NANYANG TECHNOLOGICAL UNIVERSITY

SC2001/ CX2101: Algorithm Design and Analysis

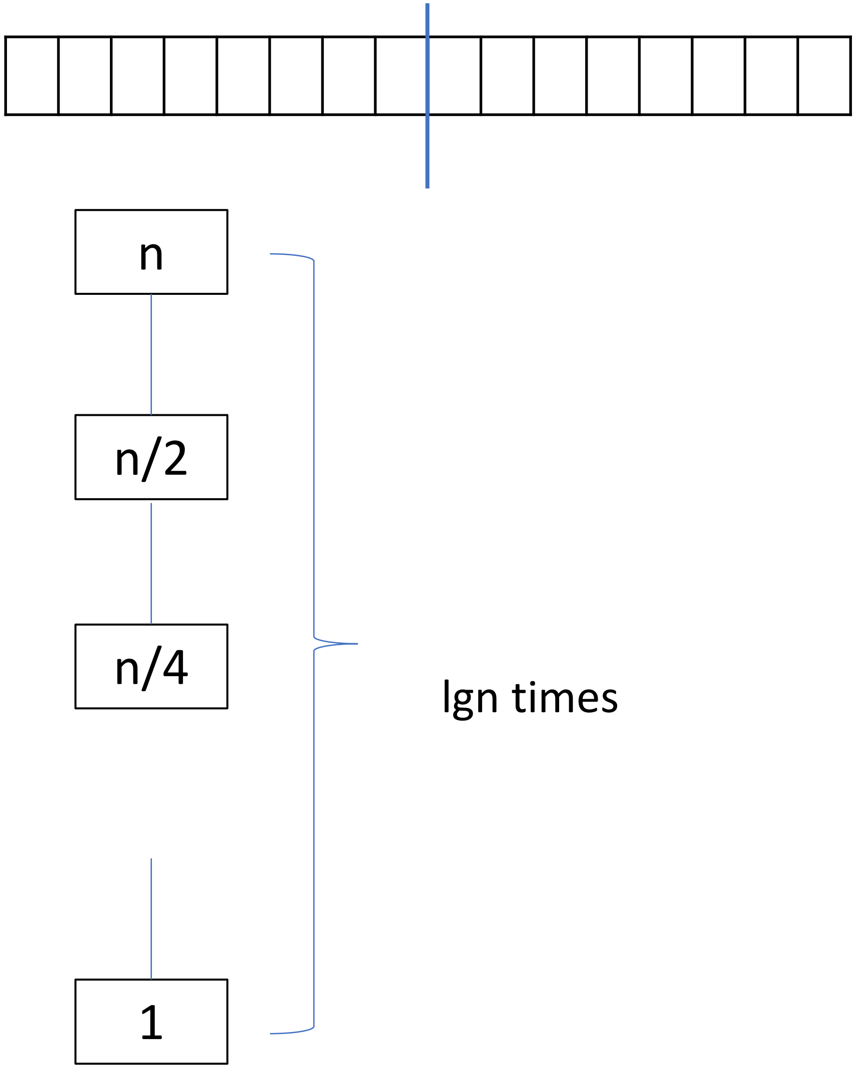
Week 9

Huang Shell Ying

Solving Recurrences – Substitution method

References for guess:

<u>Problem size is reduced by a constant factor</u> .



Binary search:

$W(n) = W(n/2) + 1$,

$W(1) = 1$

$W(n) = O(\lg n)$

Solving Recurrences – Substitution method

References for guess:

