Stdlib	Containers	Batteries	Base
	CCSeq.(--) : int -> int -> int Seq.t	BatSeq.(--) : int -> int -> int Seq.t	
		BatSeq.(---) : int -> int -> int Seq.t	
		BatSeq.(--.) : float * float -> float -> float Seq.t	
	CCSeq.(--^) : int -> int -> int Seq.t	BatSeq.(--^) : int -> int -> int Seq.t	
		BatSeq.(--~) : char -> char -> char Seq.t	
	CCSeq.(<*>) : ('a -> 'b) Seq.t -> 'a Seq.t -> 'b Seq.t		
	CCSeq.(<.>) : ('a -> 'b) Seq.t -> 'a Seq.t -> 'b Seq.t		
	CCSeq.(>>-) : 'a Seq.t -> ('a -> 'b Seq.t) -> 'b Seq.t		
	CCSeq.(>>=) : 'a Seq.t -> ('a -> 'b Seq.t) -> 'b Seq.t		Base.Sequence.(>>=) : 'a Base.Sequence.t -> ('a -> 'b Base.Sequence.t) -> 'b Base.Sequence.t
	CCSeq.(>|=) : 'a Seq.t -> ('a -> 'b) -> 'b Seq.t		Base.Sequence.(>>|) : 'a Base.Sequence.t -> ('a -> 'b) -> 'b Base.Sequence.t
		BatSeq.(//) : 'a Seq.t -> ('a -> bool) -> 'a Seq.t	
		BatSeq.(//@) : 'a Seq.t -> ('a -> 'b option) -> 'b Seq.t	
		BatSeq.(/@) : 'a Seq.t -> ('a -> 'b) -> 'b Seq.t	
		BatSeq.(@/) : ('a -> 'b) -> 'a Seq.t -> 'b Seq.t	
		BatSeq.(@//) : ('a -> 'b option) -> 'a Seq.t -> 'b Seq.t	
Seq.Nil : 'a Seq.node = Seq.Nil	CCSeq.Nil : 'a CCSeq.node = CCSeq.Nil	BatSeq.Nil : 'a BatSeq.node = BatSeq.Nil	
			Base.Sequence.all : 'a Base.Sequence.t list -> 'a list Base.Sequence.t
			Base.Sequence.all_unit : unit Base.Sequence.t list -> unit Base.Sequence.t
Seq.append : 'a Seq.t -> 'a Seq.t -> 'a Seq.t	CCSeq.append : 'a Seq.t -> 'a Seq.t -> 'a Seq.t	BatSeq.append : 'a Seq.t -> 'a Seq.t -> 'a Seq.t	Base.Sequence.append : 'a Base.Sequence.t -> 'a Base.Sequence.t -> 'a Base.Sequence.t
		BatSeq.assoc : 'a -> ('a * 'b) Seq.t -> 'b option	
		BatSeq.at : 'a Seq.t -> int -> 'a	
			Base.Sequence.bind : 'a Base.Sequence.t -> f:('a -> 'b Base.Sequence.t) -> 'b Base.Sequence.t
			Base.Sequence.bounded_length : 'a Base.Sequence.t -> at_most:int -> [`Greater | `Is of int]
			Base.Sequence.cartesian_product : 'a Base.Sequence.t -> 'b Base.Sequence.t -> ('a * 'b) Base.Sequence.t
			Base.Sequence.chunks_exn : 'a Base.Sequence.t -> int -> 'a list Base.Sequence.t
		BatSeq.combine : 'a Seq.t -> 'b Seq.t -> ('a * 'b) Seq.t	
	CCSeq.compare : 'a CCSeq.ord -> 'a Seq.t CCSeq.ord		Base.Sequence.compare : ('a -> 'a -> int) -> 'a Base.Sequence.t -> 'a Base.Sequence.t -> int
		BatSeq.concat : 'a Seq.t Seq.t -> 'a Seq.t	Base.Sequence.concat : 'a Base.Sequence.t Base.Sequence.t -> 'a Base.Sequence.t
