Analysis and Design of Algorithms

CS3230

Week 13

Summary: Part 1

Diptarka Chakraborty

Ken Sung

Revision

- Correctness proof
- Time analysis
- Lower bound
- Divide-and-conquer
- Amortization

Question 2.1

- The correctness of recursive algorithm can be proved by induction.
- Below shows the 4 steps required for an inductive proof. Can you give the correct ordering of these steps?

- (a) Show that the base case P(1) is true
- (b) Given the inductive hypothesis, show that P(n) is true
- (c) State the predicate P(n)
- (d) State the inductive hypothesis [P(j) is true for j<n]

Solution

- This is a strong induction proof.

- The order should be:
  - (c) State the predicate P(n)
  - (a) Show that the base case P(1) is true
  - (d) State the inductive hypothesis [P(j) is true for j<n]
  - (b) Given the inductive hypothesis, show that P(n) is true

Question 2.2 [Term test 2017]

- Consider an array of n elements A[1..n]. Suppose there are k distinct elements in A[1..n]. Under the comparison model (i.e. for any two elements a and b, you can determine if a<b, a=b or a>b), what is the lower bound to sort the elements in A[1..n]?

Answer

- Let n_i be the number of occurrences of the i^{th} element in A[1..n].
- The number of possible permutations is $\binom{n}{n_1 \dots n_k}$.
- The height of the decision tree is $\log \binom{n}{n_1 \dots n_k}$.
- $\log \binom{n}{n_1 \dots n_k} = n \log n - \sum_{i=1}^{k} n_i \log n_i \geq n \log n - \sum_{i=1}^{k} \frac{n}{k} \log \frac{n}{k} = n \log k$.
- Hence, the lower bound is $\Omega(n \log k)$.

Question 2.3

- Can you give a tight bound for $\sum_{i=1}^{n} i \lg i$?

- 1. $\Theta(n^2)$
- 2. $\Theta(n^2 \lg n)$
- 3. $\Theta(n \lg n)$
- 4. $\Theta(n^2 \lg \lg n)$

Solution

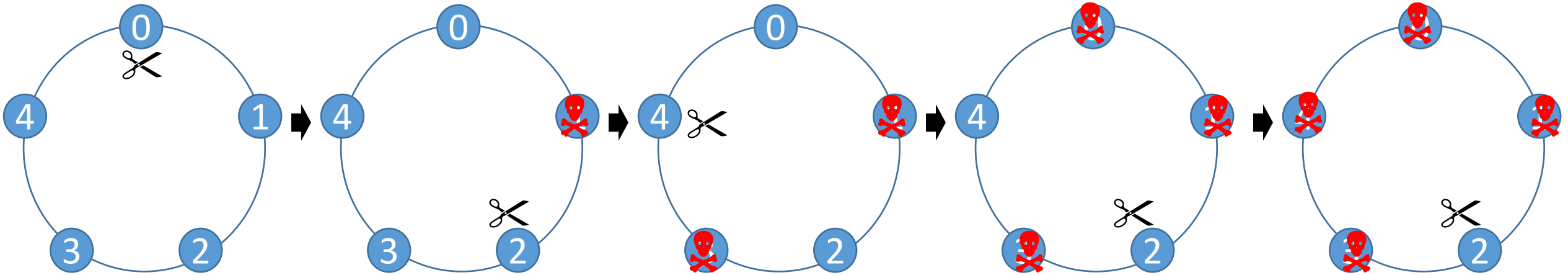
- $\sum_{i=1}^{n} i \lg i \leq \sum_{i=1}^{n} i \lg n \leq \frac{n^2}{2} \lg n.$
  - Hence, $\sum_{i=1}^{n} i \lg i = O(n^2 \lg n).$

- $\sum_{i=1}^{n} i \lg i \geq \sum_{i=\frac{n}{2}}^{n} i \lg i \geq \sum_{i=\frac{n}{2}}^{n} i \lg \left(\frac{n}{2}\right) \geq \left(\frac{n}{2}\right)^2 \lg \left(\frac{n}{2}\right)$
  - Hence, $\sum_{i=1}^{n} i \lg i = \Omega(n^2 \lg n).$

