🐆

# Explanation

## Complete Step-by-Step Trace: `recursion(1, 4, cuts)` where cuts = [0,1,3,4,5,7]

### MAIN CALL: `recursion(1, 4, cuts)`

**Goal:** Make cuts at positions 1,2,3,4 (values 1,3,4,5) in segment [0,7]

**Segment length:** cuts[5] - cuts[0] = 7 - 0 = 7

## TRY idx = 1 (cut at value 1 as LAST)

### Step 1: `left_partition = recursion(1, 0, cuts)`

- i=1, j=0 → i > j → **Return 0**

### Step 2: `right_partition = recursion(2, 4, cuts)`

**Goal:** Make cuts at positions 2,3,4 (values 3,4,5) in segment [1,7]

**Segment length:** cuts[5] - cuts[1] = 7 - 1 = 6

### TRY idx = 2 in recursion(2, 4, cuts)

### Step 2.1: `left_partition = recursion(2, 1, cuts)`

- i=2, j=1 → i > j → **Return 0**

### Step 2.2: `right_partition = recursion(3, 4, cuts)`

**Goal:** Make cuts at positions 3,4 (values 4,5) in segment [3,7]

**Segment length:** cuts[5] - cuts[2] = 7 - 3 = 4

### TRY idx = 3 in recursion(3, 4, cuts)

**Step 2.2.1:** `left_partition = recursion(3, 2, cuts)`

- i=3, j=2 → i > j → **Return 0**

**Step 2.2.2:** `right_partition = recursion(4, 4, cuts)` **Goal:** Make cut at position 4 (value 5) in segment [4,7]

**Segment length:** cuts[5] - cuts[3] = 7 - 4 = 3

**TRY idx = 4 in recursion(4, 4, cuts):**

- `left_partition = recursion(4, 3, cuts)` → i > j → **Return 0**
- `right_partition = recursion(5, 4, cuts)` → i > j → **Return 0**
- `total_cost = 0 + 0 + 3 = 3`

**`recursion(4, 4, cuts)` returns 3**

**Back to recursion(3, 4, cuts) idx=3:**

- `total_cost = 0 + 3 + 4 = 7`
- `min_cost = 7`

## TRY idx = 4 in recursion(3, 4, cuts)

**Step 2.2.3:** `left_partition = recursion(3, 3, cuts)` **Goal:** Make cut at position 3 (value 4) in segment [3,4]

**Segment length:** cuts[4] - cuts[2] = 5 - 3 = 2

**TRY idx = 3 in recursion(3, 3, cuts):**

- `left_partition = recursion(3, 2, cuts)` → i > j → **Return 0**
- `right_partition = recursion(4, 3, cuts)` → i > j → **Return 0**
- `total_cost = 0 + 0 + 2 = 2`

**`recursion(3, 3, cuts)` returns 2**

**Step 2.2.4:** `right_partition = recursion(5, 4, cuts)`

- i=5, j=4 → i > j → **Return 0**

**Back to recursion(3, 4, cuts) idx=4:**

- `total_cost = 2 + 0 + 4 = 6`

- `min_cost = min(7, 6) = 6`

**`recursion(3, 4, cuts)` returns 6**

## Back to recursion(2, 4, cuts) idx=2:

- `total_cost = 0 + 6 + 6 = 12`
- `min_cost = 12`

## TRY idx = 3 in recursion(2, 4, cuts)

### Step 2.3: `left_partition = recursion(2, 2, cuts)`

**Goal:** Make cut at position 2 (value 3) in segment [1,4]

**Segment length:** cuts[3] - cuts[1] = 4 - 1 = 3

**TRY idx = 2 in recursion(2, 2, cuts):**

- `left_partition = recursion(2, 1, cuts)` → i > j → **Return 0**
- `right_partition = recursion(3, 2, cuts)` → i > j → **Return 0**
- `total_cost = 0 + 0 + 3 = 3`