			Base.Sequence.concat_map : 'a Base.Sequence.t -> f:('a -> 'b Base.Sequence.t) -> 'b Base.Sequence.t
			Base.Sequence.concat_mapi : 'a Base.Sequence.t -> f:(int -> 'a -> 'b Base.Sequence.t) -> 'b Base.Sequence.t
Seq.cons : 'a -> 'a Seq.t -> 'a Seq.t	CCSeq.cons : 'a -> 'a Seq.t -> 'a Seq.t	BatSeq.cons : 'a -> 'a Seq.t -> 'a Seq.t	
			Base.Sequence.count : 'a Base.Sequence.t -> f:('a -> bool) -> int
			Base.Sequence.counti : 'a Base.Sequence.t -> f:(int -> 'a -> bool) -> int
	CCSeq.cycle : 'a Seq.t -> 'a Seq.t		
			Base.Sequence.cycle_list_exn : 'a list -> 'a Base.Sequence.t
			Base.Sequence.delayed_fold : 'a Base.Sequence.t -> init:'s -> f:('s -> 'a -> k:('s -> 'r) -> 'r) -> finish:('s -> 'r) -> 'r
	CCSeq.drop : int -> 'a Seq.t -> 'a Seq.t	BatSeq.drop : int -> 'a Seq.t -> 'a Seq.t	Base.Sequence.drop : 'a Base.Sequence.t -> int -> 'a Base.Sequence.t
			Base.Sequence.drop_eagerly : 'a Base.Sequence.t -> int -> 'a Base.Sequence.t
	CCSeq.drop_while : ('a -> bool) -> 'a Seq.t -> 'a Seq.t	BatSeq.drop_while : ('a -> bool) -> 'a Seq.t -> 'a Seq.t	Base.Sequence.drop_while : 'a Base.Sequence.t -> f:('a -> bool) -> 'a Base.Sequence.t

Stdlib	Containers	Batteries	Base
			Base.Sequence.drop_while_option : 'a Base.Sequence.t -> f:('a -> bool) -> ('a * 'a Base.Sequence.t) option
Seq.empty : 'a Seq.t	CCSeq.empty : 'a Seq.t	BatSeq.empty : 'a Seq.t	Base.Sequence.empty : 'a Base.Sequence.t
		BatSeq.enum : 'a Seq.t -> 'a BatEnum.t	
	CCSeq.equal : 'a CCSeq.equal -> 'a Seq.t CCSeq.equal	BatSeq.equal : ?eq:('a -> 'a -> bool) -> 'a Seq.t -> 'a Seq.t -> bool	Base.Sequence.equal : ('a -> 'a -> bool) -> 'a Base.Sequence.t -> 'a Base.Sequence.t -> bool
	CCSeq.exists : ('a -> bool) -> 'a Seq.t -> bool	BatSeq.exists : ('a -> bool) -> 'a Seq.t -> bool	Base.Sequence.exists : 'a Base.Sequence.t -> f:('a -> bool) -> bool
	CCSeq.exists2 : ('a -> 'b -> bool) -> 'a Seq.t -> 'b Seq.t -> bool		
			Base.Sequence.existsi : 'a Base.Sequence.t -> f:(int -> 'a -> bool) -> bool
	CCSeq.fair_app : ('a -> 'b) Seq.t -> 'a Seq.t -> 'b Seq.t		
	CCSeq.fair_flat_map : ('a -> 'b Seq.t) -> 'a Seq.t -> 'b Seq.t		
Seq.filter : ('a -> bool) -> 'a Seq.t -> 'a Seq.t	CCSeq.filter : ('a -> bool) -> 'a Seq.t -> 'a Seq.t	BatSeq.filter : ('a -> bool) -> 'a Seq.t -> 'a Seq.t	Base.Sequence.filter : 'a Base.Sequence.t -> f:('a -> bool) -> 'a Base.Sequence.t
