

# Self-Learning1



Can you teach me radix sort?

Explain with an example, step by step.  
No code, only sum solving

Sure! Let me teach you **Radix Sort** in a very simple, step-by-step way using a **number-based example**. No code—just solving by hand like in a classroom.

## What is Radix Sort?

Radix Sort is a **non-comparative** sorting algorithm. It sorts numbers **digit by digit**, starting from the **least significant digit (LSD)** to the **most significant digit (MSD)**.

It is often used to sort numbers (or strings) when all values have **the same number of digits** (or can be padded with zeros).

## Example:

Let's sort the following list of numbers:

170, 45, 75, 90, 802, 24, 2, 66

## Step 1: Pad numbers to have equal digits (if needed)

We find the **maximum number of digits**:

`802` has 3 digits, so we pad all numbers to have 3 digits:

```
170 → 170
045 → 045
075 → 075
090 → 090
802 → 802
024 → 024
002 → 002
066 → 066
```

## Step 2: Sort by the rightmost digit (Units place)

We group the numbers based on the **last digit**:

- 0 → 170, 090
- 2 → 802, 002
- 4 → 024
- 5 → 045, 075
- 6 → 066
- Others: none

**After sorting by units digit:**

```
170, 090, 802, 002, 024, 045, 075, 066
```

## Step 3: Sort by the middle digit (Tens place)

Now look at the **middle digit**:

- 0 → 002
- 2 → 024

- 4 → 045
- 6 → 066
- 7 → 075
- 8 → 802
- 9 → 090
- Others → 170

#### Grouped by tens digit:

- 0 → 002
- 2 → 024
- 4 → 045
- 6 → 066
- 7 → 075
- 8 → 802
- 9 → 090
- 7 → 170

#### After sorting by tens digit:

```
002, 024, 045, 066, 075, 802, 090, 170
```

### Step 4: Sort by the leftmost digit (Hundreds place)

Now look at the **hundreds digit**:

- 0 → 002, 024, 045, 066, 075, 090
- 1 → 170

- $8 \rightarrow 802$

#### Grouped by hundreds digit:

- $0 \rightarrow 002, 024, 045, 066, 075, 090$
- $1 \rightarrow 170$
- $8 \rightarrow 802$

#### Final sorted list:

```
002, 024, 045, 066, 075, 090, 170, 802
```

(Remove leading zeroes if needed)

```
2, 24, 45, 66, 75, 90, 170, 802
```

---

### That's Radix Sort!

We sorted the numbers by **each digit**, from **right to left** (LSD to MSD), and finally got the sorted list.

Want to try one small list yourself next?



Now teach me kruskals algorithm, like i know it already give me a short revision

Perfect! Here's a **quick revision** of **Kruskal's Algorithm** like you're already familiar with it:

---

### Kruskal's Algorithm – Quick Summary

- It's a **Greedy Algorithm** to find a **Minimum Spanning Tree (MST)** of a connected, weighted graph.

- It **sorts all edges by weight** and keeps adding the **smallest edge** that doesn't form a **cycle** until all vertices are connected.
- 

## Key Steps

1. **Sort all edges** in **ascending** order of weight.
  2. Initialize MST as **empty**.
  3. For each edge in sorted list:
    - If adding the edge doesn't form a **cycle** (check using **Disjoint Set / Union-Find**), **include** it in MST.
  4. Repeat until you have **(V - 1) edges** in MST.
- 

## Example

**Vertices:** A, B, C, D

**Edges:**

- A-B (1)
  - B-C (4)
  - A-C (3)
  - C-D (2)
  - B-D (5)
- 

**Sorted Edges:**

1. A-B (1)
2. C-D (2)
3. A-C (3)
4. B-C (4)
5. B-D (5)

---

**MST Construction:**

- A-B (1) → added
- C-D (2) → added
- A-C (3) → added (no cycle yet)
- B-C (4) → skipped (forms cycle)
- B-D (5) → skipped (already connected)

**MST Edges:** A-B, C-D, A-C

**Total Weight:**  $1 + 2 + 3 = 6$

---

That's it. Keep Union-Find handy to detect cycles quickly.