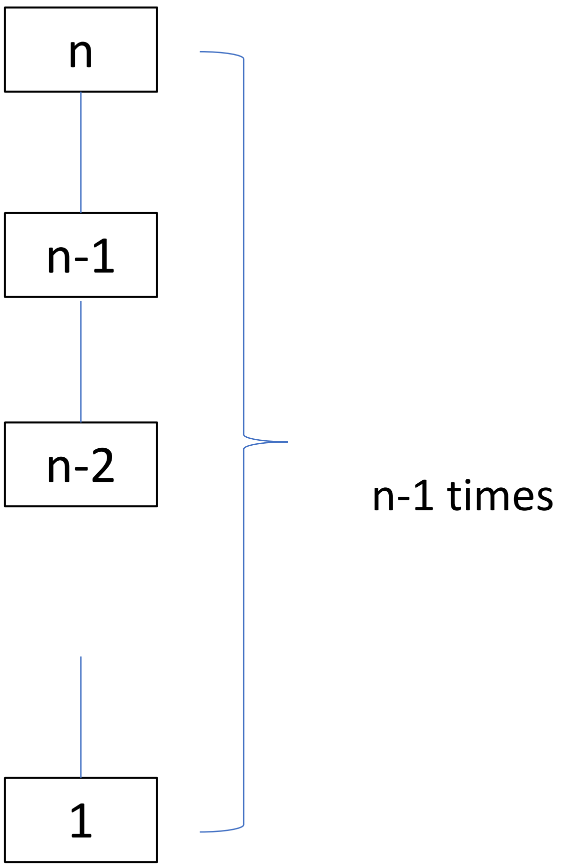
Problem size is reduced by a constant value

Insertion sort (best case)

W(n) = W(n-1) **+ 1**,

W(1) = 0

W(n) = O(n)



Solving Recurrences – Substitution method

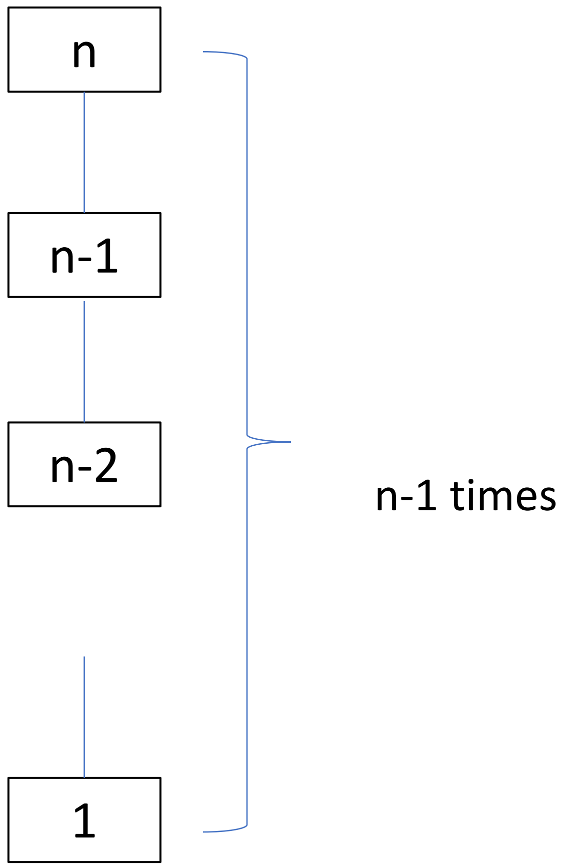
References for guess:

QuickSort (worst case):