Seq.filter_map : ('a -> 'b option) -> 'a Seq.t -> 'b Seq.t	CCSeq.filter_map : ('a -> 'b option) -> 'a Seq.t -> 'b Seq.t	BatSeq.filter_map : ('a -> 'b option) -> 'a Seq.t -> 'b Seq.t	Base.Sequence.filter_map : 'a Base.Sequence.t -> f:('a -> 'b option) -> 'b Base.Sequence.t
			Base.Sequence.filter_mapi : 'a Base.Sequence.t -> f:(int -> 'a -> 'b option) -> 'b Base.Sequence.t
			Base.Sequence.filter_opt : 'a option Base.Sequence.t -> 'a Base.Sequence.t
			Base.Sequence.filteri : 'a Base.Sequence.t -> f:(int -> 'a -> bool) -> 'a Base.Sequence.t
		BatSeq.find : ('a -> bool) -> 'a Seq.t -> 'a option	Base.Sequence.find : 'a Base.Sequence.t -> f:('a -> bool) -> 'a option
			Base.Sequence.find_consecutive_duplicate : 'a Base.Sequence.t -> equal:('a -> 'a -> bool) -> ('a * 'a) option
			Base.Sequence.find_exn : 'a Base.Sequence.t -> f:('a -> bool) -> 'a
		BatSeq.find_map : ('a -> 'b option) -> 'a Seq.t -> 'b option	Base.Sequence.find_map : 'a Base.Sequence.t -> f:('a -> 'b option) -> 'b option
			Base.Sequence.find_mapi : 'a Base.Sequence.t -> f:(int -> 'a -> 'b option) -> 'b option
			Base.Sequence.findi : 'a Base.Sequence.t -> f:(int -> 'a -> bool) -> (int * 'a) option
		BatSeq.first : 'a Seq.t -> 'a	
Seq.flat_map : ('a -> 'b Seq.t) -> 'a Seq.t -> 'b Seq.t	CCSeq.flat_map : ('a -> 'b Seq.t) -> 'a Seq.t -> 'b Seq.t	BatSeq.flat_map : ('a -> 'b Seq.t) -> 'a Seq.t -> 'b Seq.t	
	CCSeq.flatten : 'a Seq.t Seq.t -> 'a Seq.t	BatSeq.flatten : 'a Seq.t Seq.t -> 'a Seq.t	
	CCSeq.fmap : ('a -> 'b option) -> 'a Seq.t -> 'b Seq.t		
	CCSeq.fold : ('a -> 'b -> 'a) -> 'a -> 'b Seq.t -> 'a		Base.Sequence.fold : 'a Base.Sequence.t -> init:'accum -> f:('accum -> 'a -> 'accum) -> 'accum
	CCSeq.fold2 : ('acc -> 'a -> 'b -> 'acc) -> 'acc -> 'a Seq.t -> 'b Seq.t -> 'acc		
Seq.fold_left : ('a -> 'b -> 'a) -> 'a -> 'b Seq.t -> 'a	CCSeq.fold_left : ('a -> 'b -> 'a) -> 'a -> 'b Seq.t -> 'a	BatSeq.fold_left : ('a -> 'b -> 'a) -> 'a -> 'b Seq.t -> 'a	
			Base.Sequence.fold_m : bind:('acc_m -> f:('acc -> 'acc_m) -> 'acc_m) -> return:('acc -> 'acc_m) -> 'elt Base.Sequence.t -> init:'acc -> f:('acc -> 'elt -> 'acc_m) -> 'acc_m
			Base.Sequence.fold_result : 'a Base.Sequence.t -> init:'accum -> f:('accum -> 'a -> ('accum, 'e) Base.Result.t) -> ('accum, 'e) Base.Result.t
		BatSeq.fold_right : ('a -> 'b -> 'b) -> 'a Seq.t -> 'b -> 'b	
			Base.Sequence.fold_until : 'a Base.Sequence.t -> init:'accum -> f:('accum -> 'a -> ('accum, 'final) Base.Container_intf.Continue_or_stop.t) -> finish:('accum -> 'final) -> 'final