Want me to test you with a small custom graph?

0/1 knapsack problem

- using greedy
- using branch and bound

(B&B is for  
minimization  
problems)

0/1 Knapsack (0 - Absent, 1 - Present)

Object	ob <sub>1</sub>	ob <sub>2</sub>	ob <sub>3</sub>
Weight	2	4	8
Profit	20	25	60

Knapsack Capacity (M) = 12

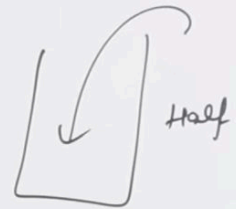
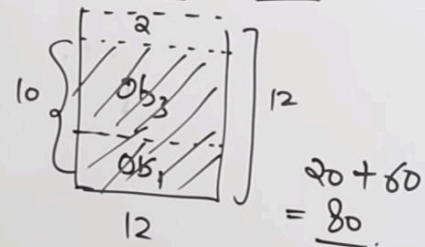




0/1 Knapsack (0 - Absent, 1 - Present)

Object	ob <sub>1</sub>	ob <sub>2</sub>	ob <sub>3</sub>	
Weight	2	4	8	
Profit	20	25	60	60/8

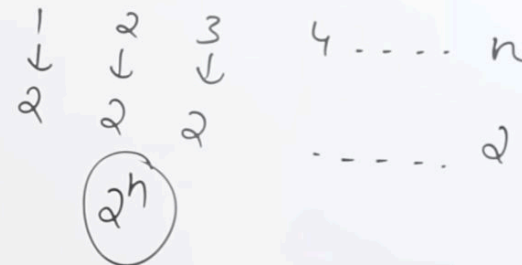
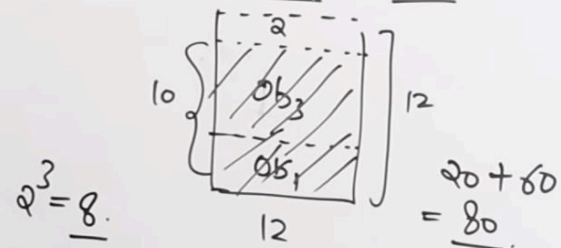
Knapsack Capacity (M) = 12  
P/w 10  $\frac{25}{4}$  6.2 7.5



0/1 Knapsack (0 - Absent, 1 - Present)

Object	ob <sub>1</sub>	ob <sub>2</sub>	ob <sub>3</sub>	
Weight	2	4	8	
Profit	20	25	60	60/8

Knapsack Capacity (M) = 12  
P/W 10 25/4 6.2 7.5.



SUBSCRIBED



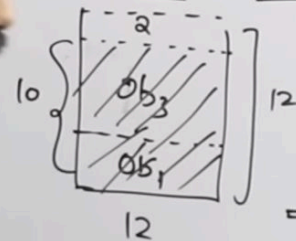
SUBSCRIBE

0/1 Knapsack (0 - Absent, 1 - Present)

Object	ob <sub>1</sub>	ob <sub>2</sub>	ob <sub>3</sub>
	2	4	8
	25	60	60/8

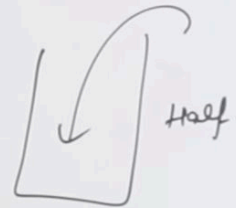
Kn Capacity (M) = 12

25/4 6.2 7.5



$$2^3 = 8$$

$$20 + 60 = 80$$



1 2 3 4 ... n  
↓ ↓ ↓  
2 2 2 ... 2

$2^n$   
ob<sub>1</sub> ob<sub>2</sub> ob<sub>3</sub> - NP  
0 0 0  
0 0 1  
0 1 0  
0 1 1  
1 0 0  
1 0 1  
1 1 0  
1 1 1  
25 60 60/8  
25 60 60/8  
25 60 60/8  
25 60 60/8

