Q1.1 Type Intersections

1.1.1

T1 = {a:number[]}; T2 = {b:string};

Typescript type expressions: type T1&T2 = {a:number[], b:string}

Examples:

{a:[3], b:"soldin"}

{a:[1,2,3], b:"sagiv"}

{a:[],b:"boo"}

1.1.2

T1 = {a:{b:number}} T2 = {a:{c:string}};

Typescript type expression: T1&T2 = {a:{b:number, c:string}}

Examples:

{a:{b: 3, c:"Shalom"}}

{a:{b: -6, c:"shabat"}}

{a:{b: 5, c:"purim"}}

1.1.3

T1 = {a: number[] }; T2 = {a: number};

Typescript type expression: type T1&T2 = {null, undefined}|{a:null|
undefined}

Examples:

let t2: T1 & T2= null;

let t3: T1 & T2= undefined;

let t4: T1 & T2= {a: null};

Q1.2 Type Inclusion

1.2.1

type T1 = {a:number, b:{}}[] type T2 = {a:number}[]

T1 < T2, T1 type is a subtype of T2. T1 has all the keys of T2,and

more, and the value is of the same type.

1.2.2

type T1 = {a: {c: any}, b: any} type T2 = {a: {c: number}, b: number}

T2 < T1, T2 type is a subtype of T1. The 'any' type could be
type, including number. so, T2 has all the keys of T1.
any

1.2.3

type T1 = {a:number, b:undefined} type T2 = {a:number, b:any}

T1 < T2, T1 type is a subtype of T2. Undefined is subtype of any
type, including any.

Q1.3 Type Inference

1.3.1

Answer: Type of v1 is: (name: string ,age: number)
let v1 = { name:"eran", age:29 };

1.3.2

Answer: Type of v2 is: {children: ({name:string}|{age:number})[] };
v2 = { children: [{name: "Roy"}, {age:25}] };

1.3.3

Answer: type V3 = (number) => number;
v3 = (x) => x + 50;

1.3.4

Answer: type v4 = <T>(f: (x:T) =>T, l: T []|{}) => T[]|{};
v4 = (f, l) => map((x)=>f(f(x)), l);

Q1.4 Type Definitions

1.4.1

It is not possible, since there's no way to represent an infinite subset of an atomic type

(string) in TS. If it was a finite subset of values we could use the | operator (e.g. type T=

"a"|"b"|"c";) , but this is not the case.

1.4.2

It's not possible, for the same reason above: number is an atomic type, hence as mentioned

above there's no way to represent a non-finite subset of its elements in TS. $|\{1,2,....\}| = \infty$.

Q3.2 sorted?

sorted? should return #t on both an empty list and a list with a single element, since both are

sorted by any comparator by definition (vacuously true).