CSE525 Monsoon 2020
Homework 2 Problem a

Ashima Garg (PhD19003)
Pramil Panjawani (PhD19008)

You CANNOT consult any other person or online resource for solving the homework problems. You can definitely ask the instructor or TAs for hints and you are encourage to do so (in fact, you will get useful hints if you ask for help at least 1-2 days before the due date). If we find you guilty of academic dishonesty, penalty will be imposed as per institute guidelines.

You are given an array $H[1 \ldots n]$ of distinct integers. Refer to Problem-1 of JE-CS374-SP2018A-HW4 [1].

(a) Write a recursive $O(n \log n)$ algorithm to compute the arrays $L[1 \ldots n]$ and $R[1 \ldots n]$.

Solution: The algorithm first divides the array H into two halves of almost equal sizes by calculating the middle index. It then recursively calls for left and right half of the arrays. When the number of elements remain one in the array. the *merge* function starts to execute.
In merge function, it firsts calculates the size of the left (H_l) and right arrays (H_r) divided in the previous operation. Loop - start from rightmost element down to 1 for left part of H (H_l) and from 1 to the rightmost element for the right part of H (H_r). If the smaller number is found on the left side of the array, then its right array (R) is updated with current index of the rightmost element. If the number found is smaller on the right side of the array (H_r), then the left array of (H_r) is updated with the current largest element found on the left side. The index is updated corresponding to the side (left H_l or right H_r) which is changed.
Each element in H gets traversed at maximum once in merge function. Therefore, time complexity of merge function is $O(n)$. The array is divided into two equal halves and then solved recursively. Therefore, the time complexity is:

$$\begin{aligned} T(n) &= T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n) \\ &= 2T(n/2) + O(n) \\ &= O(nlogn) \end{aligned}$$

```
⟨⟨Main function that recursively calls itself using Divide and Conquer Approach.⟩⟩
def whoTargetsWhom(H[1...n], left_index, right_index, L[1...n], R[1...n]):
    if left_index < right_index:
        middle_index = left_index + (right_index − left_index)/2 ⟨⟨Calculate the middle index⟩⟩
        whoTargetsWhom(H, left_index, middle_index, L, R)
        whoTargetsWhom(H, middle_index + 1, L, R)
        merge(H, left_index, middle_index, right_index, L, R)

def merge(H, left_index, middle_index, right_index, L, R):
    n1 ← middle_index − left_index + 1
    n2 ← right_index − middle_index
    i ← n1, j ← 1
    while(i > 0 and j <= n2):
        if H[i] < H[j]: ⟨⟨Number is greater in right part of array, therefore R of smaller element is updated.⟩⟩
            if R[i] ≠ None:
                R[i] ← j
                i ← i − 1
        else if H[i] > H[j]: ⟨⟨Number is greater in left part of array, therefore L of smaller element is updated.⟩⟩
            if L[j] ≠ None:
                L[j] ← i
                j ← j + 1
```

■

[1] https://courses.engr.illinois.edu/cs374/sp2018/A/homework/hw4.pdf

CSE525 Monsoon 2020 Ashima Garg (PhD19003)
Homework 2 Problem b Pramil Panjawani (PhD19008)

(b) Prove that at least $\lfloor n/2 \rfloor$ positions will be chosen as a "target".

Solution: The survivors are the elements whose both the adjacent elements are greater than the element itself as these elements cannot be target of any other element. The elements on both the left and right side of the array of such element will never find this element as its closest taller element because this element is always surrounded by larger elements of both the sides.

So, in an array of distinct elements such elements cannot be larger than n/2. So, survivors cannot be larger than n/2 therefore, the number of targets should be atleast n/2.

.

Another solution using proof by induction (an attempt using proof by induction):

Proof by Induction: Let Targets(n) represent the number of distinct elements chosen as targets in an array of size n. Targets(n) $\geq \lfloor n/2 \rfloor$

Base Case: Targets(1) $= \lfloor 1/2 \rfloor = 0$
The base case holds true as for an array of size 1, there are no elements on the left and right side, thus, $L[1] = None$ and $R[1] = None$.
Induction Hypothesis: Assume that Targets(k) has atleast $\lfloor k/2 \rfloor$ distinct elements.
Induction statement: To prove that Targets(k+1) has at least $\lfloor (k+1)/2 \rfloor$ distinct elements.

Case 1: k is even , new element is the taller than Ht(k) hero.
$\lfloor k/2+1 \rfloor = \lfloor k/2 \rfloor$
Ht(k) > Ht(k+1) ; R[j] $\leftarrow k+1$
$target(k+1) = target(k)+1 >= \lfloor k/2 \rfloor + 1$
i.e. target(k+1) has atleast $\lfloor k/2 \rfloor$ distinct value.