## 0/1 Knapsack using Branch & Bound

①



12

i	1	2	3	4
P	30	28	20	24
W	5	7	4	2

## 0/1 Knapsack using Branch & Bound

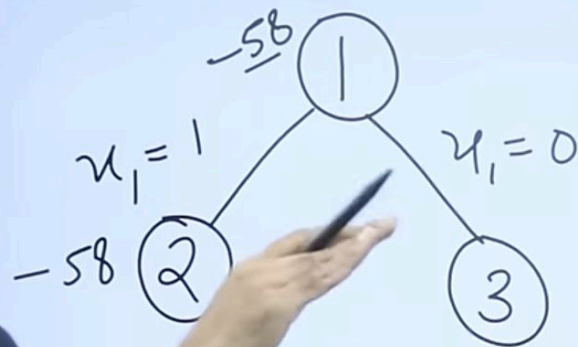
58 (1)

2	28
1	30
<hr/>	
58	

12  
70

i	1	2	3	4
P	30	28	20	24
W	<u>5</u>	7	4	2

## 0/1 Knapsack using Branch & Bound



Handwritten table showing the state of the knapsack:

2	-	28
1	-	30
		58

Annotations:  $12$  and  $70$  are written next to the table.

i	1	2	3	4
P	30	28	20	24
W	5	7	4	2

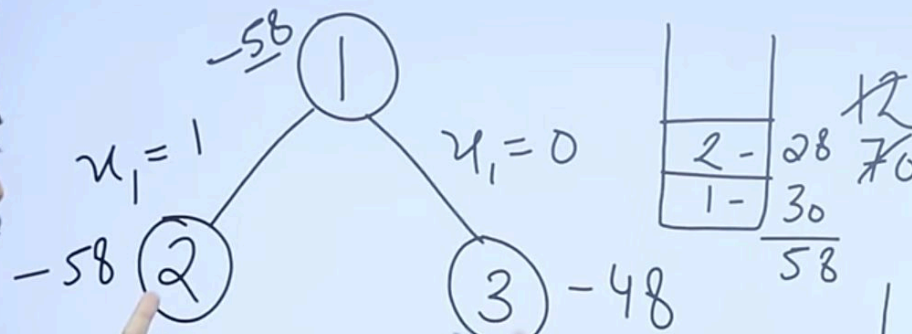
Handwritten table showing the state of the knapsack:

3	20
2	28

Annotations:  $12$  and  $51$  are written next to the table.



