ST340CW1Mihir

u1729346

22/10/2019

Question 1a: Merge Implementation

```
mergeLR <- function(left, right){
  ret <- numeric(length(left) + length(right))
  i <- 1
  j <- 1
  k <- 1
  while(i <= length(left) || j <= length((right))) {
    if(i <= length(left) && j <= length((right))){
      if(left[i] < right[j]){
        ret[k] <- left[i]
        i <- i + 1
        k <- k + 1
      }
      else{
        ret[k] <- right[j]
        j <- j + 1
        k <- k + 1
      }
    }
    else if(i <= length(left)){
      ret[k] <- left[i]
      k <- k + 1
      i <- i + 1
    }
    else if(j <= length(right)){
      ret[k] <- right[j]
      k <- k + 1
      j <- j + 1
    }
  }
  return(ret)
}
```

Merge Test:

```
left <- c(1,2,2.34)
right <- c(-12,-5,9,2000)
print(mergeLR(left,right))
```

```
## [1]   -12.00    -5.00     1.00     2.00     2.34     9.00 2000.00
```

Question 1b: Mergesort Implementation

```
mergesort <- function(a){
  if(length(a) == 1){
```

```r
    return(a)
  }
  else{
    mid <- floor(length(a)/2)
    left <- numeric(mid)
    right <- numeric(length(a) - mid)
    for(i in 1:length(left)){
      left[i] <- a[i]
      }
    for(j in 1:length(right)){
      right[j] <- a[j + mid]
    }
    return(mergeLR(mergesort(left), mergesort(right)))
  }
}
```

MergeSort Test:

```r
input <- c(1,2,2.34,2000,-12,-5,9)
print(mergesort(input))
```

```
## [1]  -12.00   -5.00    1.00    2.00    2.34    9.00 2000.00
```

Question 1d: Mergesort Correctness

Proposition: Mergesort correctly sorts an array $(a[1], ..., a[n])$ of size n.

Base Case: When $n = 1$, $a[1]$ is sorted by default since it contains only one element.

Inductive Hypothesis: When $n = k$, the array is correctly sorted by mergesort. i.e. $a[1] \leq ... \leq a[k]$.

Inductive Step: When $n = k + 1$,

$$mid = \lfloor \frac{k+1}{2} \rfloor$$

$$left = a[1], ..., a[mid]$$

$$right = a[mid + 1], ..., a[k + 1]$$

Since $\lfloor \frac{k+1}{2} \rfloor \leq k$ and $k + 1 - (\lfloor \frac{k+1}{2} \rfloor + 1) = k - \lfloor \frac{k+1}{2} \rfloor < k$, left and right will be correctly sorted by mergesort as assumed in our inductive hypothesis. We must show that mergeLR correctly outputs a sorted array when called upon left and right.

Proposition: mergeLR correctly returns a single array with the elements of the two input arrays in sorted order. Suppose left and right are two arrays with lengths p and q respectively and the algorithm is on its k^{th} iteration.

Base Case: When $k = 1$,

$$ret = concat(min\{left[1], right[1]\}, max\{left[1], right[1]\})$$

Hence, the returned array is in correct sorted order.

Inductive Hypothesis: For arbitrary k, the returned merged array is correctly sorted. i.e. $ret[1] \leq ... \leq ret[k]$.

Inductive Step: For the $(k + 1)^{th}$ iteration, suppose i is an index running through left, s.t. $1 \leq i \leq p$ and j is an index running through right, s.t. $1 \leq j \leq q$. Assume $left[i] \leq right[j]$ such that $ret[k] = left[i]$ and using the inductive hypothesis, we have $ret[1] \leq ... \leq ret[k]$. Considering the assumed case,

$$ret[k + 1] = min\{left[i + 1], right[j]\}$$

$$\implies ret[1] \leq ... \leq ret[k] \leq ret[k+1]$$

since $left[i] \leq left[i+1] \leq right[j]$ and in the case where $i+1 > p$, the if condition fails and the else if condition $j \leq length(right)$ is activated, where

$$ret[k+1] = right[j], ..., ret[p+q] = right[q]$$

and since right was already sorted, we have

$$ret[k+1] \leq ... \leq ret[p+q]$$

and hence the merged array is correctly sorted.

For the case when $right[j] < left[i]$, the proof is identical due to the symmetry of the problem.

Question 1d: Mergesort Runtime

Given an array of size n, mergesort is recursively called upon two equal halves (since array is assumed even) and once each half is of size 1, merge is executed $O(n)$ times, until the original array size is attained. Hence, we arrive at the recurrence equation given by:

$$T(1) = 1,$$

$$T(n) = 2T(\frac{n}{2}) + O(n), \quad n > 1$$

Proposition: $T(n) \leq n\log_2(n), \ \ \forall n \in \{2^k : k \in \mathbb{N}\}$

Base Case: $n = 2^1$

$$T(2) = 2T(1) + O(2) = 2 \leq 2\log_2(2) = 2$$

Hence, the base case is true.

Inductive Hypothesis: Assume true for $n = 2^k$:

$$T(2^k) = 2T(\frac{2^k}{2}) + O(2^k)$$

$$= 2T(2^{k-1}) + O(2^k) \leq 2^k log_2(2^k)$$

Inductive Step: For $n = 2^{k+1}$:

$$T(2^{k+1}) = 2T(\frac{2^{k+1}}{2}) + O(2^{k+1})$$

$$= 2T(2^k) + O(2^{k+1})$$

using our inductive hypothesis, we have:

$$\leq 2^{k+1}\log_2(2^{k+1}))$$

and hence the proposition stands $\forall n \in \{2^k : k \in \mathbb{N}\}$.

Question 1e: Quicksort vs Mergesort

Quicksort and Mergesort are both divide and conquer based sorting strategies. Quicksort uses a pivot (either chosen naively or by other methods such as median) to partition the array whereas Mergesort always partitions the array into halves. This results in Mergesort having a worst-case time complexity of $O(n\log(n))$. A scenario where the array is sorted in descending order results in Quicksort's worst-case time complexity of $O(n^2)$. Hence, Mergesort is faster and more efficient than Quicksort.