EECS 340: Assignment 4

Shaochen (Henry) ZHONG, `sxz517`

Due and submitted on 03/19/2020
EECS 340, Dr. Koyutürk

Problem 1

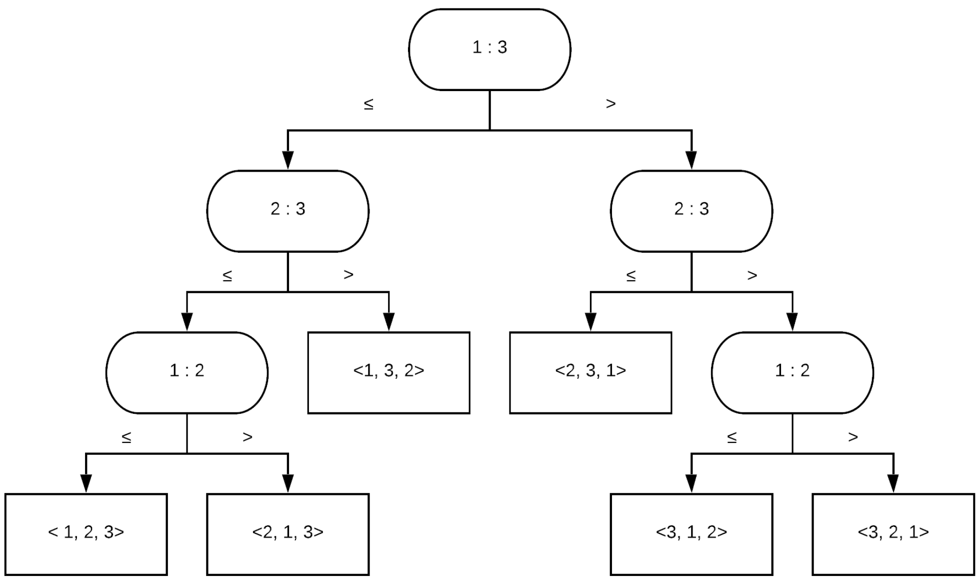

Figure 1: Decision Tree of QUICKSORT with 3 elements

Problem 2

Procedure PREPROCESS(A)

Algorithm 1 PreProcess(A)

procedure PREPROCESS(A, p, r)
    Let $C[0, 1, ..., k]$ be an arry of 0.
    for $i \leftarrow 0$ **to** n **do**
        $C[A[i]] = C[A[i]] + 1$
    for $j \leftarrow 1$ **to** k **do**
        $C[j] = C[j] + C[j - 1]$
    return C

Procedure **Query(A, a, b)**

Algorithm 2 Query(A, a, b)

procedure QUERY(A, a, b)
  if $a == 0$ **then**
    return $A[b]$
  else
    return $A[b] - A[a-1]$

Problem 3

This probably not the most reader-friendly pseudocode you'd see, but the idea behind it is very intuitive. Thus, I'd like to keep this design, and I'll give many comments to navigate you through my train-of-logic.

Algorithm 3 Sparse-Transpose(R, C, V, m, n, k)

procedure SPARSE-TRANSPOSE(A, a, b)
  Let $CH[0, 1, ..., n]$ be an arry of 0. ▷ Value holder array.
  Let $VH[0, 1, ..., n]$ be an arry of 0. ▷ Column index holder array.
  for $i \leftarrow 0$ **to** n **do**
    $start \leftarrow R[i]$ ▷ Calculate the row index of element in C
    $end \leftarrow R[i+1] - 1$
    Let $CPR[\ ]$ be an empty array.
    Let $VPR[\ ]$ be an empty array.
    for $j \leftarrow start$ **to** end **do**
      $CPR[i].append(C[j])$ ▷ Holds column indexs according to their row index.
      $VPR[i].append(V[j])$ ▷ Holds values according to their row index.
    $CH[i].append(CPR)$ ▷ $CH[i]$ represents the column index of elements on i-th row.
    $VH[i].append(VPR)$ ▷ $VH[i]$ represents the value of elements on i-th row.
  Let $R'[\ ]$ be an empty array.
  Let $C'[\ ]$ be an empty array.
  Let $V'[\ ]$ be an empty array.
  $R'.append(0)$ ▷ To fill-in the extra element as $len(R') = m + 1$.
  Let len_max to be the max len of all elements in CH. ▷ Row with most elements.
  for $i \leftarrow 0$ **to** len_max **do**
    POP-SMALLEST$(R', C', V', CH, VH, len_max)$ ▷ Pop the original row index/value to C'/V' of an element with smallest column index.
  for $i \leftarrow 1$ **to** n **do**
    $R'[i] = R'[i] + R'[i-1]$
  return R', C', V'

Algorithm 4 Pop-Smallest(R, C, V, CH, VH, len_max)

```
procedure Pop-Smallest(R, C, V, CH, VH, len_max)
    min_value ← CH[0][0]    ▷ Assume 1st element on row 1 is the one with smallest column index.
    min_i ← 0
    min_j ← 0
    for i ← 0 to len_max do    ▷ Iterate all elements' column index on a particular row.
        R_counter ← 0    ▷ Check if elements with same column index on multiple rows.
        for j ← 0 to n do    ▷ Interate all rows registered in CH
            if CH[j][i] == min_value then    ▷ If current element has the same column index
                R_counter ← R_counter + 1    ▷ Update R'.
                if j < min_j then    ▷ If current element also has smaller row index than the holder.
                    min_value ← CH[j][i]    ▷ Update holders.
                    min_i ← i
                    min_j ← j
            if CH[j][i] < min_value then    ▷ If current elements column index is smaller than the holder.
                R_counter ← R_counter + 1
                min_value ← CH[j][i]    ▷ Update holders.
                min_i ← i
                min_j ← j
                continue
        CH[min_j].remove(min_i)    ▷ Pop the element with smallest column index.
        C.append(min_j)    ▷ Append element with smallest column index's row index.
        V.append(VH[min_j][min_i])    ▷ Append element's corresponding value.
        VH[min_j].remove(min_i)    ▷ Pop such value.
        R.append(R_counter)    ▷ Register how many elements on each column.
```

The algorithm is considered $O(m+n+k)$ as the line 4-6 in Sparse-Transpose needs $O(m)$, loop through of C, V needs $O(k)$, and loop through of R' needs $O(n)$.