			Base.Sequence.foldi : ('a Base.Sequence.t, 'a, 'b) Base.Indexed_container_intf.foldi
			Base.Sequence.folding_map : 'a Base.Sequence.t -> init:'b -> f:('b -> 'a -> 'b * 'c) -> 'c Base.Sequence.t
			Base.Sequence.folding_mapi : 'a Base.Sequence.t -> init:'b -> f:(int -> 'b -> 'a -> 'b * 'c) -> 'c Base.Sequence.t
	CCSeq.for_all : ('a -> bool) -> 'a Seq.t -> bool	BatSeq.for_all : ('a -> bool) -> 'a Seq.t -> bool	Base.Sequence.for_all : 'a Base.Sequence.t -> f:('a -> bool) -> bool
			Base.Sequence.for_alli : 'a Base.Sequence.t -> f:(int -> 'a -> bool) -> bool

Stdlib	Containers	Batteries	Base
	CCSeq.for_all2 : ('a -> 'b -> bool) -> 'a Seq.t -> 'b Seq.t -> bool		
			Base.Sequence.force_eagerly : 'a Base.Sequence.t -> 'a Base.Sequence.t
	CCSeq.group : 'a CCSeq.equal -> 'a Seq.t -> 'a Seq.t Seq.t		
			Base.Sequence.group : 'a Base.Sequence.t -> break:('a -> 'a -> bool) -> 'a list Base.Sequence.t
	CCSeq.head : 'a Seq.t -> 'a option		Base.Sequence.hd : 'a Base.Sequence.t -> 'a option
	CCSeq.head_exn : 'a Seq.t -> 'a	BatSeq.hd : 'a Seq.t -> 'a	Base.Sequence.hd_exn : 'a Base.Sequence.t -> 'a
			Base.Sequence.ignore_m : 'a Base.Sequence.t -> unit Base.Sequence.t
		BatSeq.init : int -> (int -> 'a) -> 'a Seq.t	Base.Sequence.init : int -> f:(int -> 'a) -> 'a Base.Sequence.t
	CCSeq.interleave : 'a Seq.t -> 'a Seq.t -> 'a Seq.t		Base.Sequence.interleave : 'a Base.Sequence.t Base.Sequence.t -> 'a Base.Sequence.t
			Base.Sequence.interleaved_cartesian_product : 'a Base.Sequence.t -> 'b Base.Sequence.t -> ('a * 'b) Base.Sequence.t
			Base.Sequence.intersperse : 'a Base.Sequence.t -> sep:'a -> 'a Base.Sequence.t
	CCSeq.is_empty : 'a Seq.t -> bool	BatSeq.is_empty : 'a Seq.t -> bool	Base.Sequence.is_empty : 'a Base.Sequence.t -> bool
Seq.iter : ('a -> unit) -> 'a Seq.t -> unit	CCSeq.iter : ('a -> unit) -> 'a Seq.t -> unit	BatSeq.iter : ('a -> unit) -> 'a Seq.t -> unit	Base.Sequence.iter : 'a Base.Sequence.t -> f:('a -> unit) -> unit
	CCSeq.iter2 : ('a -> 'b -> unit) -> 'a Seq.t -> 'b Seq.t -> unit	BatSeq.iter2 : ('a -> 'b -> unit) -> 'a Seq.t -> 'b Seq.t -> unit	
			Base.Sequence.iter_m : bind:('unit_m -> f:(unit -> 'unit_m) -> 'unit_m) -> return:(unit -> 'unit_m) -> 'elt Base.Sequence.t -> f:('elt -> 'unit_m) -> 'unit_m
	CCSeq.iteri : (int -> 'a -> unit) -> 'a Seq.t -> unit	BatSeq.iteri : (int -> 'a -> unit) -> 'a Seq.t -> unit	Base.Sequence.iteri : ('a Base.Sequence.t, 'a) Base.Indexed_container_intf.iteri