**`recursion(2, 2, cuts)` returns 3**

### Step 2.4: `right_partition = recursion(4, 4, cuts)`

- We already calculated this: **returns 3**

## Back to recursion(2, 4, cuts) idx=3:

- `total_cost = 3 + 3 + 6 = 12`
- `min_cost = min(12, 12) = 12`

## TRY idx = 4 in recursion(2, 4, cuts)

### Step 2.5: `left_partition = recursion(2, 3, cuts)`

**Goal:** Make cuts at positions 2,3 (values 3,4) in segment [1,5]

**Segment length:** cuts[4] - cuts[1] = 5 - 1 = 4

### TRY idx = 2 in recursion(2, 3, cuts)

**Step 2.5.1:** `left_partition = recursion(2, 1, cuts)` → **Return 0**

**Step 2.5.2:** `right_partition = recursion(3, 3, cuts)`

- We already calculated this: **returns 2**

**Back to recursion(2, 3, cuts) idx=2:**

- `total_cost = 0 + 2 + 4 = 6`
- `min_cost = 6`

### TRY idx = 3 in recursion(2, 3, cuts)

**Step 2.5.3:** `left_partition = recursion(2, 2, cuts)`

- We already calculated this: **returns 3**

**Step 2.5.4:** `right_partition = recursion(4, 3, cuts)` → **Return 0**

**Back to recursion(2, 3, cuts) idx=3:**

- `total_cost = 3 + 0 + 4 = 7`
- `min_cost = min(6, 7) = 6`

**`recursion(2, 3, cuts)`** **returns 6**

### Step 2.6: `right_partition = recursion(5, 4, cuts)` → **Return 0**

### Back to recursion(2, 4, cuts) idx=4:

- `total_cost = 6 + 0 + 6 = 12`
- `min_cost = min(12, 12) = 12`

**`recursion(2, 4, cuts)`** **returns 12**

### Back to MAIN CALL idx=1:

- `total_cost = 0 + 12 + 7 = 19`
- `min_cost = 19`

---

## TRY idx = 2 (cut at value 3 as LAST)

### Step 3: `left_partition = recursion(1, 1, cuts)`

**Goal:** Make cut at position 1 (value 1) in segment [0,3]

**Segment length:** cuts[2] - cuts[0] = 3 - 0 = 3

**TRY idx = 1 in recursion(1, 1, cuts):**

- `left_partition = recursion(1, 0, cuts)` → **Return 0**
- `right_partition = recursion(2, 1, cuts)` → **Return 0**
- `total_cost = 0 + 0 + 3 = 3`

`recursion(1, 1, cuts)` **returns 3**

### Step 4: `right_partition = recursion(3, 4, cuts)`

- We already calculated this: **returns 6**

### Back to MAIN CALL idx=2:

- `total_cost = 3 + 6 + 7 = 16`
- `min_cost = min(19, 16) = 16`

---

## TRY idx = 3 (cut at value 4 as LAST)

### Step 5: `left_partition = recursion(1, 2, cuts)`

**Goal:** Make cuts at positions 1,2 (values 1,3) in segment [0,4]

**Segment length:** cuts[3] - cuts[0] = 4 - 0 = 4

### TRY idx = 1 in recursion(1, 2, cuts)

### Step 5.1: `left_partition = recursion(1, 0, cuts)` → Return 0

### Step 5.2: `right_partition = recursion(2, 2, cuts)`

- We already calculated this: **returns 3**

### Back to recursion(1, 2, cuts) idx=1:

- `total_cost = 0 + 3 + 4 = 7`

- `min_cost = 7`

### TRY idx = 2 in recursion(1, 2, cuts)

### Step 5.3: `left_partition = recursion(1, 1, cuts)`

- We already calculated this: **returns 3**

### Step 5.4: `right_partition = recursion(3, 2, cuts)` → **Return 0**

### Back to recursion(1, 2, cuts) idx=2:

- `total_cost = 3 + 0 + 4 = 7`
- `min_cost = min(7, 7) = 7`

**`recursion(1, 2, cuts)` returns 7**

### Step 6: `right_partition = recursion(4, 4, cuts)`

- We already calculated this: **returns 3**

### Back to MAIN CALL idx=3:

- `total_cost = 7 + 3 + 7 = 17`
- `min_cost = min(16, 17) = 16`

---

## TRY idx = 4 (cut at value 5 as LAST)

### Step 7: `left_partition = recursion(1, 3, cuts)`

**Goal:** Make cuts at positions 1,2,3 (values 1,3,4) in segment [0,5]

**Segment length:** cuts[4] - cuts[0] = 5 - 0 = 5

### TRY idx = 1 in recursion(1, 3, cuts)

### Step 7.1: `left_partition = recursion(1, 0, cuts)` → **Return 0**

### Step 7.2: `right_partition = recursion(2, 3, cuts)`

- We already calculated this: **returns 6**

### Back to recursion(1, 3, cuts) idx=1:

- `total_cost = 0 + 6 + 5 = 11`
- `min_cost = 11`

### TRY idx = 2 in recursion(1, 3, cuts)

### Step 7.3: `left_partition = recursion(1, 1, cuts)` → returns 3

### Step 7.4: `right_partition = recursion(3, 3, cuts)` → returns 2

### Back to recursion(1, 3, cuts) idx=2:

- `total_cost = 3 + 2 + 5 = 10`
- `min_cost = min(11, 10) = 10`

### TRY idx = 3 in recursion(1, 3, cuts)

### Step 7.5: `left_partition = recursion(1, 2, cuts)` → returns 7

### Step 7.6: `right_partition = recursion(4, 3, cuts)` → Return 0

### Back to recursion(1, 3, cuts) idx=3:

- `total_cost = 7 + 0 + 5 = 12`
- `min_cost = min(10, 12) = 10`

**`recursion(1, 3, cuts)` returns 10**

### Step 8: `right_partition = recursion(5, 4, cuts)` → Return 0

### Back to MAIN CALL idx=4:

- `total_cost = 10 + 0 + 7 = 17`
- `min_cost = min(16, 17) = 16`

---

## FINAL RESULT

**`recursion(1, 4, cuts)` returns 16**

The minimum cost to make all cuts [1,3,4,5] in stick of length 7 is **16**.

