/** * Gets or sets the length of the array. This is a number one higher than the highest index in the array. */ length: number; /** * Returns a string representation of an array. */ toString(): string; /** * Returns a string representation of an array. The elements are converted to string using their toLocaleString methods. */ toLocaleString(): string; /** * Removes the last element from an array and returns it. * If the array is empty, undefined is returned and the array is not modified. */ pop(): T | undefined; /** * Appends new elements to the end of an array, and returns the new length of the array. * @param items New elements to add to the array. */ push(...items: T[]): number; /** * Combines two or more arrays. * This method returns a new array without modifying any existing arrays. * @param items Additional arrays and/or items to add to the end of the array. */ concat(...items: ConcatArray[]): T[]; /** * Combines two or more arrays. * This method returns a new array without modifying any existing arrays. * @param items Additional arrays and/or items to add to the end of the array. */ concat(...items: (T | ConcatArray)[]): T[]; /** * Adds all the elements of an array into a string, separated by the specified separator string. * @param separator A string used to separate one element of the array from the next in the resulting string. If omitted, the array elements are separated with a comma. */ join(separator?: string): string; /** * Reverses the elements in an array in place. * This method mutates the array and returns a reference to the same array. */ reverse(): T[]; /** * Removes the first element from an array and returns it. * If the array is empty, undefined is returned and the array is not modified. */ shift(): T | undefined; /** * Returns a copy of a section of an array. * For both start and end, a negative index can be used to indicate an offset from the end of the array. * For example, -2 refers to the second to last element of the array. * @param start The beginning index of the specified portion of the array. * If start is undefined, then the slice begins at index 0. * @param end The end index of the specified portion of the array. This is exclusive of the element at the index 'end'. * If end is undefined, then the slice extends to the end of the array. */ slice(start?: number, end?: number): T[]; /** * Sorts an array in place. * This method mutates the array and returns a reference to the same array. * @param compareFn Function used to determine the order of the elements. It is expected to return * a negative value if the first argument is less than the second argument, zero if they're equal, and a positive * value otherwise. If omitted, the elements are sorted in ascending, ASCII character order. * ```ts * [11,2,22,1].sort((a, b) => a - b) * ``` */ sort(compareFn?: (a: T, b: T) => number): this; /** * Removes elements from an array and, if necessary, inserts new elements in their place, returning the deleted elements. * @param start The zero-based location in the array from which to start removing elements. * @param deleteCount The number of elements to remove. * @returns An array containing the elements that were deleted. */ splice(start: number, deleteCount?: number): T[]; /** * Removes elements from an array and, if necessary, inserts new elements in their place, returning the deleted elements. * @param start The zero-based location in the array from which to start removing elements. * @param deleteCount The number of elements to remove. * @param items Elements to insert into the array in place of the deleted elements. * @returns An array containing the elements that were deleted. */ splice(start: number, deleteCount: number, ...items: T[]): T[]; /** * Inserts new elements at the start of an array, and returns the new length of the array. * @param items Elements to insert at the start of the array. */ unshift(...items: T[]): number; /** * Returns the index of the first occurrence of a value in an array, or -1 if it is not present. * @param searchElement The value to locate in the array. * @param fromIndex The array index at which to begin the search. If fromIndex is omitted, the search starts at index 0. */ indexOf(searchElement: T, fromIndex?: number): number; /** * Returns the index of the last occurrence of a specified value in an array, or -1 if it is not present. * @param searchElement The value to locate in the array. * @param fromIndex The array index at which to begin searching backward. If fromIndex is omitted, the search starts at the last index in the array. */ lastIndexOf(searchElement: T, fromIndex?: number): number; /** * Determines whether all the members of an array satisfy the specified test. * @param predicate A function that accepts up to three arguments. The every method calls * the predicate function for each element in the array until the predicate returns a value * which is coercible to the Boolean value false, or until the end of the array. * @param thisArg An object to which the this keyword can refer in the predicate function. * If thisArg is omitted, undefined is used as the this value. */ every~~(predicate: (value: T, index: number, array: T[]) => value is S, thisArg?: any): this is S[]; /** * Determines whether all the members of an array satisfy the specified test. * @param predicate A function that accepts up to three arguments. The every method calls * the predicate function for each element in the array until the predicate returns a value * which is coercible to the Boolean value false, or until the end of the array. * @param thisArg An object~~