			Base.Sequence.join : 'a Base.Sequence.t Base.Sequence.t -> 'a Base.Sequence.t
		BatSeq.last : 'a Seq.t -> 'a	
	CCSeq.length : 'a Seq.t -> int	BatSeq.length : 'a Seq.t -> int	Base.Sequence.length : 'a Base.Sequence.t -> int
			Base.Sequence.length_is_bounded_by : ?min:int -> ?max:int -> 'a Base.Sequence.t -> bool
		BatSeq.make : int -> 'a -> 'a Seq.t	
Seq.map : ('a -> 'b) -> 'a Seq.t -> 'b Seq.t	CCSeq.map : ('a -> 'b) -> 'a Seq.t -> 'b Seq.t	BatSeq.map : ('a -> 'b) -> 'a Seq.t -> 'b Seq.t	Base.Sequence.map : 'a Base.Sequence.t -> f:('a -> 'b) -> 'b Base.Sequence.t
	CCSeq.map2 : ('a -> 'b -> 'c) -> 'a Seq.t -> 'b Seq.t -> 'c Seq.t	BatSeq.map2 : ('a -> 'b -> 'c) -> 'a Seq.t -> 'b Seq.t -> 'c Seq.t	
	CCSeq.mapi : (int -> 'a -> 'b) -> 'a Seq.t -> 'b Seq.t	BatSeq.mapi : (int -> 'a -> 'b) -> 'a Seq.t -> 'b Seq.t	Base.Sequence.mapi : 'a Base.Sequence.t -> f:(int -> 'a -> 'b) -> 'b Base.Sequence.t
		BatSeq.max : 'a Seq.t -> 'a	
			Base.Sequence.max_elt : 'a Base.Sequence.t -> compare:('a -> 'a -> int) -> 'a option
		BatSeq.mem : 'a -> 'a Seq.t -> bool	
			Base.Sequence.mem : 'a Base.Sequence.t -> 'a -> equal:('a -> 'a -> bool) -> bool
	CCSeq.memoize : 'a Seq.t -> 'a Seq.t		Base.Sequence.memoize : 'a Base.Sequence.t -> 'a Base.Sequence.t
	CCSeq.merge : 'a CCSeq.ord -> 'a Seq.t -> 'a Seq.t -> 'a Seq.t		Base.Sequence.merge : 'a Base.Sequence.t -> 'a Base.Sequence.t -> compare:('a -> 'a -> int) -> 'a Base.Sequence.t
			Base.Sequence.merge_with_duplicates : 'a Base.Sequence.t -> 'b Base.Sequence.t -> compare:('a -> 'b -> int) -> ('a, 'b) Base.Sequence.Merge_with_duplicates_element.t Base.Sequence.t
		BatSeq.min : 'a Seq.t -> 'a	
			Base.Sequence.min_elt : 'a Base.Sequence.t -> compare:('a -> 'a -> int) -> 'a option
			Base.Sequence.next : 'a Base.Sequence.t -> ('a * 'a Base.Sequence.t) option
	CCSeq.nil : 'a Seq.t	BatSeq.nil : 'a Seq.t	
			Base.Sequence.nth : 'a Base.Sequence.t -> int -> 'a option
			Base.Sequence.nth_exn : 'a Base.Sequence.t -> int -> 'a
	CCSeq.of_array : 'a array -> 'a Seq.t		
	CCSeq.of_gen : 'a CCSeq.gen -> 'a Seq.t		

Stdlib	Containers	Batteries	Base
			Base.Sequence.of_lazy : 'a Base.Sequence.t Base.Lazy.t -> 'a Base.Sequence.t
	CCSeq.of_list : 'a list -> 'a Seq.t	BatSeq.of_list : 'a list -> 'a Seq.t	Base.Sequence.of_list : 'a list -> 'a Base.Sequence.t
			Base.Sequence.of_seq : 'a Base.Import.Caml.Seq.t -> 'a Base.Sequence.t