## 0/1 Knapsack using Branch & Bound



2	28
1	30
	58

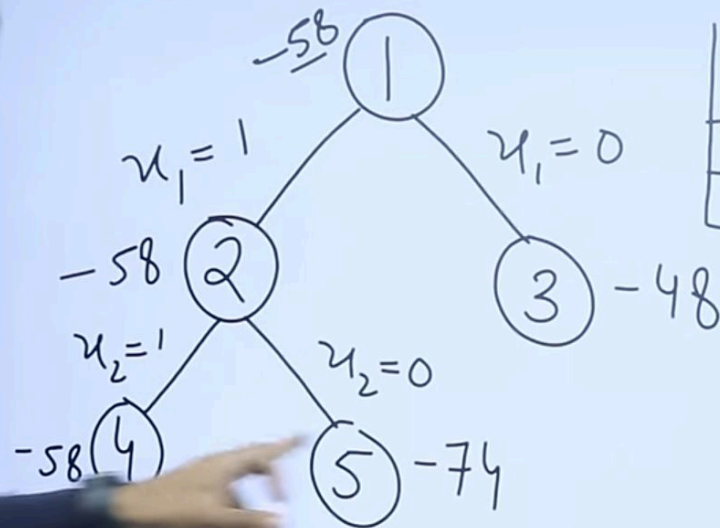
Handwritten notes:  $12$ ,  $70$

i	1	2	3	4
P	30	28	20	24
W	5	7	4	2

3	20
2	28

Handwritten notes:  $12$ ,  $51$

## 0/1 Knapsack using Branch & Bound



$$\begin{array}{r}
 2 - 28 \\
 1 - 30 \\
 \hline
 58
 \end{array}$$

Handwritten notes: 12, 70

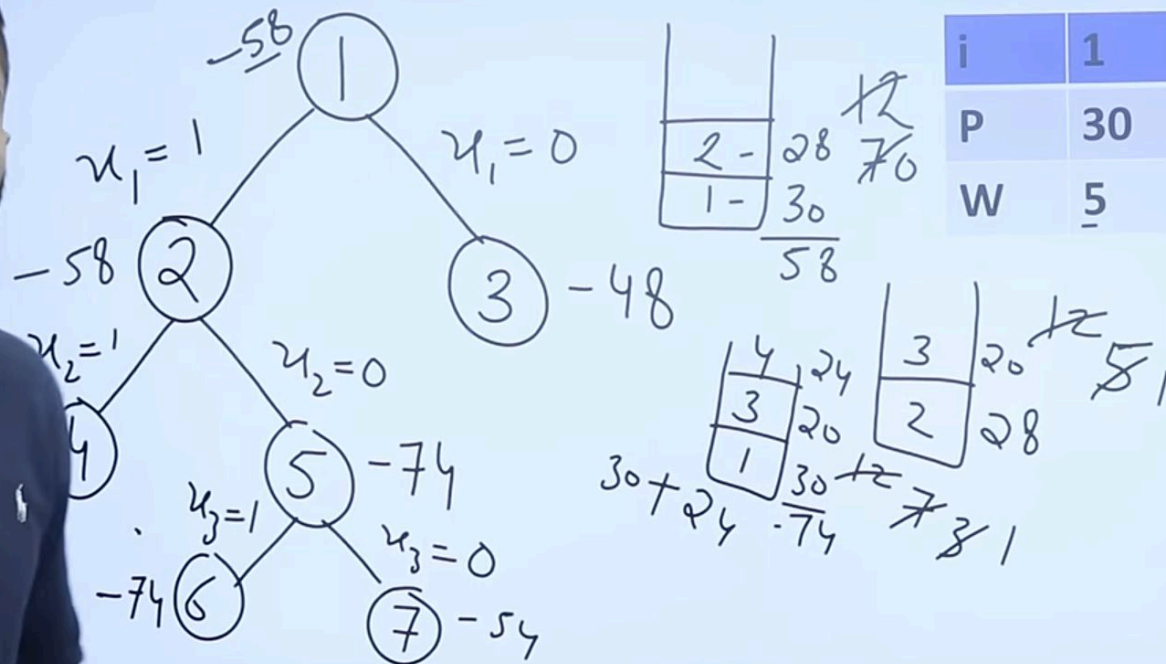
i	1	2	3	4
P	30	28	20	24
W	5	7	4	2

$$\begin{array}{r}
 4 - 24 \\
 3 - 20 \\
 1 - 30 \\
 \hline
 74
 \end{array}$$

Handwritten notes: 12, 81

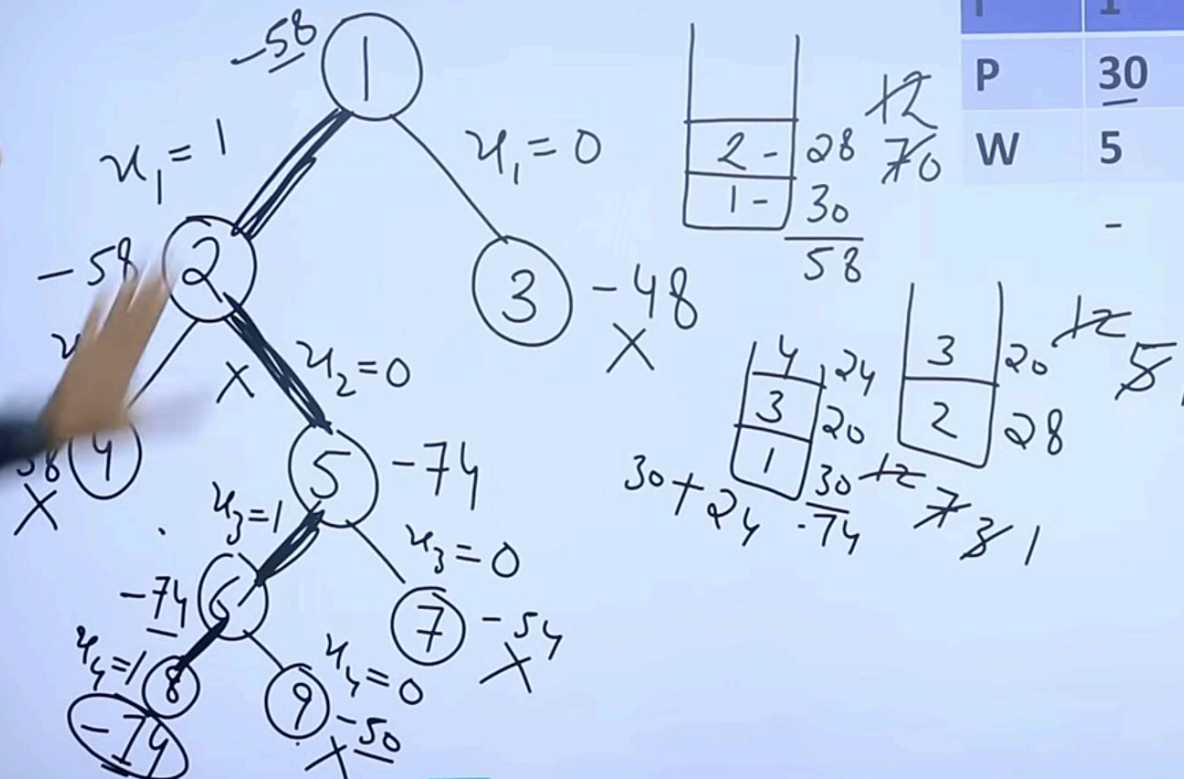


