情報科学実験レポート
ocaml3 抽象データ

6321120　横溝　尚也

提出日：６月２１日（火）

ソースコードのファイル名

課題２：ocaml_report3_1.ml
課題３：ocaml_report3_2.ml
課題４：ocaml_report3_3.ml

1　課題２

多相性のヴァリアントで定義したリスト構造を用いて，以下の処理を行うモジュールを定義しなさい．
ただし，リストの方向は「無し」（`Nil`）を参照している要素を末尾とする．
余裕のある人は，下記以外の処理も自分で定義して作成しなさい．

・`create`：空のリストをつくる
・`unshift x`：リストの先頭にデータを追加する
・`shift`：リストの先頭の要素を削除する
・`push x`：リストの末尾にデータを追加する
・`pop`：リストの末尾を削除する
・`size`：リストの要素数を返す
・`max`：最大値を返す
・`min`：最小値を返す
・`get x`：x 番目の要素を参照する
・`indexOf x`：要素が x と同じ場合にインデックスを返す．x がない場合には`-1` を返す（複数ある場合は一番小さいインデックスを返す）
・`set x y`：指定された要素を，指定された要素で置き換える（複数ある場合はすべて）
・`remove x`：指定された要素を，削除する（複数ある場合はすべて）
・`concat`：二つのリストを接続する（第一引数のリストが前に来る）

1.1　アルゴリズムの説明

今回の課題ではリスト構造を用いて Cell(a , b) の構造を作成する課題である。そのため、type 宣言を使って'a list = Nil |Cell of 'a * 'a list とした。このとき Cell の末尾には値は入れず Nil を格納する必要があったので Nil または Cell of 'a * 'a とした。講義では function をモジュール内で使っていたが、プログラムのしやすさの観点から match 文を多用している。（課題２以降でもそうしています。）

create,unshift,shift：アルゴリズムが簡単であるため省略する。

push,pop：今回扱う Cell(a,b) はリスト構造と同様に左端（リストの先頭要素）しか直接操作することはできない。よって関数 push を再帰関数として Cell の末尾まで再帰させる。今回末尾は Nil であるので Nil が訪れたときにデータを追加、または削除すればよい。

size：push,pop と同じようにデータ数を数え上げるために再帰させる。再帰させるごとに出力する要素数に１を足していき、Nil が訪れたら、要素数は数え切ったことになるので最後は 0 とすればよい。

max,min：暫定での最大値（最小値）を Cell の右側に移動させる。そしてさらに右側の要素と暫定での最大値（最小値）と比べて同じ作業を行う。この再帰をし続けると最大値（最小値）が一番右側の要素へと移動したことになるのでその値を出力すればよい。

get：max,min と似たアルゴリズムである。x 番目の要素を抽出したいのならば、左端の要素から一つづつ右の要素へと操作する要素を変更していく。その時１回再帰させるごとにはじめ０に設定していた値 count に１づつ足していく。再帰を繰り返していくと n が x になり、x になったときその要素が x 番目の要素に等しい

ため、その要素を出力する。また、要素数よりも x の値が大きいとき、つまり要素を出力できないときの処理は、n が x になる前に Nil が訪れてしまう事と同値であるため、要素の中身が Nil であったときに例外として設定した Empty を出力させればよい。

indexOf：ある要素が対象のリストの中に存在するのか調べたいときは、調べたい要素 x とリストの要素が一致すれば、その要素が何番目なのかを返す。一致しなければ残りのリスト rest で再び一致しないか再帰して調べる。再帰し続けた結果、要素の中身が Nil つまりすべての要素について調べ上げ、一致するものがなかった場合は-1 を返せばよい。一致したときに返す要素の番号は先ほどの get と同じアルゴリズムで 0 から再帰させるごとに 1 を足していき、x 番目を同時に数えていけばよい。

set：indexOf とアルゴリズムはほぼ同じである。指定した要素 x に一致すればその要素が何番目なのかを返すのではなく、今回は指定した要素 y に置き換えればよいだけである。

remove：indexOf,set と同様に指定した要素 x に一致したらその要素を削除し、前後の要素を直接接続すればよい。

concat：push とほぼ同じである。リスト l1 とリスト l2 を接続するためには l1 右端の要素と l2 の左端の要素をつなげることと同じである。しかし、前述したとおり右端の要素を直接操作することはできないので初めに再帰関数を利用して右端の要素を取り出す。Nil が現れたら、要素を追加した push とは異なりリスト l2 を追加すればよい。