- In conclusion, $\sum_{i=1}^{n} i \lg i = \Theta(n^2 \lg n)$

Question [Josephus problem] (Term test 2017)

- n people decided to commit a mass suicide. These n people are arranged in a circle and they are labeled as 0, 1, 2, …, n-1.
- The knife is passed around the circle starting from the person with label 0. For every alternative person who receive the knife, he/she will kill himself/herself. The process is repeated until all people are killed.
- Let J(n) be the index of the person who is the last to kill. (For example, suppose n=5, the suicide order is 1, 3, 0, 4, 2. So, J(5)=2.)

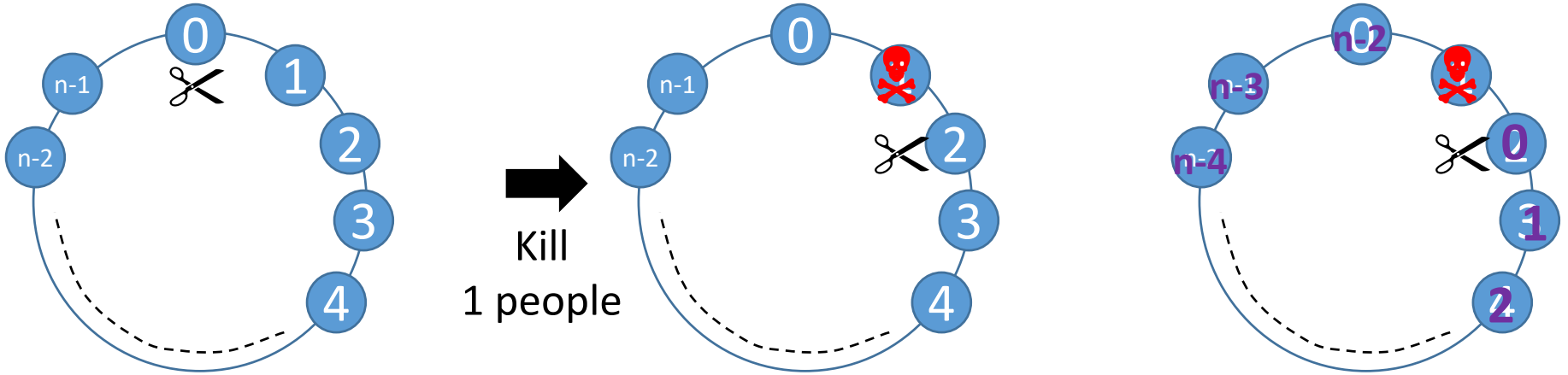


Question 2.4 [Josephus problem] (Term test 2017)

- Can you express J(n) as a recursive equation in term of J(n-1)?
- Given this recursive formula, can you give a recursive algorithm? What is the running time?

Answer

- Base case: J(1) = 0
- Recursive case: $J(n) = \begin{cases} 0 & \text{if } J(n-1) = n-2 \\ J(n-1)+2 & \text{otherwise} \end{cases}$



Answer

- Base case: J(1) = 0
- Recursive case: $J(n) = \begin{cases} 0 & \text{if } J(n-1) = n-2 \\ J(n-1)+2 & \text{otherwise} \end{cases}$

Josephus1(n)

1. If (n==1), then return 0;
2. v = Josephus1(n-1)
3. If (v==n-2), then return 0; else return (v+2);

- T(n) = T(n-1) + 1
- Running time = T(n) = O(n)