$W(n) = W(n\text{-}1)$ **+ n – 1**,

$W(1) = 0$

$W(n) = O(n^2)$



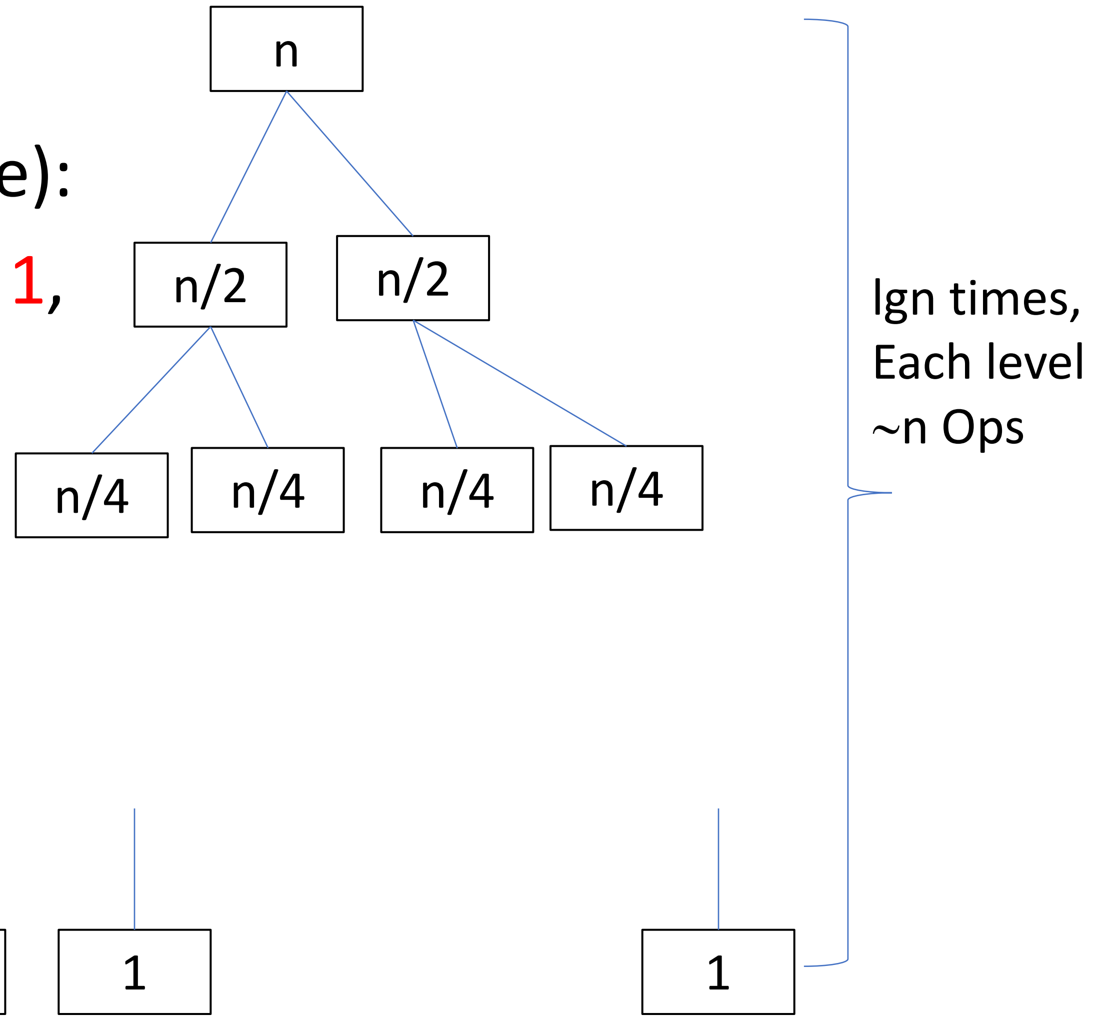
Solving Recurrences – Substitution method

References for guess:

MergeSort (worst case):