1.2 プログラムの説明

```
1 module List = struct
2  type 'a list = Nil | Cell of 'a * 'a list
3  exception Empty
4  let create = Nil
5  let unshift a lst = Cell (a , lst)
6  let shift = function
7      Nil -> raise Empty
8    |Cell (_ , a) -> a

9  let rec push a lst=
10    match lst with
11      Nil -> Cell (a , Nil)
12    | Cell (x , rest) -> Cell (x , push a rest)

13  let rec pop lst =
14    match lst with
15      Nil -> raise Empty
16    |Cell (x , rest) ->
17        if rest = Nil then Nil
18        else Cell (x , pop rest)
```

```
19 let rec size lst =
20   match lst with
21     Nil -> 0
22   |Cell (a , rest) -> 1 + size rest

23 let rec max lst =
24   match lst with
25     Nil -> raise Empty
26   |Cell (x , Nil) -> x
27   |Cell (x , Cell (y , rest)) ->
28       if x > y then max (Cell (x , rest))
29       else max (Cell (y , rest))

30 let rec min lst =
31   match lst with
32     Nil -> raise Empty
33   |Cell (x , Nil) -> x
34   |Cell (x , Cell (y , rest)) ->
35       if x > y then min (Cell (y , rest))
36       else max (Cell (x , rest))

37 let get x lst =
38   let n = 0 in
39   let rec count n lst =
40     match lst with
41       Nil -> raise Empty
42     |Cell (a , rest) ->
43         if x = n then a
44         else count (n + 1) rest
45   in count n lst

46 let indexOf x lst =
47   let n = 0
48   in let rec indexOf_2 n lst =
49        match lst with
50          Nil -> -1
51        |Cell (a , rest) ->
52            if a = x then n
53            else indexOf_2 (n + 1) rest
54   in indexOf_2 n lst
```

```
55  let rec set x y lst =
56    match lst with
57      Nil -> raise Empty
58    |Cell (a , Nil) ->
59        if a = x then Cell (y , Nil)
60        else Cell (a , Nil)
61    |Cell (a , rest) ->
62        if a = x then Cell (y , set x y rest)
63        else Cell (a , set x y rest)

64  let rec remove x lst =
65    match lst with
66      Nil -> raise Empty
67    |Cell (a , Nil) ->
68        if a = x then Nil
69        else Cell (a , Nil)
70    |Cell (a , rest) ->
71        if a = x then remove x rest
72        else (Cell (a , remove x rest))

73  let rec concat l1 l2 =
74    match l1 with
75      Nil -> l2
76    |Cell (a , Nil) -> Cell (a , l2)
77    |Cell (a , rest) -> Cell (a , concat rest l2)

78   end

open List;;
let l1 = push 8 (unshift 10 (push 5 (unshift 2 create)));;
let l2 = push 1 (unshift 12 (unshift 2 create));;
let l = concat l1 l2;;
shift l;;
pop l;;
size l2;;
max l2;;
min l2;;
get 3 l;;
indexOf 12 l;;
```

```
set 1 0 l;;
remove 5 l;;
```

１～３行目：List をモジュール化し、以下すべてをモジュール内に記述した。List の型の定義は要素の最後尾を表すためのリストでいう空リストである Nil、または Cell(a,b) と表していきたいため、Cell of 'a * 'a list とした。

２～４行目：すべてのパターンを網羅するために初めに例外として Empty を宣言しておく。空リストを作る create ではリストの末尾を Nil とするため、３行目のようなプログラムとなる。

５～８行目：unshift と shift のプログラムである。unshift は左端に要素を追加するだけであり先頭の要素操作はリスト構造上、直接操作することができる。shift も unshift と同様に先頭の要素の操作であるが、今回要素の削除をするため先頭の要素が Nil であった場合、つまりリスト全体が空リストであるとき例外 Empty を出力するようにした。

９～１８行目：push と unshift のプログラムである。今回操作する要素は×美の要素であるため、リスト構造上直接操作することはできない。そのため、先頭要素から一つづつ右の要素へと再帰していき、末尾の要素を操作する。末尾の要素に到達していない場合は１２、１６行目で記述したように rest に対して再帰を行う。再帰を繰り返して末尾の要素に到達した場合、つまり要素の中身が Nil であった場合、１１、１５行目のように例外 Empty を返す。

