CSORW4231 HOMEWORK 3

Due Mon, Mar 06

Jun Hu

(jh3846)

Problem 1. Exercise 8.1-1 on page 193 and Exercise 8.2-4 on page 197.

Solution.

8.1-1 If the array is already sorted, we only need to compare $n-1$ times between all n elements. So the smallest possible depth for the decision tree is $n-1$. Draw decision tree:

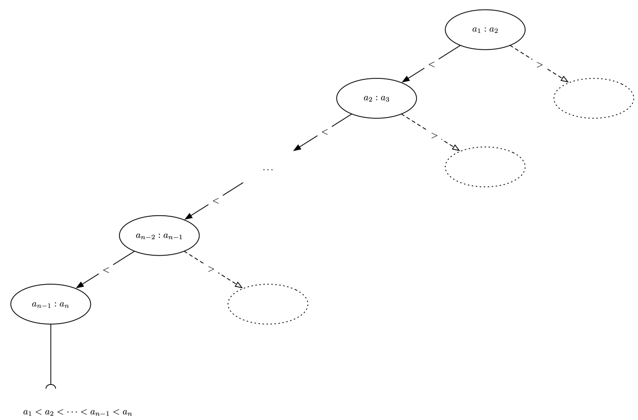

Figure 1: Decision Tree of 8.1-1

8.2-4 Borrow the method from COUNTING-SORT by creating a count set C for query. Assuming given n integer set is A.

```
COUNTING(A)
    Let C[0..k] be a new array
    for i == 0 to k
      C[i] = 0
    for j == 1 to n
      C[A[j]] = C[A[j]] + 1
    for i = i to k
      C[i] = C[i] + C[i + 1]
```

The processing for COUNTING is:

$$T_1 = c_1 k + c_2 n + c_3 k + c_4 = \Theta(n+k)$$

When given query input [a, b]:

```
QUERY(a, b)
    return C[min(b, k)] - C[max(a - 1, 0)]
```

The query returning time is:

$$T_2 = c_5 = \Theta(1)$$

□

Problem 2. Problem 8-4(b) on page 207 and 8-6(a and b) on page 208: Comparison lower bound. (If you would like to challenge yourself, check Exercise 9.1-2 on page 215.)

Solution.

8-4 b. Taking one red jar to compare with blue jars to find a matched pair can be looked as a decision tree with 3 children at each node.
The lower bound of Lower bound or the number of comparisons is the lower bound on the height of the decision tree.
Observation:

$$\text{Number}_{leaves} \geq n!$$

Let h be the height of the decision tree:

$$3^h \geq \text{Number}_{leaves}$$

Which is:

$$3^h \geq n!$$

Stirling Approximation:

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1+\Theta\left(\frac{1}{n}\right)\right)$$

Thus:

$$\begin{aligned}
h &\geq \log_3(n!) \\
&\geq \log_3\left(\sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1+\Theta\left(\frac{1}{n}\right)\right)\right) \\
&\geq \log_3\left(\frac{n}{e}\right)^n \\
&= n\log_3 n - n\log_3 e \\
&= \Omega(n\lg n)
\end{aligned}$$

8-6 a. The possible ways are picking n from $2n$ for combination:

$$\binom{n}{k}$$

8-6 b.

$$2^h \geq \text{Number}_{leaves} \geq \binom{n}{k}$$

$$\begin{aligned}
h &\geq \lg\binom{n}{k} \\
&\geq \lg\left(\frac{2^{2n}}{\sqrt{\pi n}}\left(1+O\left(\frac{1}{n}\right)\right)\right) \\
&= \lg(2^{2n}) - \lg\sqrt{\pi n} + \lg\left(1+O\left(\frac{1}{n}\right)\right) \\
&= 2n - o(n)
\end{aligned}$$

□

Problem 3. Problem 8-2 on page 206: Sorting in place in linear time. 4. Exercise 9.3-6 and Exercise 9.3-8 on page 223.

Solution.

8-2 a.

```
SORT(A)
    Let C[1..n] be a new array
    j == 1
    for i = 1 to n
      if A[i] == 0
        C[j] == A[i]
      j = j + 1
    for j = 1 to n
      if A[i] == 1
        C[j] == A[i]
      j = j + 1
    return C
```

The elements in A are collected into C keeping the same odder in A with the same key value 0 or 1, which mean the sort is stable. The array A has been visited twice in the algorithm, so $T = O(n)$

8-2 b.

```
SORT(A)
    j == 1
    for i = 1 to n
      if A[i] == 0
        exchange A[i] with A[j]
        j = j + 1
    return A
```

Only maintaining A and taking $O(n)$.

8-2 c.

```
SORT(A)
  for i = 1 to n - 1
    for j = n down to i + 1
      if A[j] < A[j - 1]
        exchange A[j] with A[j - 1]
  return A
```

8-2 d. Use algorithm in (a), which takes O(n) time. A b-bit key takes bit value as 0 or 1, we can sort in the running time of

$$O(b(n+2)) = O(bn)$$

8-2 e.

```
SORT(A)
  Let C[1..k] be a new array
  for i = 1 to k
    C[i] == 0
  for i = 1 to n
    C[A[i]] = C[A[i]] + 1
  for i = 2 to k
    C[i] = C[i] + C[i - 1
  j == 1
  while j < n
    A.copy = A[j]
    if j != C[A.copy]
      exchange A[C[A.copy]] with A[j]
      C[A.copy] = C[A.copy] -1
    else j = j + 1
  return A
```

The modified COUNTING-SORT is not stable.

9.3-6 QT(A, k)

Let C be a new array with length k - 1
if k == 1
    return
else
    i = $\lfloor \frac{k}{2} \rfloor$
    q = SELECT(A,$\lfloor i\frac{n}{k} \rfloor$)
    PARTITION(A, q)
    C.append(QT(A[1] to A[$\lfloor i\frac{n}{k} \rfloor$], $\lfloor \frac{k}{2} \rfloor$, C))
    C.append(QT(A[$\lfloor i\frac{n}{k} \rfloor$ + 1] to A[n], $\lceil \frac{k}{2} \rceil$, C))
    return q

Draw a recursion tree. At the top level we need to find k 1 order statistics, and it costs O(n) to find one. For each of the two roof contains at the most $\lfloor \frac{k-1}{2} \rfloor$ and $\lceil \frac{k-1}{2} \rceil$order statistics The sum of the costs for these two nodes is O(n). The level of the tree is $\lg(k-1)$, n numbers for each level, so the running time is $\Theta(n \lg k)$.

9.3-8

```
MEDIAN(X, Y, n)
 if n == 1
   return min(X[1], Y[1])
 if X[n/2] < Y[n/2]
   return MEDIAN(X[n/2 + 1 ... n], Y[1 ... n/2], n/2)
 else
   return MEDIAN(X[1 ... n/2], Y[n/2 + 1 .. n], n/2)
```

□

Problem 4. Problem 9.2(skip a and b) on page 225: Weighted median.

Solution.

9.2 c. Modify SELECT as: Let m be the median of median, Compute $\sum_{m_i<m} w_i$ and $\sum_{m_i>m} w_i$, Check either exceeds 1/2, If so, recursive call for the one contain weighted median If not, return. Sp the running time keeps $\Theta(n)$

9.2 d.

9.2 e

$\square$

Problem 5. Problem 11-4(a and b only) on page 284.

Solution.

11-4 a.

11-4 b.

□