By induction: $S_L \geq \text{Opt}_L/(2 \lg(n/2))$ and $S_R \geq \text{Opt}_R/(2 \lg(n/2))$.
4. $S_L + S_R \geq (1 - 1/(2 \lg n))\frac{\text{Opt}}{2 \lg(n/2)}$
5. $\frac{1}{2 \lg(n/2)} = 1 \frac{1}{2 \lg n - 2} - \frac{(2 \lg n)(2 \lg n - 2)}{2 \lg n - 1} \geq \frac{1}{(2 \lg n)(2 \lg n - 2)} \geq \frac{1}{2 \lg n}$. 
Analysis

1. If $S_M \geq \text{Opt} / (2 \lg n)$... done.
2. $\text{Opt}_L + \text{Opt}_R \geq (1 - 1/(2 \lg n)) \text{Opt}$.
3. By induction: $S_L \geq \text{Opt}_L / (2 \lg (n/2))$ and $S_R \geq \text{Opt}_R / (2 \lg (n/2))$.
4. $S_L + S_R \geq \frac{(1 - 1/(2 \lg n)) \text{Opt}}{2 \lg (n/2)}$

$$\frac{2 \lg (n/2)}{1 - \frac{2 \lg n - 2}{2 \lg n}} \geq \frac{2 \lg n - 1}{(2 \lg n)(2 \lg n - 2)} \geq \frac{1}{2 \lg n}.$$
Analysis

1. If $S_M \geq \text{Opt}/(2 \lg n)$... done.
2. $\text{Opt}_L + \text{Opt}_R \geq (1 - 1/(2 \lg n))\text{Opt}$.
3. By induction: $S_L \geq \text{Opt}_L/(2 \lg(n/2))$ and $S_R \geq \text{Opt}_R/(2 \lg(n/2))$.
4. $S_L + S_R \geq \frac{(1-1/(2 \lg n))\text{Opt}}{2 \lg(n/2)}$

$$\frac{1}{2 \lg(n/2)} = \frac{1}{2 \lg n - 2} - \frac{(2 \lg n)(2 \lg n - 2)}{2 \lg n - 1} \geq \frac{2 \lg n - 1}{(2 \lg n)(2 \lg n - 2)} \geq \frac{1}{2 \lg n}.$$
Analysis

1. If \( S_M \geq \frac{\text{Opt}}{(2 \lg n)} \) ... done.
2. \( \text{Opt}_L + \text{Opt}_R \geq (1 - 1/(2 \lg n)) \text{Opt} \).
3. By induction: \( S_L \geq \frac{\text{Opt}_L}{(2 \lg (n/2))} \) and \( S_R \geq \frac{\text{Opt}_R}{(2 \lg (n/2))} \).
4. \( S_L + S_R \geq \frac{(1-1/(2 \lg n)) \text{Opt}}{2 \lg (n/2)} \).
5. \( \frac{1}{2 \lg (n/2)} - \frac{1}{2 \lg n - 2} \)

\[
\begin{align*}
&\geq \frac{2 \lg n - 1}{(2 \lg n)(2 \lg n - 2)} \geq \\
&\geq \frac{1}{2 \lg n - 2} \geq \frac{1}{2 \lg n}.
\end{align*}
\]
Notes