Design and Analysis of Algorithms

T02a:Tutorial
Algorithm : Array Rotation

Dr. Ram P Rustagi
Sem IV (2020-Even)
Dept of CSE, KSIT
rprustagi@ksit.edu.in

Resources

- Programming Pearl
  - Author: Jon Bentley
  - Publisher: Pearson

Problem: Array Rotation

- Given a one dimensional array of `n` elements
  – Rotate left by `i` positions.
- Example:
  – Given array = `abcedfgh`,
  – Rotate left by 3 positions
    - Answer: `defghabc`
  – Rotate left by 5 positions
    - Answer: `fghabcde`
- Practical applications
  – Swapping adjacent blocks of memory of unequal size
  – Drag and drop a block of text elsewhere in a file
- Both time and space constraints are important

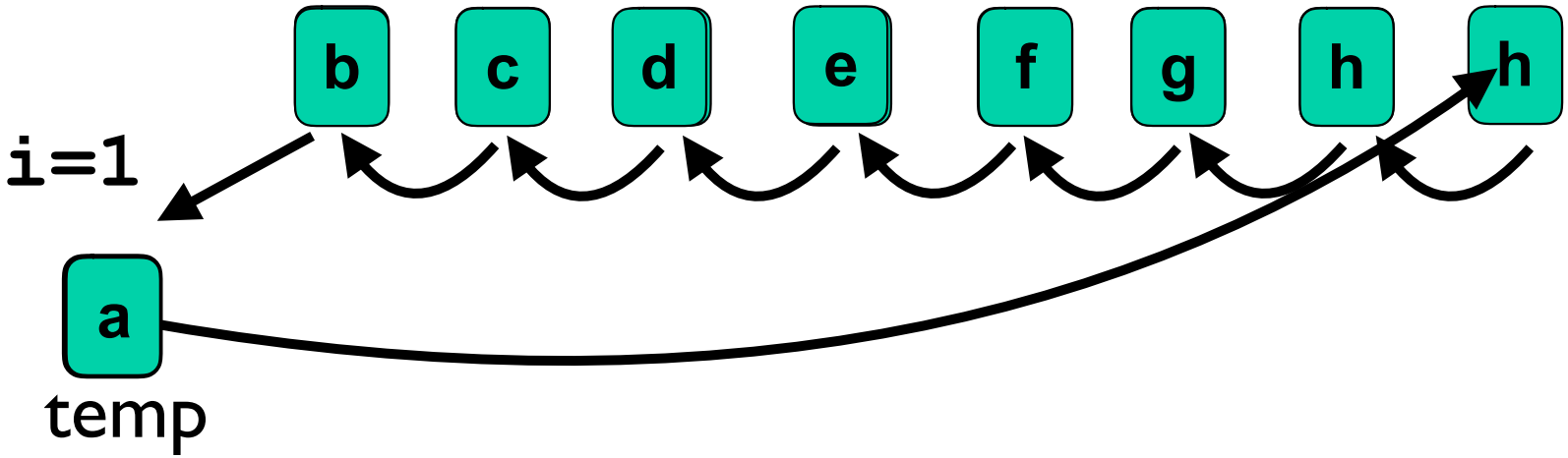
Solution 01

- Array rotation: solution 01
- Copy first `i` elements of an array in a temp array
- Move remaining `n-i` elements left `i` places
- Copy the first `i` elements from temp array
  – Back to the last positions in the original array
- Time and Space analysis:
  – Space: used `i` blocks of memory
    - Varies with input `i`.
  – Time: `n` number of copy operations.
    - Require minimum of `n` copies.
- Q: Can we solve using less space e.g.
  – Using fixed block of memory irrespective of `i`

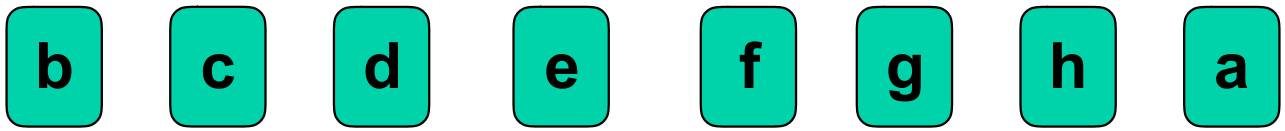
Solution 02

- Approach to solution 02:
- Consider `i=1`, then
  - Copy `1st` element in a temp variable (not array)
  - Move remaining `n-1` elements left `1` place
  - Copy temp variable to the last positions in the original array
- Consider `i=2`, then
  - Can we copy `2nd` element in a temp variable?
    - Then `4th`→`2nd`, `6th`→`4th`, `8th`→`6th` and `2nd`→`8th`.
    - Only 4 elements are shifted, Repeat with `1st`
  - Can we copy `1st` element in a temp variable
    - Then `3rd`→`2nd`, `5th`→`3rd`, `7th`→`5th` and `1st`→`7th`.
- If `i=3`, can we repeat the same process