Case 2: k is even , new element is the shorter than Ht(k) hero.
$\lfloor (k+1)/2 \rfloor = \lfloor k/2 \rfloor$
Ht(k) > Ht(k+1) ; R[j] $\leftarrow k+1$
$target(k+1) = target(k)+1 >= \lfloor k/2 \rfloor + 1$
i.e. target(k+1) has atleast $\lfloor k/2 \rfloor$ distinct value.

Case 3: k is odd , new element is taller than Ht(k) hero.
$\lfloor (k+1)/2 \rfloor = \lfloor k/2 \rfloor + 1$
Ht(k+1) > Ht(k) ; L[j] $\leftarrow k+1$
$target(k+1) = target(k)+1 >= \lfloor k/2 \rfloor + 1 = \lfloor (k+1)/2 \rfloor$
i.e. target(k+1) has atleast $\lfloor (k+1)/2 \rfloor$ distinct value.

Case 4: k is odd , new element is shorter than Ht(k) hero.
$\lfloor (k+1)/2 \rfloor = \lfloor k/2 \rfloor + 1$
Ht(k+1) > Ht(k) ; R[j] $\leftarrow k+1$
$target(k+1) = target(k)+1 >= \lfloor k/2 \rfloor + 1 = \lfloor k/2+1 \rfloor$
i.e. target(k+1) has atleast $\lfloor k/2+1 \rfloor$ distinct value.

■

CSE525 Monsoon 2020 — Ashima Garg (PhD19003)
Homework 2 Problem c — Pramil Panjawani (PhD19008)

(c) Design and analyse a recursive algorithm to output the number of rounds to identify the smallest number in the linked list by repeatedly removing the targets in the manner specified in the above link.

Solution: Assume a doubly linked list(dll) with head A. *count_rounds(A)* accepts head of this dll after converting an array to linked list which takes $O(n)$. First, calculate length of the list using *calculate_length(A)* and initialise the total *rounds* to 0. Now, while loop checks until the list has the smallest element.*survivors(A, len)* accepts the head and length of dll and returns the updated head and length of dll. *suvivors(A, len)* deletes all the targets in each round. **The survivors are the elements whose both the adjacent elements are greater than the element itself** as these elements cannot be target of any other element. Thus, time taken to run a loop of *survivors* is $O(n)$. And from Problem2c, it has been proved that atleast $\lfloor n/2 \rfloor$ elements will be targets so, in each iteration at max $\lfloor n/2 \rfloor$ elements will survive (rest will be targets). Thus, time complexity is:
T(n) $= O(n)$ (to convert array to linked list $+O(n)$ (to calculate length of linked list) $+logn$ (number of times the while loop executes) $*O(n)$ (time taken to execute survivors). $= O(nlogn)$

```
⟨⟨Recursively calculates the length of linked list for the first time⟩⟩
def calculate_length(A): ⟨⟨A is the pointer to the first node of linked list.⟩⟩
  if !A:    return 0
  return 1 + calculate_length(A → next)

⟨⟨Returns the survivors after each round i.e. updated length of linked list and the pointer to the head of linked list⟩⟩.
def survivors(A, len):
  if !A: ⟨⟨if the head of linked list is NULL.⟩⟩
      len = 0, return
  if !A → prev: ⟨⟨handles case for the first node of the linked list.⟩⟩
      if A → data > A → next → data:
          temp ← A,   A ← A → next,   A → prev = NULL,
          delete temp,      len ← len − 1
          survivors(A, len)
      if A → data < A → next → data:
          survivors(A → next, len)
  if !A → next: ⟨⟨Handles test case for the last element in the linked list⟩⟩
      if A → data > A → prev → data :
          temp ← A,   A → next ← NULL,   delete temp,
          len ← len − 1,    survivors(A, len)
      else:          survivors(A, len)
  elif (A → next → data < A → data or
      A → prev → data < A → data): ⟨⟨For nodes other than the first and the last node, check for this condition.⟩⟩
      A → prev → next ← A → next,
      len ← len − 1
      survivors(A → next, len)
  else:    survivors(A → next, len)

⟨⟨Main function that returns the total number of rounds. It assumes the head of the doubly linked list⟩⟩
def count_rounds(A):
  if !A:    return
  total_len ← calculate_length(A)
  rounds ← 0
  while total_len > 1: ⟨⟨Run this loop until only one smallest element is left in the linked list.⟩⟩
      survivors(A, total_len) ⟨⟨Update the current survivors after each round.⟩⟩
      rounds ←rounds + 1
  return rounds + 1
```

■