ADV. ALGO Test study

🧭 1. Complexity & Reduction Fundamentals

1.1 Complexity Classes

Symbol	Meaning
P	Problems solvable in polynomial time
NP	Problems whose solutions can be *verified* in polynomial time
NP-hard	At least as hard as every problem in NP (no known polytime algorithm)
NP-complete	Problems that are both NP-hard and in NP

1.3 Approximation Definitions

- For **minimization problems**:

$$\frac{ALG(I)}{OPT(I)} \leq \rho$$

- For **maximization problems**:

$$\frac{OPT(I)}{ALG(I)} \leq \rho$$

where $\rho \geq 1$ is the **approximation factor**.

1.4 General proof pattern (minimization)

1. **Lower bound:** derive something $LB \leq OPT$.
2. **Algorithm bound:** prove $ALG \leq f(LB)$.
3. Combine: $ALG \leq \rho \cdot OPT$.

1.5 Greedy proof skeleton

> “Prove feasibility, then prove an upper bound on the cost, compare with a lower bound on OPT → conclude ratio ρ.”

⚙️ 2. Greedy Algorithms

2.1 Load Balancing Problem

Setup:

- m identical machines
- Jobs $J = 1, 2, ..., n$ with processing times t_j
- Each job assigned to one machine

Goal: minimize the **makespan** (maximum load).

Greedy Algorithm (List Scheduling)

Assign each job to the machine with the **current smallest load**.

Notation

- L_i: load of machine i
- $L = \max_i L_i$: makespan of algorithm
- L^*: optimal makespan

Lower bounds for L^*

Every feasible schedule must satisfy:

$$L^* \geq \max_j t_j \quad \text{and} \quad L^* \geq \frac{1}{m}\sum_j t_j$$

(Reason: one job is the largest task, and total work divided among m machines.)

Greedy bound

When the last job k is scheduled:

$$L \leq t_k + \frac{1}{m}\sum_j t_j$$

Using both lower bounds:

$$L \leq 2L^*$$

✅ **Conclusion:** the greedy load balancing algorithm is a **2-approximation**.

2.2 Greedy for Set Cover (theoretical)

Problem:
Cover all elements of universe U with the fewest possible subsets S_i.

Greedy algorithm:
At each step, pick the set that covers the largest number of uncovered elements **per unit cost**.

Approximation Guarantee

Greedy Set Cover is an H_n**-approximation**, where

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \leq \ln n + 1.$$

Key idea (potential argument)

At each iteration, the greedy algorithm covers at least a **1/k fraction** of remaining elements, leading to a logarithmic number of iterations.

2.3 Greedy Structure Template

Step	Description
1	Show that algorithm produces a **feasible solution**
2	Find a **lower bound** for OPT
3	Relate algorithm's value to OPT
4	Derive constant or logarithmic ratio

🧮 3. LP Relaxation and Rounding

3.1 LP Relaxation definition

An **LP relaxation** of an Integer Program (IP) replaces the integrality constraint $x_i \in 0, 1$ by

$$0 \leq x_i \leq 1.$$

3.2 Property of LP relaxations

$$OPT_{LP} \leq OPT_{IP}$$

(because relaxing constraints can only decrease the minimum).

3.3 General LP-Rounding Template

Step	Description
1	Formulate the Integer Program (IP)
2	Relax integrality constraints → get LP
3	Solve LP to obtain fractional x^*
4	Round x^* to integer x using a threshold $t = 1/f$
5	Prove feasibility (pigeonhole/averaging argument)
6	Prove bound: $x_i \le f \cdot x_i^*$
7	Conclude cost$(x) \le f \cdot OPT$

3.4 Example: Vertex Cover LP Rounding

LP:

$$\min \sum_v w_v x_v \text{ s.t. } \quad x_u + x_v \ge 1 \quad \forall (u,v) \in E, x_v \ge 0.$$

Rounding rule:

$$x_v = \begin{cases} 1, & x_v^* \ge 1/2, \ 0, \ \text{otherwise.} \end{cases}$$

Feasibility: each edge has at least one endpoint $\ge 1/2$.
Cost bound: $x_v \le 2x_v^*$ → 2-approximation.