$W(n) = 2W(n/2) + n - 1,$

$W(1) = 0$



$W(n) = O(n\lg n)$

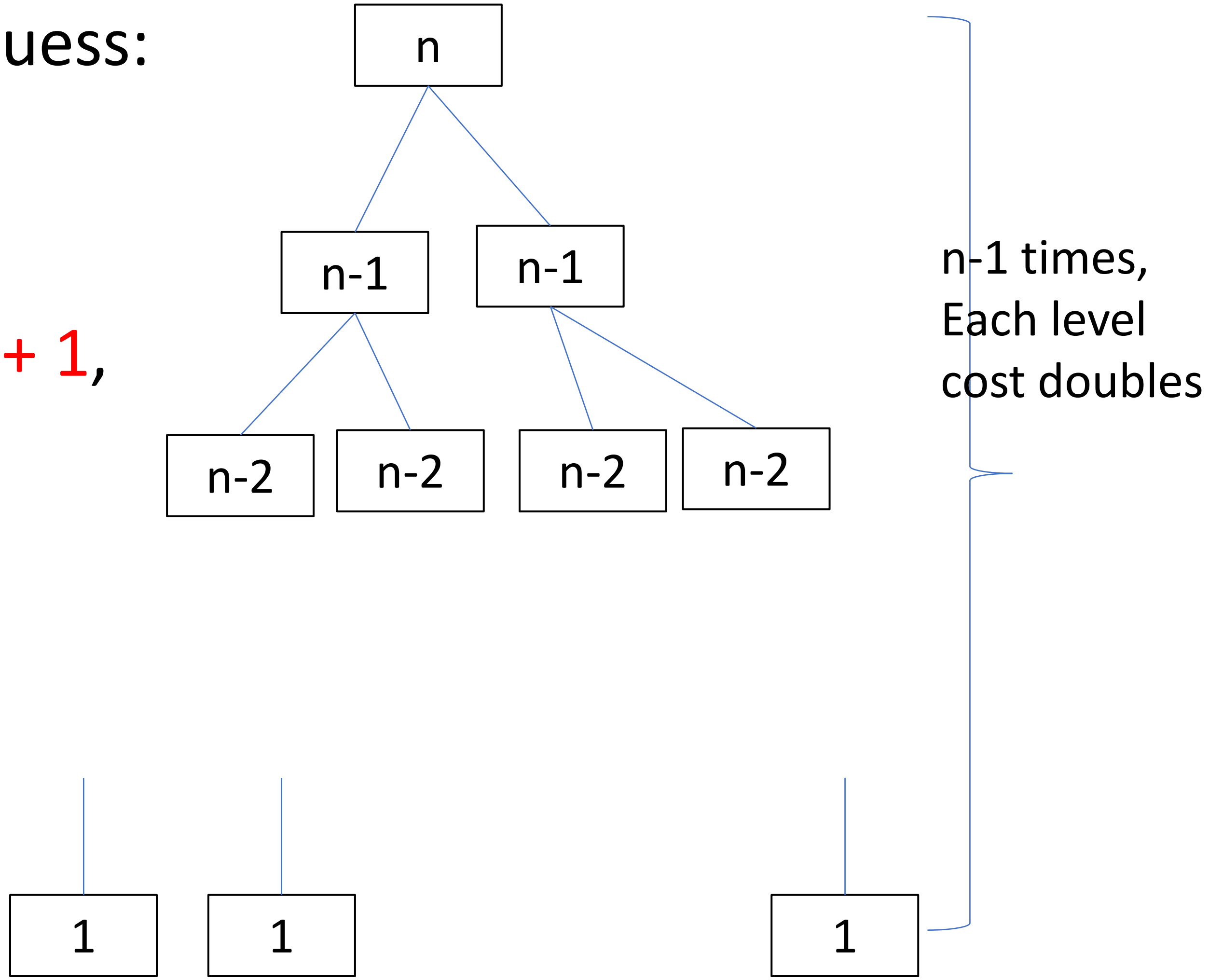
Solving Recurrences – Substitution method

References for guess:

Tower of Hanoi

$W(n) = 2W(n-1) + 1$,

$W(1) = 0$



$W(n) = O(2^n)$

Dynamic Programming

- What is dynamic programming?
  - Divide a problem into subproblems, but these subproblems are ***overlapping***
  - Solve each subproblem recursively
  - Combine the solutions to subproblems into a solution for the given problem
  - ***Do not compute the answer to the same subproblem more than once***
  - Use a data structure to remember the solutions of subproblems

Dynamic Programming

Consider the Virahanka number V(n) defined by the following equations:

$V(n) = 1$, when $n = 0$ or 1;