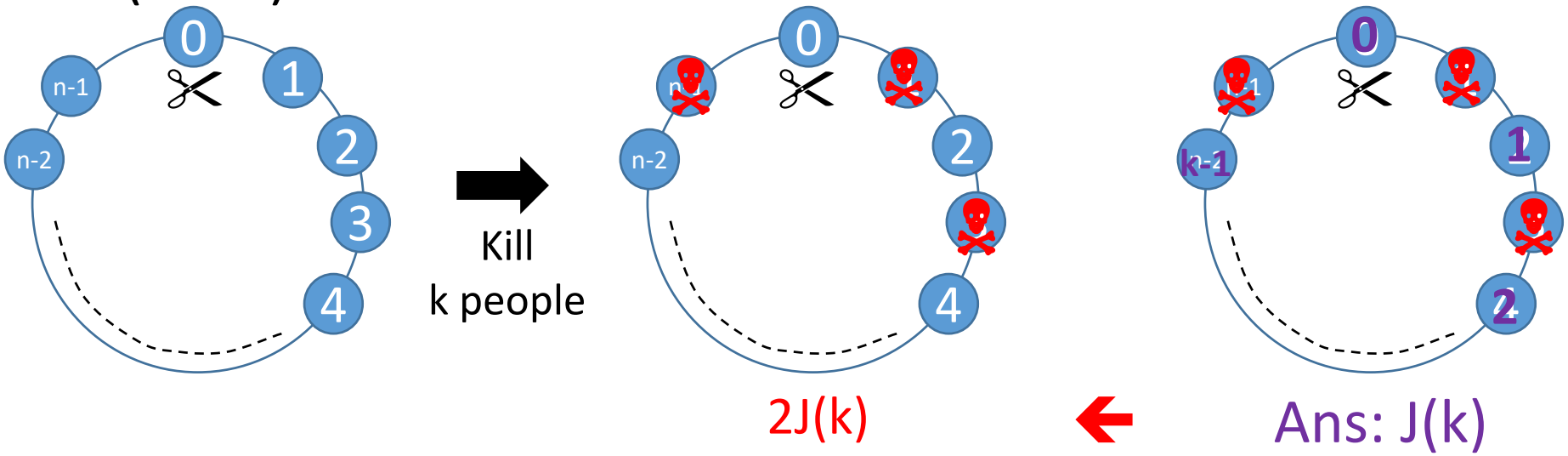
Question 2.5 [Josephus problem] (Term test 2017)

- Can you express J(n) as a recursive equation in term of J(n/2) when (1) n is even or (2) n is odd?
- By these two recursive formula, can you give an efficient algorithm? What is the running time?

Answer

- Base case: J(1)=0
- Recursive case: $$J(2k) = 2J(k)$$ $$J(2k+1) = 2(J(k)+1)$$

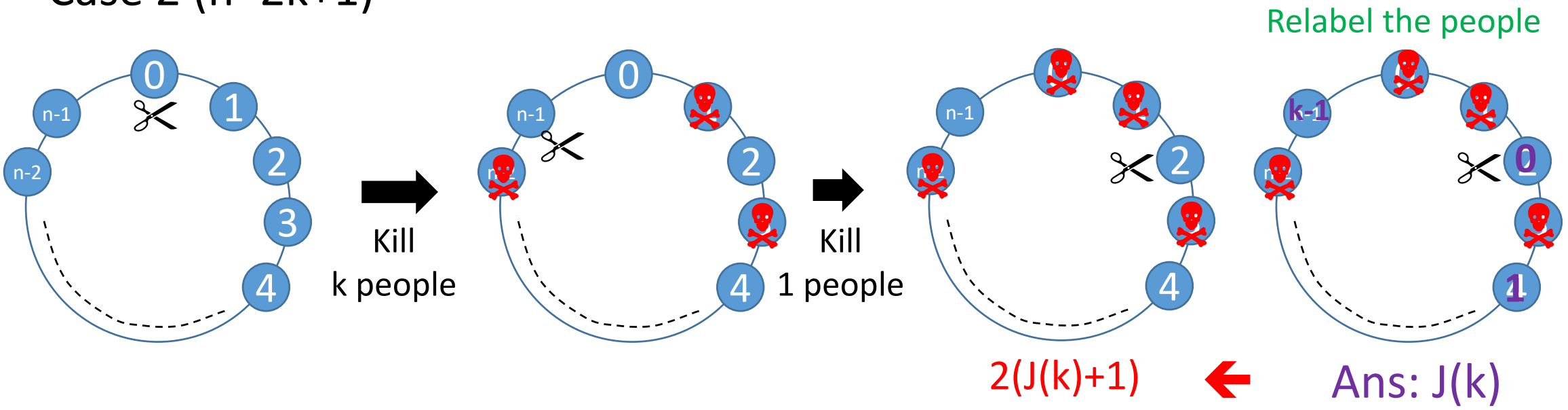
Case 1 (n=2k)