to which the this keyword can refer in the predicate function. * If thisArg is omitted, undefined is used as the this value. */ every(predicate: (value: T, index: number, array: T[]) => unknown, thisArg?: any): boolean; /** * Determines whether the specified callback function returns true for any element of an array. * @param predicate A function that accepts up to three arguments. The some method calls * the predicate function for each element in the array until the predicate returns a value * which is coercible to the Boolean value true, or until the end of the array. * @param thisArg An object to which the this keyword can refer in the predicate function. * If thisArg is omitted, undefined is used as the this value. */ some(predicate: (value: T, index: number, array: T[]) => unknown, thisArg?: any): boolean; /** * Performs the specified action for each element in an array. * @param callbackfn A function that accepts up to three arguments. forEach calls the callbackfn function one time for each element in the array. * @param thisArg An object to which the this keyword can refer in the callbackfn function. If thisArg is omitted, undefined is used as the this value. */ forEach(callbackfn: (value: T, index: number, array: T[]) => void, thisArg?: any): void; /** * Calls a defined callback function on each element of an array, and returns an array that contains the results. * @param callbackfn A function that accepts up to three arguments. The map method calls the callbackfn function one time for each element in the array. * @param thisArg An object to which the this keyword can refer in the callbackfn function. If thisArg is omitted, undefined is used as the this value. */ map(callbackfn: (value: T, index: number, array: T[]) => U, thisArg?: any): U[]; /** * Returns the elements of an array that meet the condition specified in a callback function. * @param predicate A function that accepts up to three arguments. The filter method calls the predicate function one time for each element in the array. * @param thisArg An object to which the this keyword can refer in the predicate function. If thisArg is omitted, undefined is used as the this value. */ filter(predicate: (value: T, index: number, array: T[]) => value is S, thisArg?: any): S[]; /** * Returns the elements of an array that meet the condition specified in a callback function. * @param predicate A function that accepts up to three arguments. The filter method calls the predicate function one time for each element in the array. * @param thisArg An object to which the this keyword can refer in the predicate function. If thisArg is omitted, undefined is used as the this value. */ filter(predicate: (value: T, index: number, array: T[]) => unknown, thisArg?: any): T[]; /** * Calls the specified callback function for all the elements in an array. The return value of the callback function is the accumulated result, and is provided as an argument in the next call to the callback function. * @param callbackfn A function that accepts up to four arguments. The reduce method calls the callbackfn function one time for each element in the array. * @param initialValue If initialValue is specified, it is used as the initial value to start the accumulation. The first call to the callbackfn function provides this value as an argument instead of an array value. */ reduce(callbackfn: (previousValue: T, currentValue: T, currentIndex: number, array: T[]) => T): T; reduce(callbackfn: (previousValue: T, currentValue: T, currentIndex: number, array: T[]) => T, initialValue: T): T; /** * Calls the specified callback function for all the elements in an array. The return value of the callback function is the accumulated result, and is provided as an argument in the next call to the callback function. * @param callbackfn A function that accepts up to four arguments. The reduce method calls the callbackfn function one time for each element in the array. * @param initialValue If initialValue is specified, it is used as the initial value to start the accumulation. The first call to the callbackfn function provides this value as an argument instead of an array value. */ reduce(callbackfn: (previousValue: U, currentValue: T, currentIndex: number, array: T[]) => U, initialValue: U): U; /** * Calls the specified callback function for all the elements in an array, in descending order. The return value of the callback function is the accumulated result, and is provided as an argument in the next call to the callback function. * @param callbackfn A function that accepts up to four arguments. The reduceRight method calls the callbackfn function one time for each element in the array. * @param initialValue If initialValue is specified, it is used as the initial value to start the accumulation. The first call to the callbackfn function provides this value as an argument instead of an array value. */ reduceRight(callbackfn: (previousValue: T, currentValue: T, currentIndex: number, array: T[]) => T): T; reduceRight(callbackfn: (previousValue: T, currentValue: T, currentIndex: number, array: T[]) => T, initialValue: T): T; /** * Calls the specified callback function for all the elements in an array, in descending order. The return value of the callback function is the accumulated result, and is provided as an argument in the next call to the callback function. * @param callbackfn A function that accepts up to four arguments. The reduceRight method calls the callbackfn function one time for each element in the array. * @param initialValue If initialValue is specified, it is used as the initial value to start the accumulation. The first call to the callbackfn function provides this value as an argument instead of an array value. */ reduceRight(callbackfn: (previousValue:

~~U, currentValue: T, currentIndex: number, array: T[]) => U, initialValue: U): U; [n: number]: T;~~