3.5 Example: Set Cover LP Rounding

LP:

$$\min \sum_S c_S x_S \text{ s.t. } \quad \sum_{S: e \in S} x_S \ge 1, \quad \forall e, x_S \ge 0.$$

Rounding rule:
Pick all sets S with $x_S^* \geq 1/f$, where $f = \max_e |S : e \in S|$.

Feasibility: at least one set per element has $x_S^* \geq 1/f$.
Cost bound: cost$(x) \leq f\cdot$cost$(x^*) \leq f \cdot OPT$.

✅ f**-approximation**

3.6 Example: Dominating Set LP Rounding

Each constraint involves at most $\Delta + 1$ vertices (neighbors + itself).

Threshold:

$$t = \frac{1}{\Delta + 1}$$

Feasibility: if all were smaller than $1/(\Delta + 1)$, the sum < 1 — contradiction.
Cost bound: $x_v \leq (\Delta + 1)y_v^* \rightarrow (\Delta + 1)$-approximation.

3.7 Rounding Summary Table

Problem	Constraint size	Threshold	Approx. Ratio
Vertex Cover	2	1/2	2
Set Cover	f	1/f	f
Dominating Set	Δ+1	1/(Δ+1)	Δ+1

🔁 4. Primal–Dual Method

4.1 Weak Duality Theorem

For any feasible primal (P) and dual (D):

$$\text{value(D)} \leq \text{value(P)}.$$

This allows proving approximation ratios by comparing primal and dual costs.

4.2 Complementary Slackness (qualitative)

- If a primal constraint is *tight*, the corresponding dual variable can be > 0.
- If a dual constraint is *tight*, the corresponding primal variable can be > 0.

Used in designing primal–dual algorithms.

4.3 Primal–Dual Algorithm Structure

Step	Description
1	Write LP and its dual
2	Start with all dual vars = 0
3	Repeatedly increase some dual vars until a constraint becomes tight
4	Add corresponding primal variable (edge/vertex) to solution
5	Stop when all primal constraints are satisfied
6	Use degree/counting argument to show bounded ratio

4.4 Example 1 — Vertex Cover (Weighted)

Primal (LP):

$$\min \sum_v w_v x_v \quad \text{s.t. } x_u + x_v \geq 1, \forall (u,v) \in E, \quad x_v \geq 0.$$

Dual:

$$\max \sum_{(u,v)\in E} y_{uv} \quad \text{s.t.} \sum_{(u,v)\in E(i)} y_{uv} \le w_i, \forall i, \quad y_{uv} \ge 0.$$

Algorithm:

1. Initialize $y_{uv} = 0$, $C = \emptyset$.
2. Raise y_{uv} on uncovered edges until some vertex v becomes **tight** ($\sum y_{uv} = w_v$).
3. Add v to C.
4. Stop when all edges are covered.

Analysis:

- Each chosen vertex tight → cost$(C) = \sum w_v = \sum_v w_v \le 2\sum_{(u,v)} y_{uv} \le 2OPT$.
  ✅ **2-approximation.**

4.5 Example 2 — Edge Cover (Weighted)

Primal (P):

$$\min \sum_{e\in E} w_e x_e \quad \text{s.t.} \sum_{e\in\delta(v)} x_e \ge 1, \forall v, \quad x_e \ge 0.$$

Dual (D):

$$\max \sum_{v\in V} y_v \quad \text{s.t. } y_u + y_v \le w_{uv}, \forall (u,v) \in E, \quad y_v \ge 0.$$

Algorithm

1. Initialize $y_v = 0$, $C = \emptyset$, all vertices uncovered.
2. While some vertex v uncovered:
   - Increase y_v until some incident edge (v,u) becomes **tight** ($y_v + y_u = w_{uv}$).
   - Add (v,u) to C and mark v, u covered.