Answer

- Base case: J(1)=0
- Recursive case: $$J(2k) = 2J(k)$$ $$J(2k+1) = 2(J(k)+1)$$

Case 2 (n=2k+1)



Answer

- Base case: $J(1) = 0$
- Recursive case: $J(2k) = 2J(k)$
  $J(2k+1) = 2(J(k)+1)$

Josephus2(n)

1. If n==1, then return 0;
2. If n is even,
   Return 2*Josephus2(n/2);
   Else
   Return 2*(Josephus2((n-1)/2)+1);

- T(n) = T(n/2)+1.
- Hence, running time = T(n) = O(log n).

Final Answer:
J(n) can be computed in O(1) time.

- Lemma: $J(n) = 2\left(\mathrm{n} - 2^{\lfloor \lg n \rfloor}\right)$ for n≥1.
- Proof:
  - Base case: For n=1, $2\left(1 - 2^{\lfloor \lg 1 \rfloor}\right) = 0 = J(1)$.
  - Recursive case:
    - Suppose $J(p) = 2\left(\mathrm{n} - 2^{\lfloor \lg p \rfloor}\right)$ for p<n.
    - Case 1: n=2k (even)
      - Note that $\mathrm{n} - 2^{\lfloor \lg n \rfloor} = 2\mathrm{k} - 2^{\lfloor \lg(2k) \rfloor} = 2\mathrm{k} - 2^{\lfloor \lg k \rfloor + 1} = 2\left(k - 2^{\lfloor \lg k \rfloor}\right)$.
      - Since $J(2k) = 2J(k)$,
        - $J(n) = J(2k) = 2J(k) = 2 \cdot 2\left(k - 2^{\lfloor \lg k \rfloor}\right) = 2\left(\mathrm{n} - 2^{\lfloor \lg n \rfloor}\right)$.
    - Case 2: n=2k+1 (odd)
      - Note that $\mathrm{n} - 2^{\lfloor \lg n \rfloor} = (2\mathrm{k} + 1) - 2^{\lfloor \lg(2k+1) \rfloor} = (2\mathrm{k} + 1) - 2^{\lfloor \lg k \rfloor + 1} = 2\left(k - 2^{\lfloor \lg k \rfloor}\right) + 1$.
      - Since $J(2k + 1) = 2(J(k) + 1)$,
        - $J(n) = J(2k + 1) = 2(J(k) + 1) = 2\left(2\left(k - 2^{\lfloor \lg k \rfloor}\right) + 1\right) = 2\left(\mathrm{n} - 2^{\lfloor \lg n \rfloor}\right)$.
  - By induction, the lemma follows.

Example:

$$J(5) = 2\left(5 - 2^{\lfloor \lg 5 \rfloor}\right) = 2$$

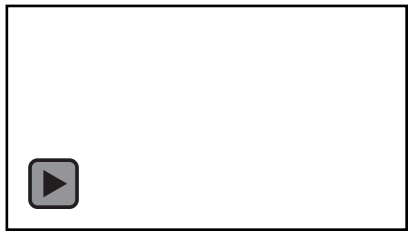
- Base case: $J(1) = 0$
- Recursive case: $J(2k) = 2J(k)$, $J(2k + 1) = 2(J(k) + 1)$