１９～２２行目：size のプログラムである。要素数を返したいため再帰させ、一つ右の要素へと移動させるごとにカウントを１増やす。末尾まで到達したとき、つまり Nil が訪れたとき、カウントを終了したいため０を返している。

２３～３６行目：max,min のプログラムである。まず先頭要素と２番目の要素の大小を比較する（２７～２９，３４～３６行目)。最大値（最小値）の出力 s¥をしたいので、比較した２数のうち大きい方（小さい方）が２番目の要素に来るように順番を変更する。順序を変更したのちに、再帰させて再び２番目と３遍目の要素の大小比較をして同じ処理を行う。n 番目と n+1 番目の比較をしたいとき、つまり n+1 番目の要素が存在しなくなった時 n 番目に位置している要素が最大値（最小値）であるためその値を出力している（２６，３３行目)。

３７～５４行目：get,indexOf のプログラムである。どちらの関数も調べたい要素が何番目に位置しているのかインデックスを知る必要があるため、初期値 n=0 として、再帰させるごとにカウントしていく（３８，４７行目)。get では n と調べる対象の要素 a が一致するとき、、indexOf では x と a が一致するときにそれぞれ返したい値を返す（４２～４５，５１～５４行目)。すべての要素を調べ終わったとき、つまり Nil が出現したときには get では例外となる Empty、indexOf では問題指示通り-1 を返す。

５５～７２行目：set,remove のプログラムである。指定された要素 x に一致した場合、set では y に変え remove では x を削除してさらに右の要素について調べる（６１～６３，７０～７２行目)。最後の要素を調べる時行う処理は同じである。しかし処理後に全要素の調査が終わったため、再起を終了させる（５８～６０，６７～６９行目)。

７３～７７行目：concat のプログラムである。l1 の最終要素と l2 の先頭要素を接続したいため、再帰を利用して l1 の最終要素を取り出す（７７行目)。取り出せたら、Cell(a,l2) として接続すればよい。

1.2.1 実行結果

```
  end ;;
module List :
  sig
    type 'a list = Nil | Cell of 'a * 'a list
    exception Empty
    val create : 'a list
    val unshift : 'a -> 'a list -> 'a list
    val shift : 'a list -> 'a list
    val push : 'a -> 'a list -> 'a list
    val pop : 'a list -> 'a list
    val size : 'a list -> int
    val max : 'a list -> 'a
    val min : 'a list -> 'a
    val get : int -> 'a list -> 'a
    val indexOf : 'a -> 'a list -> int
    val set : 'a -> 'a -> 'a list -> 'a list
    val remove : 'a -> 'a list -> 'a list
    val concat : 'a list -> 'a list -> 'a list
  end
# open List ;;
# let l1 = push 8 (unshift 10 (push 5 (unshift 2 create))) ;;
val l1 : int List.list = Cell (10, Cell (2, Cell (5, Cell (8, Nil))))
# let l2 = push 1 (unshift 12 (unshift 2 create)) ;;
val l2 : int List.list = Cell (12, Cell (2, Cell (1, Nil)))
# let l = concat l1 l2 ;;
val l : int List.list =
  Cell (10, Cell (2, Cell (5, Cell (8, Cell (12, Cell (2, Cell (1, Nil)))))))
# shift l ;;
- : int List.list =
Cell (2, Cell (5, Cell (8, Cell (12, Cell (2, Cell (1, Nil))))))
```

図 1　実行結果-課題 2 - 1

```
# let l = concat l1 l2 ;;
val l : int List.list =
  Cell (10, Cell (2, Cell (5, Cell (8, Cell (12, Cell (2, Cell (1, Nil)))))))
# shift l ;;
- : int List.list =
Cell (2, Cell (5, Cell (8, Cell (12, Cell (2, Cell (1, Nil))))))
# pop l ;;
- : int List.list =
Cell (10, Cell (2, Cell (5, Cell (8, Cell (12, Cell (2, Nil))))))
# size l2 ;;
- : int = 3
# max l2 ;;
- : int = 12
# min l2 ;;
- : int = 1
# get 3 l ;;
- : int = 8
# indexOf 12 l ;;
- : int = 4
# set 1 0 l ;;
- : int List.list =
Cell (10, Cell (2, Cell (5, Cell (8, Cell (12, Cell (2, Cell (0, Nil)))))))
# remove 5 l ;;
- : int List.list =
Cell (10, Cell (2, Cell (8, Cell (12, Cell (2, Cell (1, Nil))))))
```

図 2　実行結果-課題 2 - 2

1.3　考察

今回の課題 2 の内容は以前に match 文、リストの特性を利用してプログラムした内容に似ている。リストの特性と似た Cell(a,b) を自分で作成し、それを module 化することで実行したときに複数の関数が簡潔に出

力された。

先頭要素の操作はリストでも Cell(a,b) の形でも簡単に行えた。しかし、右端の要素の操作はリスト構造と同様に Cell も操作が難しく、再帰関数を利用せざるを得なかった。

今回のように Cell(a,b) と組を使用してリストと同じような構造を作りたいときに抽象型'a list を宣言しておくことでプログラムの記述がかなり楽になったことがわかる。

2 課題３

データ型 Vector のモジュールを定義しなさい．
Vector モジュールは以下の関数を持つ．
・vempty: 空の Vector を返す関数．引数はなし
at: 対象の Vector の指定された位置にある要素を返す関数．第 1 引数は位置（0 以上の整数），第 2 引数は対象の Vector．
・vector: 対象の Vector（リスト）を作成する関数．第 1 引数は対象の Vector
・vlength: 対象の Vector のサイズ（長さ）を返す関数．第 1 引数は対象の Vector
・vshow: 対象の Vector 内部のすべての要素を表示する関数．第 1 引数は対象の Vector
・isvempty: 対象の Vector が空であれば true，そうでなければ false を返す関数．第 1 引数は対象の Vector
 # 内部構造はリストで構わない．
 # 文字列の出力は print_string，整数の出力は print_int，何もしない関数は () で書くことができる．また，式の区切りは; で記述できる．
 # リストの出力する関数は以下のように書くことができる．

```
let rec print_list = function
 [] -> ()
 | e::l -> print_int e ; print_string "," ;       print_list l;;
```

 # e が int 以外の場合エラーになってしまうが，気にしないでかまいません．

2.1 アルゴリズムの説明

venpty：簡単な関数であるため省略

at: 先頭要素から右の要素へとずらしていく。その際、指定された位置 n を１づつ引いていく。n が０になったときその要素が指定された要素に等しい。調べたいリストの要素数よりも指定した数 n が大きかった場合はエラーとしたいため、例外扱いする。

vector: 簡単な関数であるため省略

vlength: 右の要素へと再帰させるごとに返す値に１を足し続ける。末尾の要素に到達したとき、長さの数え上げは終了しているので０を返す。

vshow: 引数リストの要素を一つづつ出力していけばよい。見やすく出力するために改行したり、;を挟んだりする。

isvempty: 簡単な関数であるため省略

2.2 プログラムの説明

```
1 module Vector = struct
2   exception Empty
```

```
3   let vempty = []

4   let rec at n lst =
5     match (n , lst) with
6       (_ , []) -> raise Empty
7     |(0 , a :: rest) -> a
8     |(n , a :: rest) -> at (n - 1) rest

9   let vector lst = lst
10
11   let rec vlength lst =
12     match lst with
13       [] -> 0
14     |a :: rest -> 1 + (vlength rest)

15   let vshow lst =
16     let rec print_list lst=
17       match lst with [] -> ()
18        | a :: rest -> print_int a ; print_string "," ;
20      print_list rest
21     in print_list lst

22   let isvempty x =
23     if x = vempty then true
24     else false

25end

open Vector;;
let a0 = vempty;;
isvempty a0;;
let a1 = vector [1;2;3;4];;
isvempty a1;;
vlength a1;;
at 3 a1;;
at 4 a1;;
vshow a1;;
```

1～3行目：モジュール Vector を作成し、例外 Empty を宣言している。3行目では空の vector を返す関数を作成したいので let vempty の値を空リストとした。

４～８行目：at のプログラムである。８行目で n 番目の要素を探し出すために n 回再起を行う。n 回再起が終わったとき、のプログラムをするために再帰させるごとに n から１を引く。n が０になったときに値を返す。