Feasibility

Each step covers both v and u.
Loop stops when all vertices are covered → valid edge cover.

Approximation ratio

Every chosen edge (u, v) is tight:

$$w_{uv} = y_u + y_v$$

Algorithm cost:

$$ALG = \sum_{(u,v)\in C} w_{uv} = \sum_{v} \deg_C(v) y_v$$

Each vertex appears in ≤2 chosen edges ⇒ $\deg_C(v) \le 2$:

$$ALG \le 2\sum_{v} y_v \le 2OPT$$

✅ **2-approximation**

4.6 General Primal–Dual Pattern

Step	Concept	Example
Raise duals	Increase “prices” on uncovered requirements	uncovered vertex in Edge Cover
Tight constraints	When equality reached, add primal variable	edge (u, v) becomes tight
Stop condition	All primal constraints satisfied	every vertex covered
Approximation	Counting argument on degrees	each vertex counted ≤2

🧠 5. Approximation Schemes

Type	Definition	Example
Constant-factor	Fixed ρ (e.g., 2, f)	Vertex Cover (2), Set Cover (f)
Logarithmic	Ratio grows with input size	Set Cover $O(\log n)$
PTAS	For every $\varepsilon > 0$, polytime for fixed ε	Knapsack
FPTAS	Polytime in both input size and $1/\varepsilon$	Bounded Knapsack

Key properties

- **PTAS:** time = $poly(n)$ for fixed ε.
- **FPTAS:** time = $poly(n, 1/\varepsilon)$.
- **Weak duality** always underlies primal–dual correctness.
- **Complementary slackness** guides when to stop raising duals.

Reduction

- A problem X is reducible to a problem Y if an algorithm for solving Y can be used to solve X.

Notation:

X ≤P Y - means "X is polynomial-time reducible to Y".

iF X <=P Y, then Y is at least as hard as X.

We want to show:

$$\text{HAM-CYCLE} \leq_P \text{TSP}.$$

That means:

"If we can solve TSP, we can also solve HAM-CYCLE."

The Perfect Solution (L*)

- Must be at least the size of the biggest single job.
- Must be at least the average load across all machines.

Aproximization ratio

Problem Type	Ratio Definition	Goal of Proof
Minimization (e.g., Load Balancing, Vertex Cover)	V / V* is less than or equal to a constant rho (rho) [1].	Prove V is less than or equal to rho times V* [2].
Maximization (e.g., Subset-Sum, Knapsack)	V* / V is less than or equal to a constant rho [3].	Prove V* is less than or equal to rho times V [3].

The challenge is proving this relationship *without knowing* V* [4]. We do this by finding lower bounds for V* and upper bounds for V [2].

The core goal is to prove that the value of the solution found by an algorithm (V) is "close" to the value of the optimal solution (V*).

Standard Strategy for Minimization Problems:

1. Find a Lower Bound for V*: Identify properties that V* must satisfy, such as the maximum job size or the average load.
   - Example (Load Balancing): V* is greater than or equal to the total load divided by the number of machines (the average load).
2. Find an Upper Bound for V: Analyze the greedy algorithm's structure to prove V is bounded by V* plus some extra terms.
3. Combine the Bounds: Use the lower bounds for V* to relate the "extra terms" in the V inequality back to V* itself.

Greedy

Load Balancing - decision NP-complete problem

- Problem: Assign n jobs to m machines to minimize the maximum load on any machine.

Input: m machines, n jobs. Each job j has a processing time t_j.

Constraints:

- Each job runs contiguously on one machine.
- Each machine processes one job at a time.

Notation:

- Ji - set of jobs assigned to machine i.
- Li - load on machine i = sum of processing times of jobs in Ji.
- L = max(Li) - makespan (maximum load across all machines).

Two always-true lower bounds for the optimal makespan L*:

1. L* ≥ max(t_j) for all jobs j (the makespan must be at least as large as the longest job).
2. L* ≥ total_load / m (the makespan must be at least the average load per machine).

