ST340CW1Mihir

u1729346

22/10/2019

Question 1a: Merge Implementation

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

Question 1b: Mergesort Implementation

```
mergesort <- function(a){
  if(length(a) == 1){
    return(a)
  }
  else{
    mid <- floor(length(a)/2)
    left <- numeric(mid)
    right <- numeric(length(a) - mid)
    for(i in 1:length(left)){
```

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

```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 $T(n)$ is the total number of comparisons made by mergesort on an input of size n 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 $\Theta(n)$ times, until the original array size is attained. Hence, we arrive at the recurrence equation given by:

$$T(1) = 1,$$

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

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.