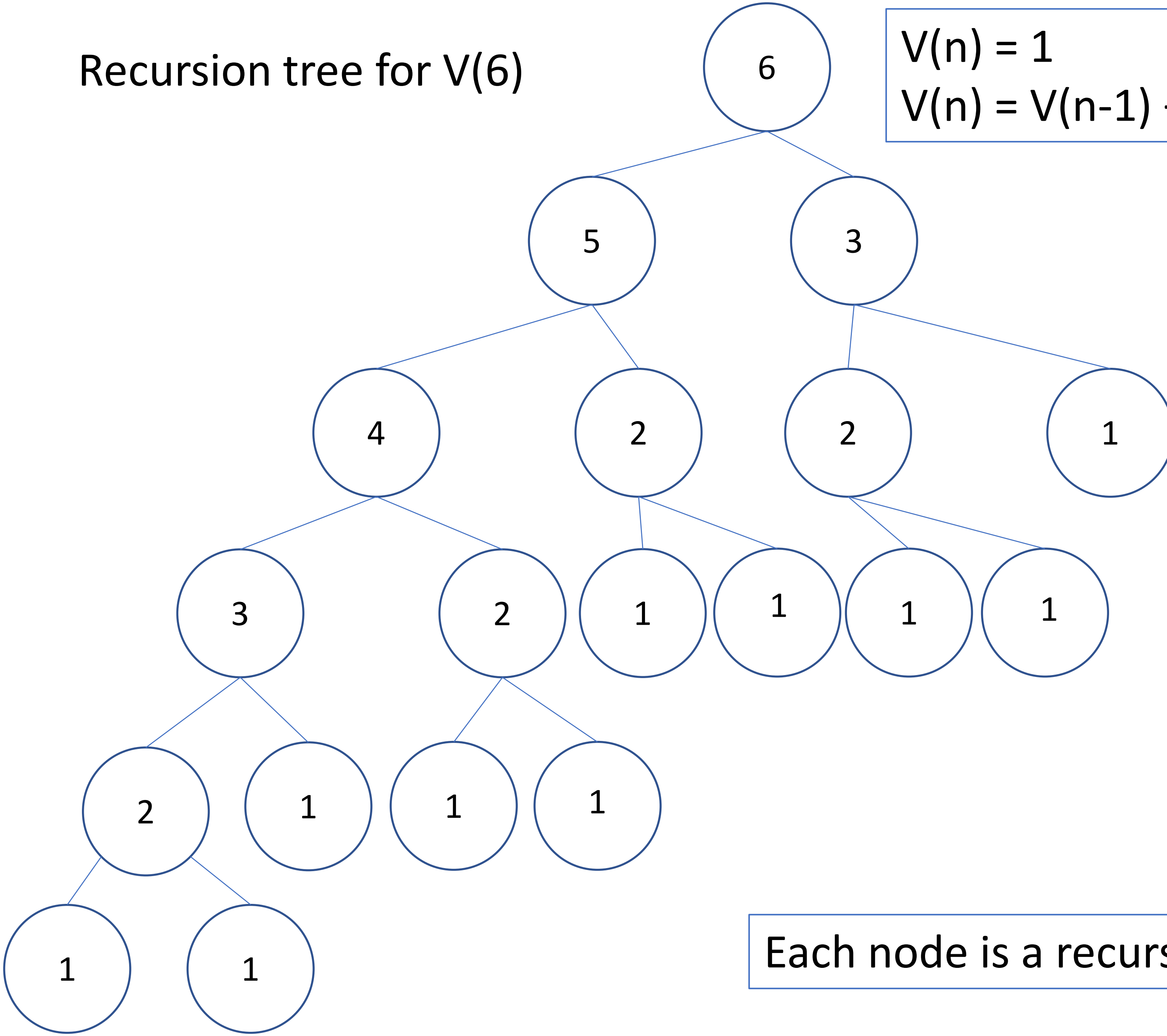
$V(n) = V(n-1) + V(\lfloor n/2 \rfloor)$, when $n>1$;

(here $\lfloor n/2 \rfloor$ is the floor function for n/2. E.g. $\lfloor 3/2 \rfloor = \lfloor 2/2 \rfloor = 1$).