## 0/1 Knapsack using Branch & Bound



# 0/1 Knapsack using Branch & Bound

i	1	2	3	4
P	30	28	20	24
W	5	7	4	2

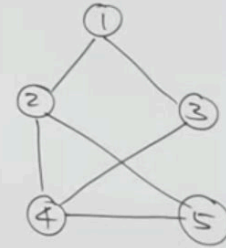
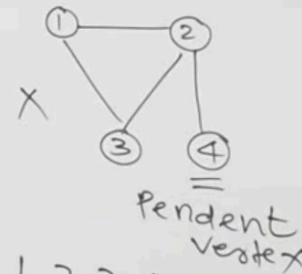
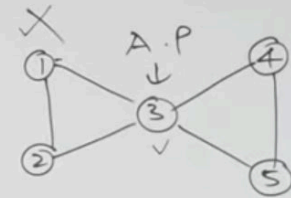
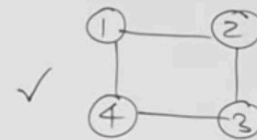


# Hamiltonian Circuit using Backtracking (All vertices covered once and return to start)

One of the most  
famous NP-Complete  
problems

# Backtracking

## Hamiltonian Cycle



1 2 3 4 1  
2 3 4 1 2

1 3 2 4

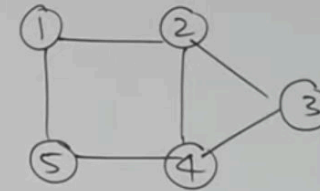
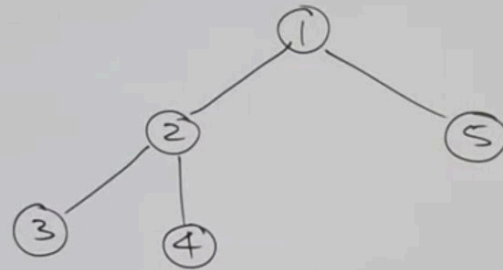
1 2 3 4 5  
2 1 3 4 5

