Programming: Diffuse Tokens

Problem Story

Jānis got tired of paying for a ChatGPT subscription and wants to develop his own Large Language Model (LLM) to assist himself and developers in Lemon AI.

> ***LLMs*** *are neural networks that generate text. They encode language as* ***Tokens*** *— chunks of text such as words, word pieces, or characters (e.g.,* `Hello`*,* `AI`*,* `olymp`*,* `iad`*,* `!` *are tokens encoding* `Hello AI olympiad!`*).*

The traditional approach to building LLMs is to use Transformer-based models.

> ***Transformer*** *is a revolutionary architecture introduced in the paper "Attention is All You Need" by Vaswani et al. in 2017. Since then, it has become the baseline architecture for many NLP (Natural Language Processing) tasks, including text generation.*

However, Jānis knows about the downside of Transformer models: they are autoregressive in their nature, limiting the speed and efficiency of text generation.

> ***Autoregressive models*** *must generate tokens one-by-one in order: after token is generated, it is appended to the result, and the next token is generated based on the updated result. That limits generation to be sequential.*

Jānis wants to explore **Diffusion models**, which can generate tokens in arbitrary order for faster, parallel inference.

> ***Diffusion models*** *are non-autoregressive and can generate tokens in any order, not just left-to-right.*

However, Jānis knows about a problem with Diffusion models — they often collapse into autoregressive-like generation: generating tokens sequentially, left-to-right. To prevent this, Jānis constrains the generation order: generate non-adjacent tokens first, then the rest.

Before actually building the Diffusion LLM, Jānis wants to prototype and measure how much more efficient it is.

He has a very special GPU, which consumes more power as the values of the tokens on input and output increase. To minimize costs, he wants to compute the **accumulated power cost** (as described below).

Jānis came up with the following problem:

Problem

We have a sequence of n tokens: $P = [p_1, p_2, \ldots, p_n]$, where each token p_i has a power cost equal to its value. We need to find the optimal generation order that minimizes the **accumulated power cost** while preventing **autoregressive collapse**.

The **accumulated power cost** for a sequence $[a_1, a_2, \ldots, a_n]$ is:

```
sum = a₁ + (a₁ + a₂) + (a₁ + a₂ + a₃) + ... + (a₁ + a₂ + ... + aₙ)
```

For example, $[1, 3, 9]$ has accumulated cost $= 1 + (1 + 3) + (1 + 3 + 9) = 18$

The constraint to prevent **autoregressive collapse** is:

- Divide tokens into two groups by their position parity in the original sequence: odd positions (1st, 3rd, 5th, ...) and even positions (2nd, 4th, 6th, ...)
- All tokens from one parity group must be output before any tokens from the other parity group

The output should be a permutation (reordering) of the original sequence P.

Input

The first argument contains an integer n ($1 \le n \le 10^5$) — the length of the sequence.

The second argument contains n space-separated integers $p_1, p_2, \ldots, p_n$ ($1 \le p_i \le 10^9$) — the elements of the sequence.

Output

Print n space-separated integers representing the optimal valid permutation.

If multiple optimal solutions exist, you may output any of them.

Limits

- Test case count ≤ 100
- Sum of n over all test cases $\le 5 \times 10^5$
- Execution time limit: 5 seconds
- Memory limit: 512 MB

Examples

Input:

```
3
5 1 4
```

Output:

```
1 4 5
```

This answer follows the alternating pattern: first selecting the token at the even position (2nd: value 1), then tokens at odd positions (1st and 3rd: values 5 and 4, sorted as 4, 5). The accumulated cost is 1 + (1+4) + (1+4+5) = 16, which is optimal.

Grading

Score = passed tests / total tests

Further exploration

If you are interested in how LLMs work, check out this video by 3Blue1Brown: http://youtube.com/watch?v=LPZh9BOjkQs

If you want to learn more, continue with his Neural Networks series after LLM video! The playlist includes a guest video from Welch Labs on how Transformers work: https://youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi