* pseudo code for getMinPos "口語"

```
getMinPos
    (integer array arr, integer len)
    minpos ← 0
    for i ← 1 to len-1 do
        if arr[i] smaller than arr[minpos] then
            minpos ← i
    return minpos
```

* bad pseudo code: too detailed (長舌)

```
a ← arr[i]
b ← arr[minpos]
if a smaller than b then
    minpos ← i
```

* bad pseudo code: vars that only you know or incorrect (火星話)

```
a ← 0
for b ← 1 to len-1
    if arr[b] smaller than arr[a] then
        b ← a
return minpos
```

* bad pseudo code: too abstract (惜字如金) 文言

run a loop that updates minpos in every iteration

* trade-off : the selection sort algorithm

```
selSort
  (integer array arr, integer len)

  for i ← 0 to len-1 do
    /* find minimum index between arr[i] ~ arr[len-1] */
    min ← getMinPos ( arr[i] to arr[len-1] )
    /* put arr[minimum index] at arr[i] */
    swap (arr[i], arr[min])
```

sufficient if you are talking to usual programmers

* data 資料　　structure 收納方式 (結構)

食譜 : 調味料結構
樂譜 : 樂團 結構
陣譜 : 兵力 結構
程式譜 : 資料 結構

* 200 份考卷 (data), 如何 "排列" (收納)?
  - 隨便
  - 最高分 → 最低分
  - 按學號
  - 依尾數分 + 份
  - ...

* basic "operations" on a data structure
  - construct ("排")
  - get Blah Blah (看), (找)
  - put Blah Blah (加)
  - remove Blah Blah (減)

) dynamic : "maintain" (收拾)

* Unordered array
  - construct : nothing
  - get By Index
    get By Value
    get Min , get Min Pos
  - put By Index , put Anywhere
  - remove By Index
    remove Min

* ordered array
  - construct : selection sort ?
  - get Min : easy
  - put : need "insert"
  - remove Min : need "move others"

* why data structures and algorithms

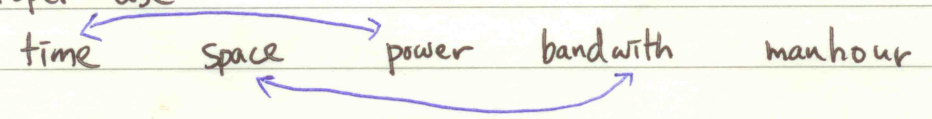
storage resources <---> computation resources

memory, HD, HD"s"? CPU, GPU, ..., machine"s"?

use resources properly

good program

* proper use:



- "all important", but different platforms have different constraints

- trade-off of different factors

e.g. "easy program" (low manhour) ⟺ "slow" (large CPU time)

- trade-off of data structure operations

e.g. "more organized" (careful/slow maintain) ⟺ "more efficient find" (fast get)

* Programming ≠ Coding

programming ≈ building house

requirement
analysis
design
refinement and coding
verification { proof
test/debug

*	Intro to C	this class
requirement	simple	simple
analysis	simple	simple
design	simple	☆
coding	☆	○
proof	none	○
test	simple	☆
debug	☆	○