９行目：vector のプログラムである。リストを作成したいので let vector lst = lst とした。

１１～１７行目：vlength のプログラムである。再帰させるごとに返す値に１を足し、再起を終了させるときに０とすればよい。

１５～２１行目：vshow のプログラムである。左から順にリストの要素を出力していく（１８～１９行目）。すべての要素を出力し終わったら、() として出力を終了させる。（問題には追加順でデータを出力と書いてありましたが、追加とは逆順に出力しました。）

２２～２４行目：isvempty のプログラムである。引数とした vector が空リストであるか否かを if 文で判別し、true,false を返している。

2.2.1 実行結果

```
module Vector :
  sig
    exception Empty
    val vempty : 'a list
    val at : int -> 'a list -> 'a
    val vector : 'a -> 'a
    val vlength : 'a list -> int
    val vshow : int list -> unit
    val isvempty : 'a list -> bool
  end
# open Vector ;;
# let a0 = vempty ;;
val a0 : 'a list = []
# isvempty a0 ;;
- : bool = true
# let a1 = vector [1;2;3;4] ;;
val a1 : int list = [1; 2; 3; 4]
# isvempty a1 ;;
- : bool = false
# vlength a1 ;;
- : int = 4
# at 3 a1 ;;
- : int = 4
# at 4 a1 ;;
Exception: Vector.Empty.
# vshow a1 ;;
1,2,3,4,
```

図３　実行結果-課題３

2.3 考察

今回の課題ではモジュール内に指定されたプログラムを記述するだけでモジュールを使用する恩恵があまり得られなかったが、同じように長いプログラムを複数回書くことになったときにこのモジュールを記述することでプログラムがかなり簡潔になる。しかし、様々なサイトを見てモジュールについて調べたところモジュールを書くことによってかえってプログラムが複雑化することもあるので、場面に応じてモジュールを使いこなす必要があると考えた。モジュール内のプログラム内容については以前にリスト操作の課題（１回目の課題）とほぼ同じであるためモジュール内の考察は省略する。

3　課題４

抽象データ型 `Student` を定義しなさい．
このデータ型は，学籍番号と名前のデータベースであり，以下の関数を持つものとする．
また内部構造は隠蔽すること．
・`empty`: 引数はなし．データベースを空にする
・`add`: データを追加する関数．第 1 引数は学籍番号，第 2 引数は名前，第 3 引数は対象のデータベース
・`find`: ある名前に対応する学籍番号を返す関数．第 1 引数は名前，第 2 引数は対象のデータベース
・`show`: データベースのすべての要素を表示する関数．第 1 引数は対象のデータベース
・`is_empty`: 対象のデータベースが空であれば `true`，そうでなければ `false` を返す関数．第 1 引数は対象のデータベース

3.1　アルゴリズムの説明

内部構造を隠蔽したいため、それぞれの関数がどの型を引数にとってどの型で出力されているのかを一つ一つ確認し、それをはじめのシグネチャの部分に記述する。

今回宣言した date の型をデータの末尾に記述したい、Nil または Cell(a,b) となるように type を用いて宣言する。

add:lst の先頭にあるデータを追加したいため、アルゴリズムは課題２の shift と同様である。

find: ある名前に対応する学籍番号を返したいのでアルゴリズは課題２の indexOf と同様である。

show: 要素として格納されているデータをすべて出力したいのでアルゴリズムは課題３の vshow と同様である。

is_empty: データの中身が空かどうかで bool 型で出力したい。アルゴリズムは課題３の vempty と同様である。

3.2　プログラムの説明

```
1 module Student:
2 sig
3   exception Not_found
4   type date
5   val empty : date
6   val add : int -> string -> date -> date
7   val find : string -> date -> int
8   val show : date -> unit
9   val is_empty : date -> bool
10 end
11 =  struct
```

```
12  exception Not_found

13  type date = Nil | Cell of int * string * date

14  let empty = Nil

15  let rec add num name lst =
16    match lst with
17      Nil -> Cell (num , name , Nil)
18
19    |Cell (num' , name' , rest) -> Cell (num , name , lst)

20  let rec find name lst =
21    match lst with
22      Nil-> raise Not_found
23    |Cell (num , name', rest) ->
24        if name' = name then num
25        else find name rest

26  let rec show lst =
27    let rec print_date lst =
28      match lst with
29        Nil -> print_string ":"
30      | Cell (num , name , rest) ->
31          print_int num ; print_string ":" ; print_string name ;
32          print_newline () ; print_date rest
33    in print_date lst

34  let is_empty lst =
35    if lst = empty then true
36    else false
37 end;;


open Student;;
let a0 = empty;;

let a1 = add 1 "takimoto" a0;;
```

```
let a2 = add 2 "matsuzawa" a1;;

let a3 = add 3 "katsurada" a2;;

find "katsurada" a3;;

find "takeda" a3;;

show a3;;
```

