Ppl

Q 1.1

1.1.1

T1 = {a: number[]; }
T2 = {b: string};
answer: {a :number[] , b:string}
example:
T1 = {a=[1,2,3]}
T2 = {b="aa"}
T = {a :[1,2,3] , b:"aa"}

1.1.2

T1 = {a: {b: number}}
T2 = {a: {c: string}};
answer: {a: { b: number, c: string}}
example:
T1 = {a: {b: 2}}
T2 = {a: {c: "bb"}};
T= {a: { b: 2, c:"bb"}}

1.1.3

T1 = {a: number[]; }
T2 = {a: number};
answer:{a: number[] & number}
example:
T1 = {a: [1,2]; }
T2 = {a: 3};
T= undefined

Q 1.2

1.2.1

type T1 = {a: number, b:{}}[]
type T2 = {a: number}[]
answer: T1<T2
if x is of type T1 than x = {a: number, b:{}} , so that x is a map with the key a and the value of a is a type number so is of type T2, and for T1 there is another key.

1.2.2

type T1 = {a: {c: any}, b:any}
type T2 = {a: {c: number}, b: number}
answer: T2<T1
if x is of type T2 than x = {a: {c: number}, b: number}, so that x is a map with the key c and b that the a

is a values in type number so they of type any (that include type number).

1.2.3

type T1 = {a: number, b:undefined}
type T2 = {a: number, b:any}
answer:T1<T2
if x is of type T2 than x = {a: number, b:undefined}, so that x is a map with the key b that the type undefined is also from type any, because undefined is a sub type of all types.

Q 1.3

1.3.1

v1 = { name: "peter", age:20 };
answer: type of v1 is- {name: string , age: number)

1.3.2

v2 = { children: [{name: "john"}, {age:12}] };
answer: type of v2 is -{ children: ({name: string}|{age: number})[] };

1.3.3

v3 = (x) => x + 2;
answer: type of v3 is - (x: any)=> any

1.3.4

v4 = (f, l) => map((x)=>f(f(x)), l);
answer: type of v4 is – (f: T1=>T1 ,l:T1[])=> T1[]

Q 1.4

1.4.1 Is it possible to define a type in TypeScript for the set of all strings with length larger than 2 using the type constructors defined in class?

answer: No, it is impossible to define this new type because it is not an atomic type. Because of that we can't create a compound type with recursive array or interface of atomic type.

1.4.2 Is it possible to define a type for the set of all numbers larger than 0?

answer: No, it is impossible to define this type because it isn't an atomic type. as was mentioned in section 1.4.1 we can't this type.