- Goal: Minimize the makespan L.

Ex. 1 CONJUNTO 1

Convert to an approximation ratio using the given totals

Divide both sides by L^* (note $L^* > 0$):

$$\frac{L}{L^*} \le 1 + \frac{50}{L^*}.$$

But we already proved $L^* \ge 300$. Hence

$$\frac{L}{L^*} \le 1 + \frac{50}{300} = 1 + \frac{1}{6} = 1.166\overline{6} < 1.17.$$

Finally, recall the average load $A = \frac{1}{10} \sum t_j \le L^*$ (OPT can never be **below** the average). So

$$L < 1.17 \cdot L^* \le 1.17 \cdot A.$$

Conclusion: On these restricted instances, `greedyLB1`'s makespan is **always less than 17% above the average load**.

> Exam checklist to reproduce:
>
> 1. **"State LB: $L^* \ge \max t_j$ and $L^* \ge \frac{1}{10} \sum t_j$."**
> 2. **"State List Scheduling fact: $L \le L^* + t_k$."**
> 3. **"Use $t_k \le 50$ and $L^* \ge 300$ to get ratio < 1.17."**
> 4. **"Relate to average: $A \le L^* \Rightarrow L < 1.17\,A$."**

Vertex Cover Greedy (ex 2 CONJUNTO 1)

(a) greedyVC1

- **Algorithm:** For each uncovered edge (u,v), add **one endpoint** (e.g. u) to the cover.
- **Problem:** Can pick many unnecessary vertices depending on tie-breaking or order.
- **Counterexample:**
  Star graph — center C connected to n leaves.
  - OPT = {C}, size = 1
  - Algorithm may pick all leaves → size = n
  - Ratio = n / 1 → unbounded
- ✅ **Conclusion:** Not a ρ-approximation for any constant ρ (ratio unbounded).
- greedyVC1 is not a ρ-approximation algorithm for any constant ρ, because in some graphs (like a star), it can produce arbitrarily worse results than optimal.

(b) greedyVC2

```
greedyVC2((V,E)):
    cover = ∅
    for each v in V: inCover[v] = false

    for each (v,w) in E:
        if not inCover[v] and not inCover[w]:
            cover.add(v);  inCover[v] = true
            cover.add(w);  inCover[w] = true

    return cover
```

- **Algorithm:** For each uncovered edge (u,v), add **both endpoints** u and v to the cover.
- **Feasibility:** Always covers all edges (any uncovered edge triggers both endpoints).
- **Key idea:**
  Let **M** = set of edges that caused both endpoints to be added.
  - M is a **matching** (no shared vertices).
  - Any vertex cover must include ≥ 1 vertex per edge in M → |OPT| ≥ |M|.
  - Algorithm adds 2 vertices per edge in M → |ALG| = 2|M|.
    ⇒ |ALG| ≤ 2 × |OPT|.
- ✅ **Conclusion:** greedyVC2 is a **2-approximation** algorithm.

LP Rounding (Bending the rules)

- Hard Problem (IP - Integer Programming) - Choices must be 0 or 1 (eg. vertex is not/is in the cover)
- Easier Problem(LP - Linear Programming) - Choices can be any value in a range (eg. vertex can be 0.2 in the cover)

We solve the LP, then round the values to get an approximate solution to the IP.

Step:

1. Formulate the problem as a precise Integer Program (IP)
2. Relax into a Linear Program (LP) by allowing variables to take on fractional values.
3. Solve efficiently to find the optimal fractional solution.
4. Round the fractional solution to get an approximate solution to the original IP.

