# CGJ008

## Big-O Quiz

|  |  |  |  |
| --- | --- | --- | --- |
| **Function** | **f(n)** |  | **O()** |
| A | 2n+1 | 2n | O(n) |
| B | 2n | 2n | O(n) |
| C |  |  | O |
| D |  |  | O |
| E | 1 | 1 | O(1) |
| F |  |  | O(n) |
| G | 2n+2 | 2n | O(n) |
| H | ⌈Log₂ n⌉ + 1 | Log₂ n | O(Log₂ n) |
| I |  | 2n | O(n) |
| J |  |  | O |
| K |  |  | O() |
| L |  | Log₂ n | O(Log₂ n) |
| M |  |  | O |
| N |  |  | O |
| O |  |  | O |
| P |  |  | O |
| Q |  |  | O |
| R |  |  | O(n) |
| S |  |  | O |
| T |  |  | O |
| U |  |  | O |
| V |  | 2n | O(n) |
| W |  | 2n | O(n) |
| X |  |  | O |
| Y |  |  |  |
| Z |  |  |  |

## Union Find

c) In this implementation (using an integer array, id[]), it is not possible. We need to set the lesser root of p and q as root of the combined set, to keep track of the oldest account. Which excludes the option to always choose the tallest tree as root.

If we were to solve the problem using node objects in the tree structure, one could perform weighted union, swapping the values of the root nodes if the lesser root is the top of the shorter tree.

### d) Scheme

union(n-1, n-2), then union(n-2, n-3), then union(n-3, n-4) … union(n-(n-1), n-n). Now a call to find(n-1) will access every element of the array. m = n.

### Analysis

Each call of union() with p and q being root or single elements, will access the array 3 times. Making such calls n-1 times to create the tallest possible tree for n elements, means 3(n-1) accesses to the array. Adding a call to   
find(n-1) increases m to be equal to n, and the find()-call costs 2n-1 accesses. Hence:

|  |  |  |
| --- | --- | --- |
|  |  |  |