Question 2.6

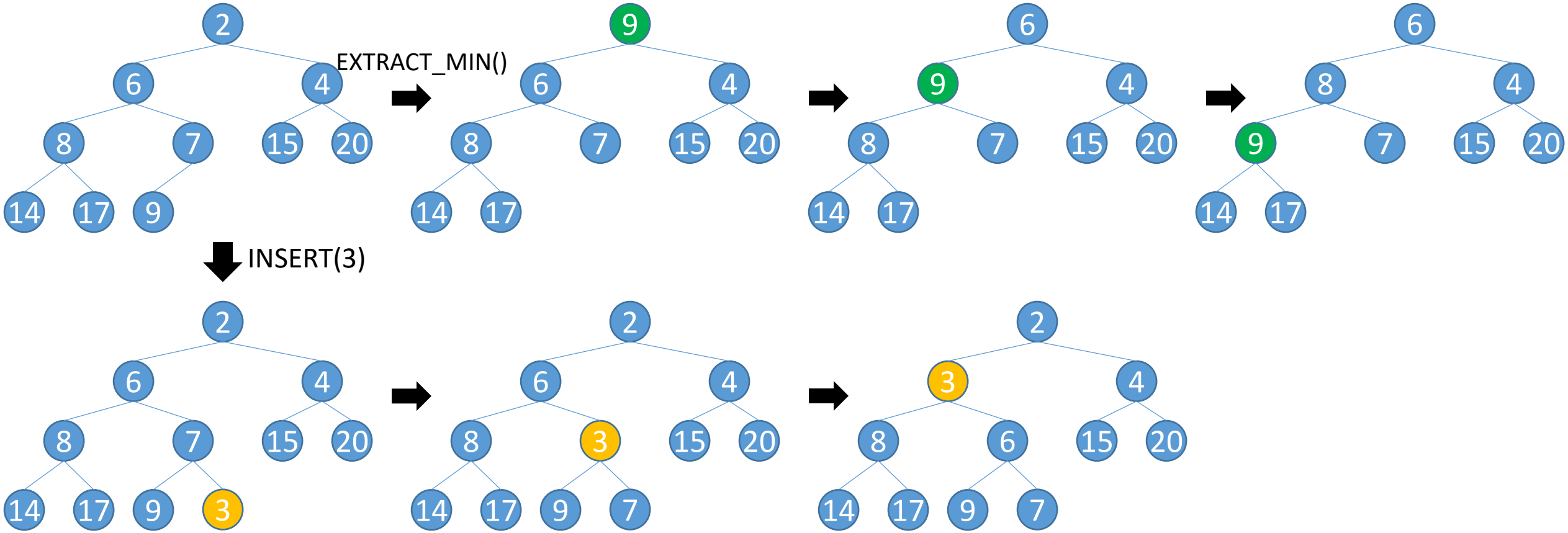
- We implement a binary min-heap data-structure with n elements allowing the following operations:
  - INSERT(x): Insert x into the heap
  - EXTRACT_MIN(): Delete the minimum element in the heap

- We know that both INSERT(x) and EXTRACT_MIN() run in O(log n) worst-case time.

- Suppose we start with an empty heap. Can you show that INSERT(x) runs in O(log n) amortized time and EXTRACT_MIN() runs in O(1) amortized time?
- (You can use either accounting or potential method.)



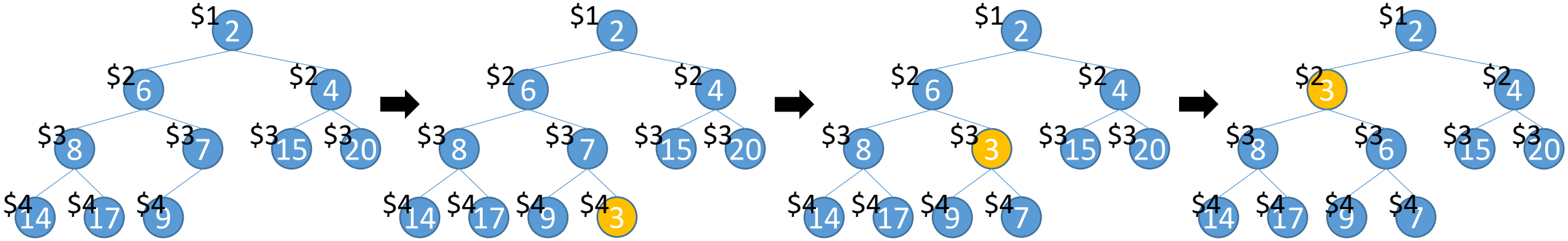
EXTRACT_MIN() and INSERT(x)



Answer

- Accounting method:
  - INSERT(x): We create a node with element x at depth d (d=lg n). Allocate $d to it. Then, move x up the heap until it satisfies the heap property.
  - INSERT(x) requires $2d = O(2 lg n).

