# Problem 2715: Timeout Cancellation

## Problem Information

**Difficulty:** Easy
**Acceptance Rate:** 89.52%
**Paid Only:** No

## Problem Description

Given a function `fn`, an array of arguments `args`, and a timeout `t` in milliseconds, return a cancel function `cancelFn`.

After a delay of `cancelTimeMs`, the returned cancel function `cancelFn` will be invoked.

setTimeout(cancelFn, cancelTimeMs)

Initially, the execution of the function `fn` should be delayed by `t` milliseconds.

If, before the delay of `t` milliseconds, the function `cancelFn` is invoked, it should cancel the delayed execution of `fn`. Otherwise, if `cancelFn` is not invoked within the specified delay `t`, `fn` should be executed with the provided `args` as arguments.

**Example 1:**

**Input:** fn = (x) => x * 5, args = [2], t = 20 **Output:** [{"time": 20, "returned": 10}] **Explanation:** const cancelTimeMs = 50; const cancelFn = cancellable((x) => x * 5, [2], 20); setTimeout(cancelFn, cancelTimeMs); The cancellation was scheduled to occur after a delay of cancelTimeMs (50ms), which happened after the execution of fn(2) at 20ms.

**Example 2:**

**Input:** fn = (x) => x**2, args = [2], t = 100 **Output:** [] **Explanation:** const cancelTimeMs = 50; const cancelFn = cancellable((x) => x**2, [2], 100); setTimeout(cancelFn, cancelTimeMs); The cancellation was scheduled to occur after a delay of cancelTimeMs (50ms), which happened before the execution of fn(2) at 100ms, resulting in fn(2) never being called.

**Example 3:**

**Input:** fn = (x1, x2) => x1 * x2, args = [2,4], t = 30 **Output:** [{"time": 30, "returned": 8}] **Explanation:** const cancelTimeMs = 100; const cancelFn = cancellable((x1, x2) => x1 * x2, [2,4], 30); setTimeout(cancelFn, cancelTimeMs); The cancellation was scheduled to occur after a delay of cancelTimeMs (100ms), which happened after the execution of fn(2,4) at 30ms.

**Constraints:**

* `fn` is a function * `args` is a valid JSON array * `1 <= args.length <= 10` * `20 <= t <= 1000` * `10 <= cancelTimeMs <= 1000`

## Code Snippets

**JavaScript:**

```
/**
* @param {Function} fn
* @param {Array} args
* @param {number} t
* @return {Function}
*/
var cancellable = function(fn, args, t) {

};

/**
* const result = [];
*
* const fn = (x) => x * 5;
* const args = [2], t = 20, cancelTimeMs = 50;
*
* const start = performance.now();
*
* const log = (...argsArr) => {
* const diff = Math.floor(performance.now() - start);
* result.push({"time": diff, "returned": fn(...argsArr)});
* }
*
```

```
* const cancel = cancellable(log, args, t);
*
* const maxT = Math.max(t, cancelTimeMs);
*
* setTimeout(cancel, cancelTimeMs);
*
* setTimeout(() => {
* console.log(result); // [{"time":20,"returned":10}]
* }, maxT + 15)
*/
```

**TypeScript:**

```
type JSONValue = null | boolean | number | string | JSONValue[] | { [key:
string]: JSONValue };
type Fn = (...args: JSONValue[]) => void

function cancellable(fn: Fn, args: JSONValue[], t: number): Function {

};

/**
* const result = [];
*
* const fn = (x) => x * 5;
* const args = [2], t = 20, cancelTimeMs = 50;
*
* const start = performance.now();
*
* const log = (...argsArr) => {
* const diff = Math.floor(performance.now() - start);
* result.push({"time": diff, "returned": fn(...argsArr)});
* }
*
* const cancel = cancellable(log, args, t);
*
* const maxT = Math.max(t, cancelTimeMs);
*
* setTimeout(cancel, cancelTimeMs);
*
* setTimeout(() => {
* console.log(result); // [{"time":20,"returned":10}]
* }, maxT + 15)
*/
```