１～１０行目：モジュール student を作成し、隠蔽したい部分をシグネチャに記述している。新たな型 date を type によって定義する。add は学籍番号、名前、データベースを入力し、新たなデータベースが出力されるので int-¿string-¿date-¿date となる。find は名前、データベースを入力して学籍番号を返すので string-¿date-¿int となる。同様に show,is_empty もシグネチャ内に記述する。

１２行目以降がモジュールの本体となる関数の宣言である。

１３～１４行目：date の型を Nil または Cell(学籍番号、名前、他のデータ) となるように新たな型を宣言している。１４行目では空のデータベースを作成する関数であるため、let empty = Nil とした。

１５～１９行目：add のプログラムである。指示された通りに引数を設定して、既存のデータベースと、新たに追加したい num' と nam' を新たな Cell の要素として Cell に加える（１９行目)。既存のデータベースが存在せず、lst の中身が Nil であったときは別で Cell(num,name,Nil) と処理している（１７行目)。

２０～２５行目：find のプログラムである。調べたい名前 name に Cell の要素である name' が一致した場合、学籍番号を返したいので num を出力する（２３～２４行目)。name が name' に一致しなかった場合はさらに Cell の内部にある要素の調査をしたいので find name rest とした（２５行目)。データベースのすべての要素について調査をし、一致するものが見つからなかった場合は、要素 Nil が出てくるのでその場合１２行目で宣言したエラー Not_found を返す。

２６～３３行目：show のプログラムである。課題内容に書かれていた出力するプログラムを囲繞して記述した（３１～３２行目)。この時出力結果を見やすく出力したかったため、学籍番号と名前の間に：を挿入したり、各データごとに改行を施す print_newline'() を記述した。プログラムの内容は課題３の vshow とほぼ同じである。

３４～３６行目：１４行目で作成した関数 empty が空かどうかを bool 型を用いて返す関数である。プログラムの構造は課題３の isvempty とほぼ同じである。

3.2.1 実行結果

```
module Vector :
  sig
    exception Empty
    val vempty : 'a list
    val at : int -> 'a list -> 'a
    val vector : 'a -> 'a
    val vlength : 'a list -> int
    val vshow : int list -> unit
    val isvempty : 'a list -> bool
  end
# open Vector ;;
# let a0 = vempty ;;
val a0 : 'a list = []
# isvempty a0 ;;
- : bool = true
# let a1 = vector [1;2;3;4] ;;
val a1 : int list = [1; 2; 3; 4]
# isvempty a1 ;;
- : bool = false
# vlength a1 ;;
- : int = 4
# at 3 a1 ;;
- : int = 4
# at 4 a1 ;;
Exception: Vector.Empty.
# vshow a1 ;;
1,2,3,4,
```

図 4　実行結果-課題 4

3.3 考察

モジュール内に自分でシグネチャを記述することでコンストラクタの Nil が使えてしまったり、勝手に値を取り出せてしまうことを防いでいる。そのほかにも今回の課題のようにモジュール内にいくつかの関数を記述し、学籍番号と名前の一覧を最終的に記述したいときにそれを出力するまでの過程を隠蔽し、そのモジュールを実行した人が欲しいものだけを出力することを可能にしている。

またシグネチャ内に隠蔽したいものを記述することで、モジュールの外側からは内部のリスト構造にアクセスできなくなる。

問題文に show 関数で出力する順番は余裕があれば学籍番号順にするとありましたが、レポートの提出期限が迫っており、プログラムする時間がありませんでした。しかしそのアルゴリズムは、以前の課題にもあったソートのアルゴリズムを使うのではないかと考えた。Cell の中の要素には学籍番号と名前が含まれている。Cell の一番左側の要素が学籍番号であるのその値をそれぞれ大小比較していき、ソートすればよいと考えた。その際、実行時間を短縮させるためにも実行時間の短いクイックソートのアルゴリズムを使うのが最適なのではないかと考えた。