		BatSeq.of_string : ?first:string -> ?last:string -> ?sep:string -> (string -> 'a) -> string -> 'a Seq.t	
	CCSeq.pp : ?pp_start:unit CCSeq.printer -> ?pp_stop:unit CCSeq.printer -> ?pp_sep:unit CCSeq.printer -> 'a CCSeq.printer -> 'a Seq.t CCSeq.printer		
		BatSeq.print : ?first:string -> ?last:string -> ?sep:string -> ('a BatInnerIO.output -> 'b -> unit) -> 'a BatInnerIO.output -> 'b Seq.t -> unit	
	CCSeq.product : 'a Seq.t -> 'b Seq.t -> ('a * 'b) Seq.t		
	CCSeq.product_with : ('a -> 'b -> 'c) -> 'a Seq.t -> 'b Seq.t -> 'c Seq.t		
	CCSeq.pure : 'a -> 'a Seq.t		
	CCSeq.range : int -> int -> int Seq.t		Base.Sequence.range : ?stride:int -> ?start:[`exclusive | `inclusive] -> ?stop:[`exclusive | `inclusive] -> int -> int -> int Base.Sequence.t
			Base.Sequence.reduce : 'a Base.Sequence.t -> f:('a -> 'a -> 'a) -> 'a option
		BatSeq.reduce : ('a -> 'a -> 'a) -> 'a Seq.t -> 'a	Base.Sequence.reduce_exn : 'a Base.Sequence.t -> f:('a -> 'a -> 'a) -> 'a
			Base.Sequence.remove_consecutive_duplicates : 'a Base.Sequence.t -> equal:('a -> 'a -> bool) -> 'a Base.Sequence.t
	CCSeq.repeat : ?n:int -> 'a -> 'a Seq.t		Base.Sequence.repeat : 'a -> 'a Base.Sequence.t
Seq.return : 'a -> 'a Seq.t	CCSeq.return : 'a -> 'a Seq.t	BatSeq.return : 'a -> 'a Seq.t	Base.Sequence.return : 'a -> 'a Base.Sequence.t
			Base.Sequence.round_robin : 'a Base.Sequence.t list -> 'a Base.Sequence.t
			Base.Sequence.sexp_of_t : ('a -> Base.Ppx_sexp_conv_lib.Sexp.t) -> 'a Base.Sequence.t -> Base.Ppx_sexp_conv_lib.Sexp.t
			Base.Sequence.shift_left : 'a Base.Sequence.t -> int -> 'a Base.Sequence.t
			Base.Sequence.shift_right : 'a Base.Sequence.t -> 'a -> 'a Base.Sequence.t
			Base.Sequence.shift_right_with_list : 'a Base.Sequence.t -> 'a list -> 'a Base.Sequence.t
	CCSeq.singleton : 'a -> 'a Seq.t		Base.Sequence.singleton : 'a -> 'a Base.Sequence.t
		BatSeq.split : ('a * 'b) Seq.t -> 'a Seq.t * 'b Seq.t	
	CCSeq.sort : cmp:'a CCSeq.ord -> 'a Seq.t -> 'a Seq.t		
	CCSeq.sort_uniq : cmp:'a CCSeq.ord -> 'a Seq.t -> 'a Seq.t		
			Base.Sequence.split_n : 'a Base.Sequence.t -> int -> 'a list * 'a Base.Sequence.t
			Base.Sequence.sub : 'a Base.Sequence.t -> pos:int -> len:int -> 'a Base.Sequence.t
			Base.Sequence.sum : (module Base.Container_intf.Summable with type t = 'sum) -> 'a Base.Sequence.t -> f:('a -> 'sum) -> 'sum
	CCSeq.tail : 'a Seq.t -> 'a Seq.t option		
	CCSeq.tail_exn : 'a Seq.t -> 'a Seq.t		