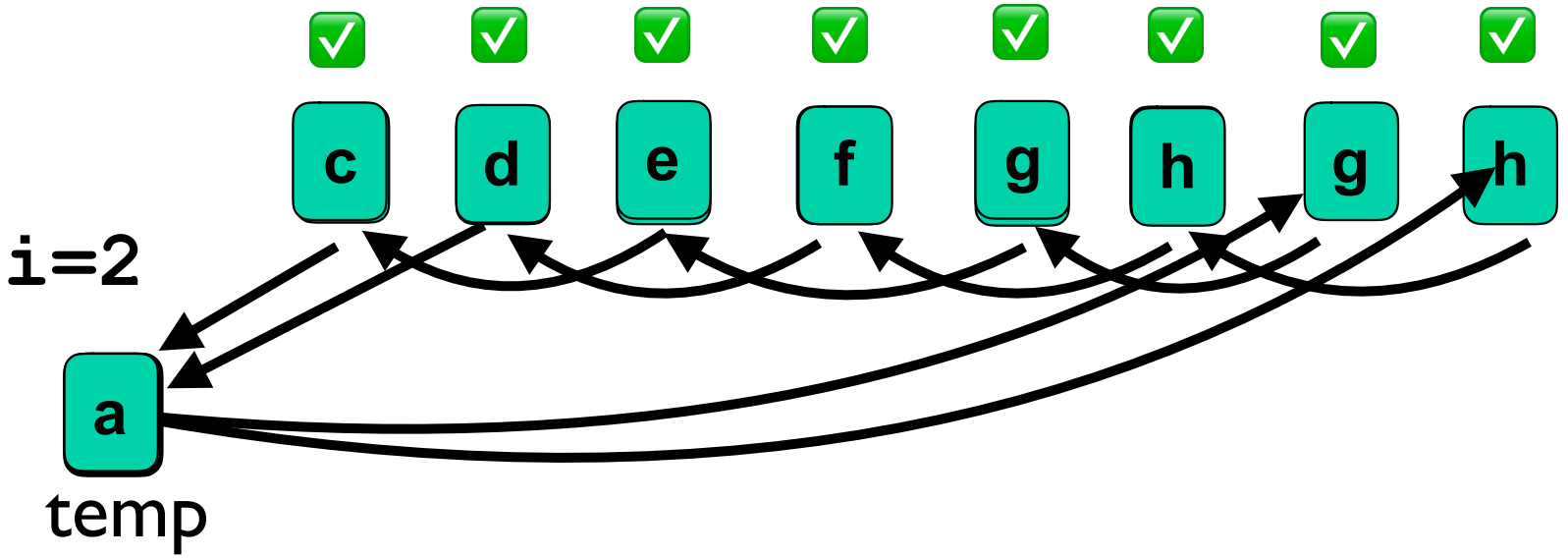
Example : Rotate Left by 1



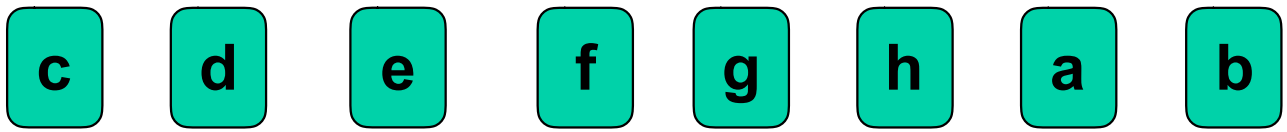
Expected result



Example : Rotate Left by 2



Expected result



Solution 03

- Consider the given array $A=a_1a_2...a_n$
  - To be rotated left k positions, and thus expected
    $A_{r,k}=a_{k+1}a_{k+2}...a_na_1a_2...a_k$
- Consider $B=a_{k+1}a_{k+2}...a_n$, and $C=a_1a_2...a_k$
  - Then $A=BC$, and
  - $A_{r,k} = CB = (B'C')'=(C')'(B')'$
    - where X' = reverse of X
- Algorithm rotate(0,n-1,k)
  reverse(0,k-1)
  reverse(k,n-1)
  reverse(0,n-1)
- This approach is Using Divide & Conquer
  - Most efficient in terms of time and space

Solution 03 (python code)

```
def reverse(arr, i, j):
  # reverse list from index i to j(inclusive)
  mid = (j - i + 1) // 2
  for k in range(mid):
    # swap the corresponding elements
    arr[i+k],arr[j-k] = arr[j-k],arr[i+k]

# main program
print("Input array is", arr)
reverse(arr, 0, rotateleft-1)
reverse(arr, rotateleft, len(arr)-1)
reverse(arr, 0, len(arr)-1)
print("Rotated array is", arr)
```

Summary

- Basic understanding how algorithmic efficiency plays a role in writing an efficient program.