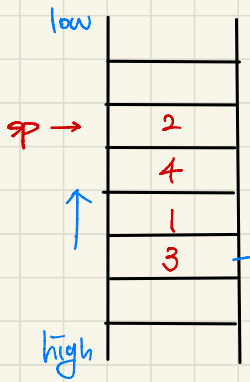
<Quicksort>

① recieve inputs, and put them into the stack.

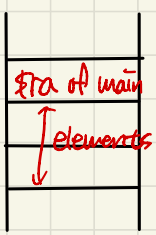


ex) N=4, A[] = {3, 1, 4, 2}

save the value of (N-1) in $s1

→ Save this address in $s0

② save $ra of main into the stack, and call quicksort.

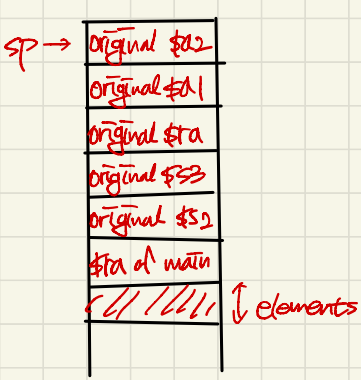


When call quicksort...

- $a1 : low (for first, $a1 = 0)
- $a2 : high. (for first, $a2 = N-1)

③ (In quicksort) save original $ra, $a1, $a2, $s2, $s3.

$a1 → low, $a2 → high, $s2 → mid_left, $s3 → mid_right.



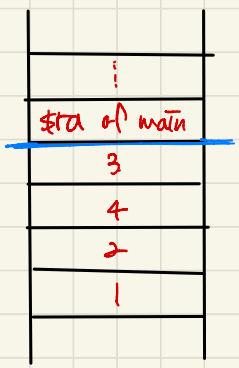
④ (In quicksort) If low >= high, load original $ra, $a1, $a2, $s2, $s3 and return / Else, call partition.

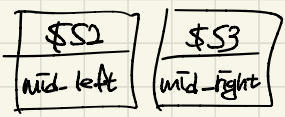
⑤ (In partition) $a1 is still low and $a2 is still high.

Do partition and save mid_left to $s2, and mid_right to $s3

[3 1 4 2] → [1 2 4 3]

after first partition (pivot=2)





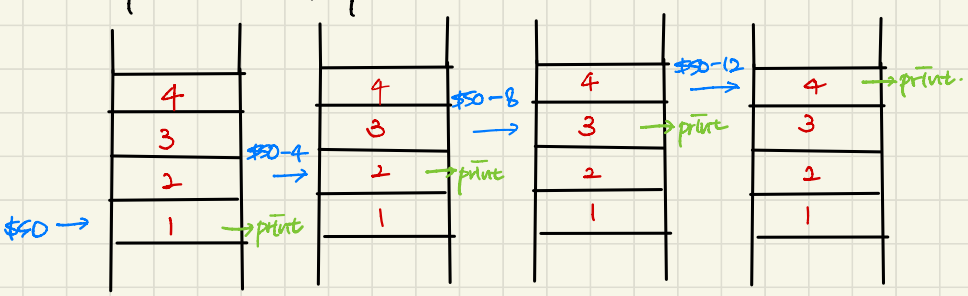
and go back to $ra (quicksort)

⑥ (In quicksort) Call quicksort with $a1 = low, $a2 = mid_left - 1

⑦ (In quicksort) Call quicksort with $a1 = mid_right + 1, $a2 = high.

⑧ (In main) After finish quicksort, call print_stack.

⑨ (In print_stack) print the item in the stack



* offset ⇒ $50 - 4x(index)

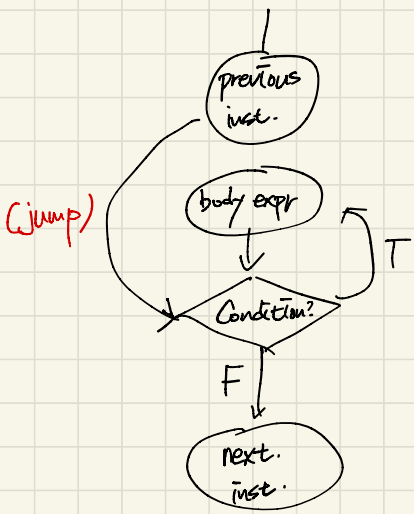
↳ 0, 1, 2, ···, N-1

saved in $s1

* print : (print_int (item) + print_string ("\n"))

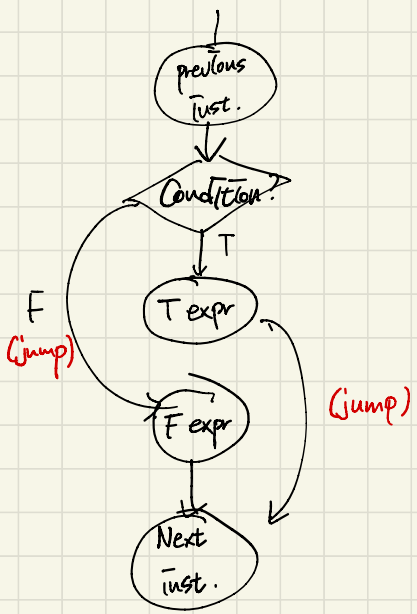
⑩ (In main) After finish printing, load $ra of main to $ra and return (end of program)

* while logic



⇔ while (condition) {
  body expr;
}

* if logic.



⇔ if (condition) {
  T expr;
} else {
  F expr;
}