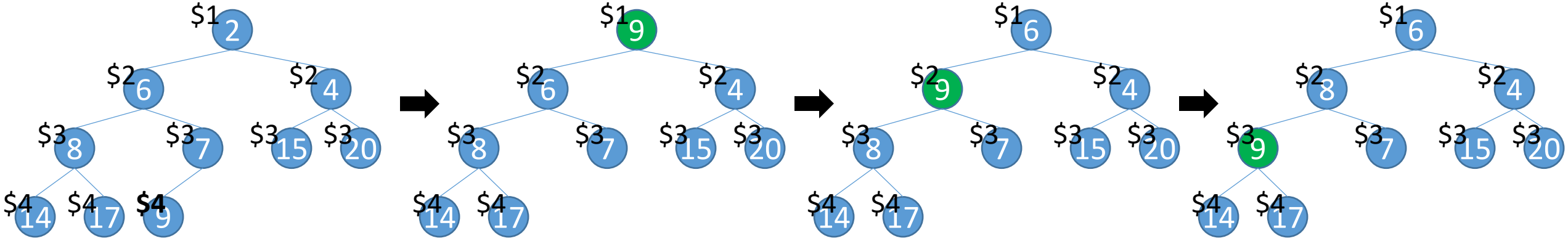
INSERT(3):



Answer

- Accounting method:
  - EXTRACT_MIN(): (i) Delete the lowest node v (of depth d) and obtain $d. (ii) Replace the root by v in the heap. (iii) Use $d for the heapify operation.
  - (i)+(ii) takes O(1) amortized time. (iii) is free of charge.

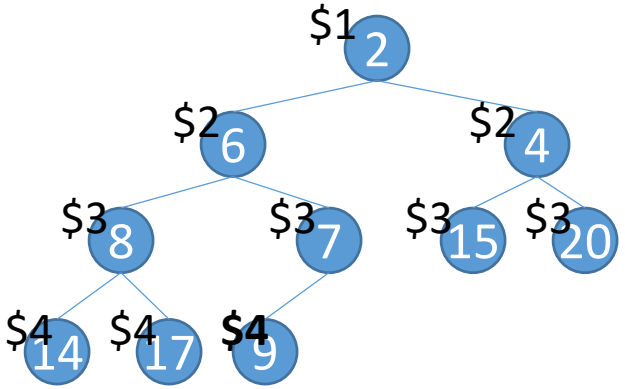
EXTRACT_MIN():



Answer

- Potential method.
- $\Phi(D_i)$=sum of depths of all nodes.
- Example: $\Phi(D_i)=2^0*1+2^1*2+2^2*3+3*4=29$.

- The detail is as EXERCISE.



Summary: Analysis

- Correctness proof
  - Correctness of iterative algorithm: Invariant
  - Correctness of recursive algorithm: Induction
- Worst case analysis
  - Substitution method
  - Telescoping method
  - Recursion tree
  - Master methodMaster Theorem
- Lower bound for comparison model
- Randomization
  - Indicator variable
  - Linearity of expectation
- Amortized analysis
  - Accounting method
  - Potential method
- NP-hardness
  - Polynomial time reduction
  - Certificate
- Approximation
  - Bound the approximation algorithm by the optimal solution

Summary: Design

- Three most used algorithm tricks:
  - Search
    - Binary search
  - Sorting
    - Randomized QuickSort
    - Linear Sorting
  - Select
    - Randomized QuickSelect
    - Worst-Case linear select

Summary: Design
Steps for solving optimization problem?

1. Identify the optimal substructure
2. Formulate **recursive equation**
3. If the substructures do not overlap, run **recursion**; otherwise, run **dynamic programming**.
4. Try to find some greedy choice (or local optimal choice).
   By exchange argument, check if it leads to global optimal. If yes, give a **greedy algorithm**.

END!