---

## Setup After Sorting and Adding Boundaries

```
 cuts = [1,3,4,5]   # Original cuts (sorted)
cuts = [0] + cuts + [7]
 # becomes [0,1,3,4,5,7]
c = 4
 # original cuts count
i = 1, j = 4
 # Start with recursion(1, 4, cuts)
```

## Understanding the Recursive Call `recursion(1, 4, cuts)`

**What this means:**

- We want to make all cuts between positions 1 and 4 in the cuts array
- Positions 1,2,3,4 correspond to cuts[1,2,3,4] = [1,3,4,5]
- The segment boundaries are cuts[0] = 0 and cuts[5] = 7
- So we're finding minimum cost to cut the segment [0,7] at positions 1,3,4,5

## Step-by-Step Execution

### Level 1: `recursion(1, 4, cuts)`

```
 Segment: [0, 7] (length = 7)
Trying each cut as the LAST cut:

idx = 1 (cut at position 1):
├── left_partition = recursion(1, 0, cuts) = 0  (i > j, base case)
├── right_partition = recursion(2, 4, cuts)    (need to solve)
├── cur_cost = cuts[5] - cuts[0] = 7 - 0 = 7
└── total_cost = 0 + right_partition + 7

idx = 2 (cut at position 3):
├── left_partition = recursion(1, 1, cuts)     (need to solve)
├── right_partition = recursion(3, 4, cuts)    (need to solve)
├── cur_cost = 7
└── total_cost = left_partition + right_partition + 7

idx = 3 (cut at position 4):
├── left_partition = recursion(1, 2, cuts)     (need to solve)
├── right_partition = recursion(4, 4, cuts)    (need to solve)
├── cur_cost = 7
└── total_cost = left_partition + right_partition + 7

idx = 4 (cut at position 5):
├── left_partition = recursion(1, 3, cuts)      (need to solve)
├── right_partition = recursion(5, 4, cuts) = 0 (i > j, base case)
├── cur_cost = 7
└── total_cost = left_partition + 0 + 7
```

### Level 2 Examples:

#### `recursion(2, 4, cuts)` - Cuts in segment [1, 7]:

```
 Positions to cut: 2,3,4 → cuts at 3,4,5
Segment length: cuts[5] - cuts[1] = 7 - 1 = 6

idx = 2: cost = 0 + recursion(3,4) + 6
idx = 3: cost = recursion(2,2) + recursion(4,4) + 6
idx = 4: cost = recursion(2,3) + 0 + 6
```

#### `recursion(1, 1, cuts)` - Cut in segment [0, 3]:

```
 Only position 1 to cut → cut at 1
Segment length: cuts[2] - cuts[0] = 3 - 0 = 3

idx = 1: cost = 0 + 0 + 3 = 3
```

## Key Insights About the Indexing

1. `i` **and** `j` **are positions in the cuts array**, not cut values
2. **Segment boundaries**: cuts[i-1] to cuts[j+1]
3. **Cuts to make**: at positions i, i+1, ..., j
4. **Base case**: When i > j, no cuts needed → return 0

## Example Trace for Small Subproblem

Let's trace `recursion(2, 3, cuts)`:

```
 Cuts to make: positions 2,3 → cuts at 3,4
Segment: [cuts[1], cuts[4]] = [1, 5] (length = 4)

idx = 2 (cut at 3 last):
├── left: recursion(2, 1) = 0  (base case)
├── right: recursion(3, 3) = ?
├── cost: 4
└── total: 0 + recursion(3,3) + 4
```

```
idx = 3 (cut at 4 last):
├── left: recursion(2, 2) = ?
├── right: recursion(4, 3) = 0  (base case)
├── cost: 4
└── total: recursion(2,2) + 0 + 4
```

Where:

- `recursion(3, 3)` = cost to cut at position 3 in segment [3,5] = 2
- `recursion(2, 2)` = cost to cut at position 2 in segment [1,4] = 3

The algorithm recursively explores all possible "last cut" choices and returns the minimum cost path, just like the DP visualization shows but using top-down recursion instead of bottom-up iteration.