Recursion tree for V(6)

$V(n) = 1$

$V(n) = V(n-1) + V(\lfloor n/2 \rfloor)$



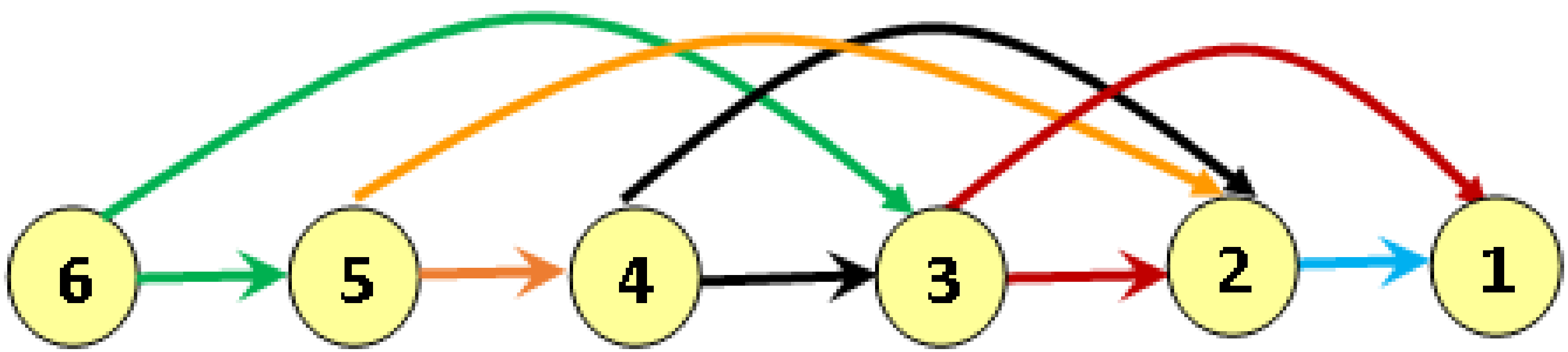
Each node is a recursive call

Dynamic Programming

$V(n) = 1$, when $n = 0$ or 1;

$V(n) = V(n-1) + V(\lfloor n/2 \rfloor)$, when $n>1$;

(i) Draw the subproblem graph for V(6).



Each node is a (sub)problem.

Dynamic Programming

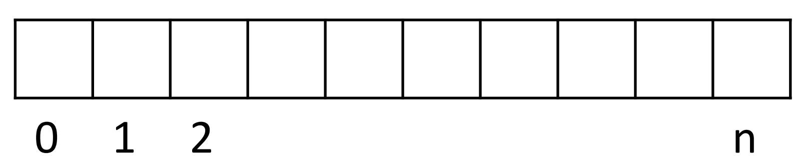
$V(n) = 1$, when $n = 0$ or 1;

$V(n) = V(n-1) + V(\lfloor n/2 \rfloor)$, when $n>1$;

(ii)Give dynamic programming algorithm to compute V(n) using the bottom up approach.

```
v[0] = v[1] = 1
For (j= 2 to n)
    v[j] = v[j-1] + v[j/2];
return v[n];
```

Array V:



Dynamic Programming

$V(n) = 1$, when $n = 0$ or 1;

$V(n) = V(n-1) + V(\lfloor n/2 \rfloor)$, when $n>1$;

(iii) Give dynamic programming algorithm to compute $V(n)$ using the top down approach.

```
// v array initialized to all -1
int DP_V(n)
{
    If (n == 0 or n == 1) { v[n] = 1; return 1 }
    If (v[n-1] == -1)
        v1 = DP_V(n-1)
    else v1 = v[n-1];
    If (v[n/2] == -1)
        v2 = DP_V(n/2)
    else v2 = v[n/2];
    v[n] = v1 + v2;
    Return v[n];
}
```