	CCSeq.take : int -> 'a Seq.t -> 'a Seq.t	BatSeq.take : int -> 'a Seq.t -> 'a Seq.t	Base.Sequence.take : 'a Base.Sequence.t -> int -> 'a Base.Sequence.t
	CCSeq.take_while : ('a -> bool) -> 'a Seq.t -> 'a Seq.t	BatSeq.take_while : ('a -> bool) -> 'a Seq.t -> 'a Seq.t	Base.Sequence.take_while : 'a Base.Sequence.t -> f:('a -> bool) -> 'a Base.Sequence.t
			Base.Sequence.tl : 'a Base.Sequence.t -> 'a Base.Sequence.t option
		BatSeq.tl : 'a Seq.t -> 'a Seq.t	Base.Sequence.tl_eagerly_exn : 'a Base.Sequence.t -> 'a Base.Sequence.t
	CCSeq.to_array : 'a Seq.t -> 'a array		Base.Sequence.to_array : 'a Base.Sequence.t -> 'a array
		BatSeq.to_buffer : ?first:string -> ?last:string -> ?sep:string -> ('a -> string) -> Buffer.t -> (unit -> 'a BatSeq.node) -> unit	
	CCSeq.to_gen : 'a Seq.t -> 'a CCSeq.gen		

Stdlib	Containers	Batteries	Base
	CCSeq.to_iter : 'a Seq.t -> 'a CCSeq.iter		
	CCSeq.to_list : 'a Seq.t -> 'a list		Base.Sequence.to_list : 'a Base.Sequence.t -> 'a list
	CCSeq.to_rev_list : 'a Seq.t -> 'a list		Base.Sequence.to_list_rev : 'a Base.Sequence.t -> 'a list
			Base.Sequence.to_seq : 'a Base.Sequence.t -> 'a Base.Import.Caml.Seq.t
		BatSeq.to_string : ?first:string -> ?last:string -> ?sep:string -> ('a -> string) -> 'a Seq.t -> string	
Seq.unfold : ('b -> ('a * 'b) option) -> 'b -> 'a Seq.t	CCSeq.unfold : ('b -> ('a * 'b) option) -> 'b -> 'a Seq.t	BatSeq.unfold : ('b -> ('a * 'b) option) -> 'b -> 'a Seq.t	Base.Sequence.unfold : init:'s -> f:('s -> ('a * 's) option) -> 'a Base.Sequence.t
			Base.Sequence.unfold_step : init:'s -> f:('s -> ('a, 's) Base.Sequence.Step.t) -> 'a Base.Sequence.t
			Base.Sequence.unfold_with : 'a Base.Sequence.t -> init:'s -> f:('s -> 'a -> ('b, 's) Base.Sequence.Step.t) -> 'b Base.Sequence.t
			Base.Sequence.unfold_with_and_finish : 'a Base.Sequence.t -> init:'s_a -> running_step:('s_a -> 'a -> ('b, 's_a) Base.Sequence.Step.t) -> inner_finished:('s_a -> 's_b) -> finishing_step:('s_b -> ('b, 's_b) Base.Sequence.Step.t) -> 'b Base.Sequence.t
	CCSeq.uniq : 'a CCSeq.equal -> 'a Seq.t -> 'a Seq.t		
	CCSeq.unzip : ('a * 'b) Seq.t -> 'a Seq.t * 'b Seq.t		
	CCSeq.zip : 'a Seq.t -> 'b Seq.t -> ('a * 'b) Seq.t		Base.Sequence.zip : 'a Base.Sequence.t -> 'b Base.Sequence.t -> ('a * 'b) Base.Sequence.t
			Base.Sequence.zip_full : 'a Base.Sequence.t -> 'b Base.Sequence.t -> [`Both of 'a * 'b | `Left of 'a | `Right of 'b] Base.Sequence.t