Tubics Coding Challenge

This coding challenge is one part of our recruiting process where you can show us your best coding and problem solving skills.

Developers at tubics enjoy a lot of freedom; we take ownership of entire features from design to implementation to testing and deployment.

We don’t tell you how to do things, because we assume you know best and you’ll do your best.

Problem

Every player has a planet. To let players attack another player’s planets, we need you to create armies of randomly distributed spaceships (such as “UNSC Infinity”, “Millenium Falcon”, “Enterprise”, etc.)

For example, we'll call your code telling it we need a random army that's 167 spaceships strong. Assuming our available unit types to be, for example, “UNSC Infinity”, “Millenium Falcon” and “Enterprise”, what we want from you is that you tell us what such a random army would look like, e.g.

Our Input: 167
Example result: 63 UNSC Infinity 57 Galactica 47 Enterprise

(The "text output" above is just an example; we need this as structured data)

Boundaries

- Each spaceship MUST be > 0
- The result MUST be different (non-deterministic) with each call. When we call your code 100 times with the same parameters, we expect 100 different results.
- Obvious biases in the result are strongly discouraged. An obvious bias would be if e.g. one spaceship type often is the largest (or smallest etc.)
- Downtime is the enemy. Once deployed, code updates are strongly discouraged. We want this to be deployed-and-forget while still being future proof.
- We appreciate an O(1) solution
  O(1) describes an algorithm that will always execute in the same time (or space) regardless of the size of the input data set.

Remarks

- This is _your_ solution and you own it end to end. That is, you are free to decide the design, implementation, docs, etc. Just give us your best.
- Imagine this is a stand-alone application that needs to be integrated to the tubics ecosystem and used by other services, so consider how this would be deployed/hosted.
- We appreciate a solution that makes use of the tubics tool-stack: NextJS, React, Typescript, NodeJS, Express.
- Invest a reasonable amount of time, no more and no less.
- Your code must be correct and it is up to you to prove that it is actually working as intended (docs, testing and code coverage).
- We appreciate a smart algorithm. We'll look at your general way of problem solving, how elegant your solution is, that is, how well your solution performs with as few resources as possible (resources including CPU, memory, dependencies).
- We are looking for pragmatic solutions. It is a relatively simple problem and we are looking for a simple, pragmatic solution.
- We are looking for flexible solutions that can be easily extended and enhanced for future use cases.
- We are also looking for simple solutions over over-engineered ones.
- We also appreciate a good build and deployment setup.
- If you have questions please feel free to ask any time.

Once you are done with your solution, please zip the source code and send it via email back to us. If you have a demo running somewhere publicly, please include the URL. Essentially include everything that we need to run your solution. Instructions _are_ helpful.

Thank you very much in advance for investing your time in solving our coding challenge, we really appreciate your efforts.

Note: This coding challenge is confidential, so you are not allowed to share it with anyone or make the solution publicly available.