Goal	What you must show	Why it's true
1️⃣ Feasibility	The rounded solution covers everything	Because LP constraints guarantee some variable in each constraint was large enough (≥ threshold)
2️⃣ Approximation factor	Cost of rounded ≤ ρ × OPT	Because each rounded variable ≤ ρ × its fractional value, and fractional OPT ≤ true OPT

Symbol	Meaning
(x_i)	decision variable for item i
(x_i^*)	fractional value from the LP
(x'_i)	rounded (integer) value
(w_i)	weight / cost of item i
(OPT)	optimal true (integer) cost
(w(x))	total cost = $\sum w_i x_i$
(ρ)	approximation ratio (2 for VC, f for SC)

Problem 1 — Maximum Subset-Sum (MSS) CONJUNTO 2

You're given positive integers $P = \{x_1, \dots, x_n\}$ and a capacity M (assume each $x_i \le M$). Goal: pick a subset with **maximum** sum $\le M$. adAlg-problems02

1(a) `greedySS1` — "scan once, add if it fits"

1(a) `greedySS1` — "scan once, add if it fits"

text Copy code

```text
X = Ø ; s = 0
for i=1..n:
  if s + x_i ≤ M:
     add x_i ; s += x_i
return X
```

Claim: `greedySS1` is **not** a ρ-approximation for any constant ρ. (So: no constant-factor guarantee.) adAlg-problems02

Adversarial family (order matters):
For any $M \ge 2$, take the instance with order $[1, M, M, \dots]$.

- `greedySS1` takes the first item 1 $(1 \le M)$; now $s = 1$.
- Next item is M: $s + M = 1 + M > M$ → it **can't** be added. All subsequent M's also can't be added.
- So the algorithm returns value $v = 1$.
- OPT returns the single item $\{M\}$ with value $v^* = M$.

Approximation ratio (maximization definition) is $\frac{v^*}{v} = \frac{M}{1} = M$, which can be **arbitrarily large** as M grows. Hence **no constant** ρ can bound v^*/v. ✅

1(b) `greedySS2` — "stop at first overflow, compare with the culprit"

1(b) `greedySS2` — "stop at first overflow, compare with the culprit"

text Copy code

```text
X = Ø ; s = 0
for i=1..n:
  if s + x_i ≤ M: add x_i ; s += x_i
  else:
     if s ≥ x_i: return X
     else:       return {x_i}
return X
```

Claim: `greedySS2` is a **2-approximation**. adAlg-problems02

Key idea (one-line): Let k be the **first** index where $s + x_k > M$. Then
$M < s + x_k \Rightarrow \max\{s, x_k\} \ge \frac{s + x_k}{2} > \frac{M}{2}$.
`greedySS2` returns exactly $\max\{s, x_k\}$, so its value $v > M/2$. The optimal value $v^* \le M$. Therefore

$$\frac{v^*}{v} \le \frac{M}{M/2} = 2.$$

Edge case: if no overflow ever occurs, the algorithm takes **all** items (total $\le M$), which is optimal. ✅

Why this guarantees 2-approx (the one-line insight)

At the first overflow,

$$s + x_k > M \quad \Rightarrow \quad \max\{s, x_k\} \geq \frac{s + x_k}{2} > \frac{M}{2}.$$

Picture it: draw two bars of lengths s and x_k.
Their combined length already **passes** M.
So **one** of them must be **longer than** $M/2$.
That "longer bar" is exactly what the algorithm returns.

Since any feasible solution is $\leq M$, the optimal value $v^* \leq M$.
Your value $v = \max\{s, x_k\} > M/2$.
Therefore

$$\frac{v^*}{v} \leq \frac{M}{M/2} = 2.$$

Problem 2 — Packing containers into trucks of capacity (C)

We must minimize the number of trucks. Greedy rule from the sheet: fill a truck with items w_1, w_2, ... **in order** until the next item would overflow C; dispatch that truck; repeat with a fresh truck. All w_i, C are positive integers and w_i <= C.

2(a) Show the greedy may be suboptimal

Counterexample: (C=10), items (in order): (6,6,4,4).

- Greedy:
  - Truck1: takes 6; next 6 would overflow → dispatch → load = 6.
  - Truck2: takes 6; next 4 would overflow → dispatch → load = 6.
  - Truck3: takes 4; next 4 fits? yes → Truck3=8 → dispatch.
    Total trucks = 3.
- OPT: pack (6+4) and (6+4) → **2 trucks**.

So greedy isn't optimal.

*(can also use (6,6,4,4) to show it can be **much** worse on longer sequences.)*

2(b) Prove the greedy is a **2-approximation**

we need to prove A ≤ 2 OPT.

load(2j−1)+load(2j)>(C−x)+x=C.

So every complete pair carries more than C in total.

Let:

- A = #trucks used by the greedy,
- T^* = optimum #trucks,
- $W = \sum_i w_i$ = total weight.

Lower bound on OPT: $T^* \geq \left\lceil \frac{W}{C} \right\rceil$. (You must carry W weight with capacity C per truck.)

Pairing trick (hint from sheet): Consider trucks in the greedy solution **by consecutive pairs:** $(\text{Truck } 1, \text{Truck } 2), (\text{Truck } 3, \text{Truck } 4), \ldots$

- Every truck except possibly the last is **closed** exactly when the next item doesn't fit. That means: if the first truck of a pair is closed just before placing some item x, then its load $> C - x$. When we include the second truck's load (which **does** take x), **the pair's total load** $> \ C$.
- Therefore, each full pair carries **more than** C.

Now split by parity of A:

- **If A is even:** there are $A/2$ full pairs, so
  $W > (A/2)\,C \ \Rightarrow\ A < 2\,\frac{W}{C}$.
  Hence $A \leq 2 \left\lceil \frac{W}{C} \right\rceil \leq 2T^*$.
- **If A is odd:** there are $\lfloor A/2 \rfloor$ full pairs $> C$ each, plus a last truck with load > 0.
  Thus $W > \lfloor A/2 \rfloor C \Rightarrow A \leq 2\,\frac{W}{C} + 1$.
  Using integers/$\lceil \cdot \rceil$, you still get $A \leq 2 \left\lceil \frac{W}{C} \right\rceil \leq 2T^*$.

So in all cases, the greedy uses at most **twice** the optimal number of trucks. ✅ adAlg-problems02

Case 1 — A is even

There are exactly $A/2$ full pairs.
Each pair carries $> C$.
So the total weight

$$W \ > \ \frac{A}{2}\,C.$$

Rearrange:

$$A \ < \ 2\,\frac{W}{C}.$$

Use the lower bound on OPT:

$$\frac{W}{C} \ \leq \ \left\lceil \frac{W}{C} \right\rceil \ \leq \ \mathrm{OPT},$$

thus

$$A \ \leq \ 2 \left\lceil \frac{W}{C} \right\rceil \ \leq \ 2\,\mathrm{OPT}.$$

(We moved from "<" to "≤" cleanly by integer rounding; that's fine for an approximation bound.)

Case 2 — A is odd

There are $\lfloor A/2 \rfloor$ full pairs, plus one last, **possibly light**, unpaired truck.

- Each of the $\lfloor A/2 \rfloor$ pairs carries $> C$.
- The last truck carries > 0 (otherwise we wouldn't have opened it).

Hence

$$W > \left\lfloor \frac{A}{2} \right\rfloor C.$$

That implies

$$\left\lfloor \frac{A}{2} \right\rfloor < \frac{W}{C} \implies A \leq 2\frac{W}{C} + 1.$$

Again compare to OPT:

$$A \leq 2 \left\lceil \frac{W}{C} \right\rceil \leq 2\,\mathrm{OPT}.$$

So **in both cases** (even or odd A) we conclude $A \leq 2\,\mathrm{OPT}$. ✅

CONJUNTO 3 - Minimum Dominating Set with LP Rounding

Problem: Minimum Dominating Set (MDS)

A **dominating set** $D \subseteq V$ satisfies: every vertex is either in D or has a neighbor in D.
Equivalently, for each $u \in V$, at least one vertex in its **closed neighborhood** $N[u] = \{u\} \cup N(u)$ is chosen. adAlg-problems03

Let Δ be the **maximum degree** of the graph.