1 2 5 4 3 1

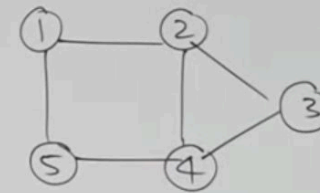
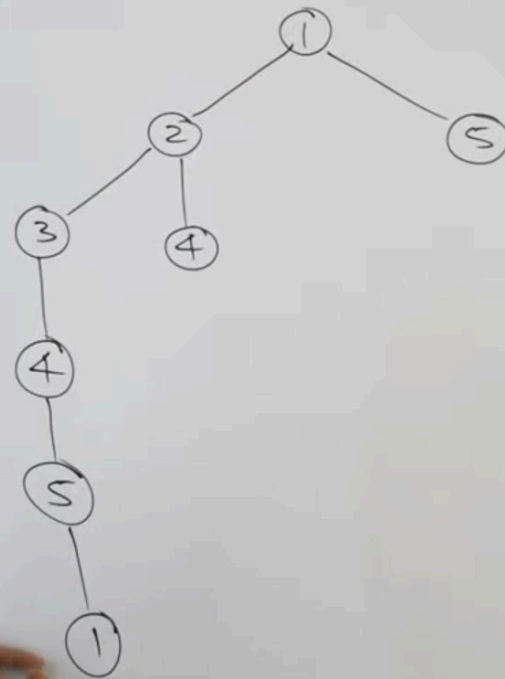
1 3 4 5 2 1



# Backtracking Hamiltonian Cycle



# Backtracking Hamiltonian Cycle

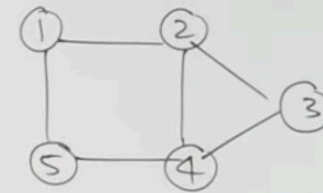
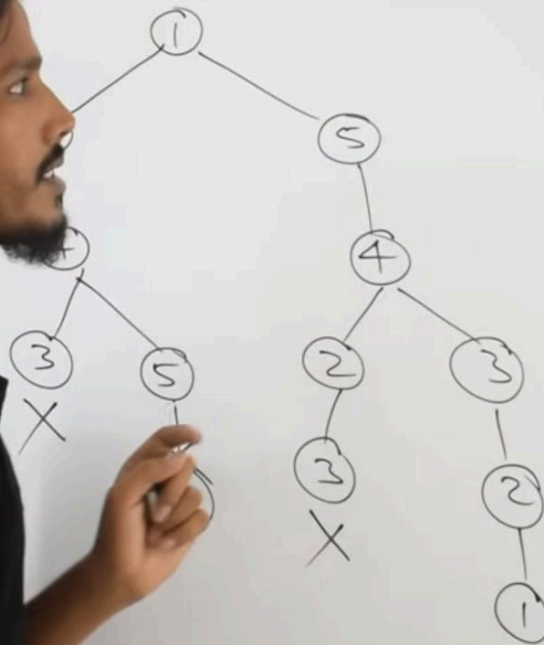


1 2 3 4 5 1



# Backtracking

## Hamiltonian Cycle



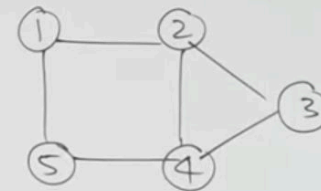
1 2 3 4 5 1  
1 2 4 3 X  
1 2 4 5 1  
1 5 4 2 3 1  
1 5 ~~4~~ 3 2 1



# Backtracking

## Hamiltonian Cycle

NP-C



1 2 3 4 5 1

1 2 4 3 X

1 2 4 5 1

1 5 4 2 3 1

1 5 ~~4~~ 3 2 1

