

12
.1

Assign Cookies

EASY

■ What You Will Learn

- ✓ The greedy mindset: always try to satisfy the least greedy child first
- ✓ How sorting + two pointers gives an $O(n \log n)$ solution
- ✓ Why greedy works here — and when it doesn't
- ✓ The classic 'match smallest resource to smallest need' pattern

■ Intuition First

You're a parent with a bag of cookies of different sizes. Each child has a 'greed factor' — the minimum cookie size they'll accept. Your goal: make as many children happy as possible.

The trick? Start with the least greedy child and give them the smallest cookie that satisfies them. This way you waste as little cookie as possible on each child, leaving bigger cookies for greedier kids.

■ The Greedy Choice

- Sort children by greed factor (smallest first)
- Sort cookies by size (smallest first)
- Use two pointers: try to match `cookie[j]` to `child[i]`
- If `cookie[j] >= greed[i]`: assign it, move both pointers
- If `cookie[j] < greed[i]`: this cookie satisfies no one — move `j`

■ Visual Explanation

Children's greed factors (sorted): [1, 2, 3]

Cookie sizes (sorted): [1, 1, 2, 3]

Think of it as matching items on two number lines:

Children: [1]■[2]■[3]

Cookies: [1]■[1]■[2]■[3]

Match: ↑ ↑ ↑

Result: 1→ch1 2→ch2 3→ch3 → 3 children satisfied ✓

■ How It Works — Step by Step

1. Sort `greed[]` array and `cookie[]` array in ascending order

2. Initialize $i=0$ (child pointer), $j=0$ (cookie pointer), $count=0$
3. While $i < children.length$ AND $j < cookies.length$:
4. If $cookies[j] \geq greed[i]$: child i is satisfied $\rightarrow i++, j++, count++$
5. Else: cookie too small for this child $\rightarrow j++$ (try next cookie)
6. Return $count$

■ Dry Run — $greed=[1,2,3]$, $cookies=[1,1,2,3]$

After sorting both arrays:

$greed[]$ — children sorted by hunger (least greedy first)

1	2	3
---	---	---

$cookies[]$ — sorted by size (smallest first)

1	1	2	3
---	---	---	---

Step 1 $i=0, j=0$: $cookies[0]=1 \geq greed[0]=1 \rightarrow$ MATCH! $count=1, i=1, j=1$

1	2	3
---	---	---

Step 2 $i=1, j=1$: $cookies[1]=1 < greed[1]=2 \rightarrow$ too small, $j=2$

1	2	3
---	---	---

Step 3 $i=1, j=2$: $cookies[2]=2 \geq greed[1]=2 \rightarrow$ MATCH! $count=2, i=2, j=3$

1	2	3
---	---	---

Step 4 $i=2, j=3$: $cookies[3]=3 \geq greed[2]=3 \rightarrow$ MATCH! $count=3, i=3 \rightarrow$ done

1	2	3
---	---	---

$\rightarrow$ All 3 children satisfied! Answer = 3 ✓

■ Java Code

```
public static int assignCookies(int[] greed, int[] cookies) {
    Arrays.sort(greed); // Sort children: least greedy first
    Arrays.sort(cookies); // Sort cookies: smallest first
    int i = 0; // Child pointer
    int j = 0; // Cookie pointer
    int count = 0;
    while (i < greed.length && j < cookies.length) {
        if (cookies[j] >= greed[i]) {
```

```
// Cookie satisfies this child
count++;
i++; // Move to next child
}
// Always try next cookie (whether matched or not)
j++;
}
return count;
}
```

■ Time & Space Complexity

Case	Time	Space	What it really means
All Cases	$O(n \log n)$	$O(1)$	Dominated by sorting — the two-pointer scan is just $O(n)$
Space	$O(1)$	$O(1)$	Sorting in-place, only two integer pointers extra

■ Why Greedy Works Here

- Greedy works when a locally optimal choice doesn't hurt globally
- Using the smallest satisfying cookie for the least greedy child is NEVER worse than any alternative
- Proof: if you gave a bigger cookie to a less greedy child, you'd be wasting it — same or fewer total children satisfied

■ Common Mistakes

- Sorting only one array — BOTH must be sorted
- Using `cookies[j] > greed[i]` instead of `>=` (missing exact matches)
- Returning `j` or `i` instead of `count` — those pointers include unmatched iterations

■ Pro Tips

- `j++` happens outside the `if` — it's unconditional. Cookie is used or discarded, either way move on
- This pattern (two sorted arrays, two pointers) appears everywhere: interval problems, job scheduling
- If cookies can be reused: think `HashMap` instead

■ Visual Thinking

Picture a vending machine dispensing snacks. Snacks are sorted smallest-to-biggest on a conveyor belt.

Kids line up smallest-to-biggest appetite. The machine tries each snack:

→ Snack big enough for front kid? Dispense, kid leaves, next kid steps up.

→ Snack too small? Drop it on the floor, pull next snack.

Animation idea: show two sorted rows sliding past each other with match lines connecting pairs.

Green glow on successful matches, gray fade on discarded cookies.

■ Practice Problems

[Easy] `greed=[1,2]`, `cookies=[1,2,3]` — trace manually

[Easy] What happens when cookies are all smaller than all greed values?

[Med] What if each child can receive multiple cookies?

[Med] Assign tasks to workers: `tasks=[2,4,1]`, `workers=[0,3,3]`

[Hard] What if children can swap cookies among themselves? Max happiness?

■ Quick Revision Box

- Sort both arrays ascending
- Two pointers `i` (child), `j` (cookie)
- Match if `cookies[j] >= greed[i] → count++, i++`
- Always `j++` regardless of match
- $O(n \log n)$ sort dominates, $O(1)$ space
- Classic 'least greedy first' pattern