

CLRS Exercise

Tongda Xu

November 8, 2018

1 7

1.1 7.3

1.1.1 a

This is certain concerning the *Randomized* procedure, the probability of any index i is chosen from $[0, n - 1]$ is:

$$\begin{aligned} \Pr(\text{pivot} = i) &= \frac{1}{n} \\ E(X_i) &= 1 * \Pr(\text{pivot} = i) + 0 * \Pr(\text{pivot} \neq i) = \frac{1}{n} \end{aligned}$$

1.1.2 b

It is certain that if i th element is chosen as pivot, *Random-Partition* cost $\Theta(n)$ time, and it will call *QuickSort* $[1, q - 1]$, *QuickSort* $[q + 1, n]$ recursively.

Concerning only the first *Partition*, this would be the result:

$$\begin{aligned} E(T(n)) &= \sum_{i=1}^n \Pr(\text{pivot} = i)(T(i - 1) + T(n - i) + \Theta(n)) \\ &= \sum_{i=1}^n X_i(T(i - 1) + T(n - i) + \Theta(n)) \end{aligned}$$

1.1.3 c

$$\begin{aligned} \text{Concerning } X_i &= \frac{1}{n} \\ E(T(n)) &= \sum_{i=1}^n \frac{1}{n}(T(i - 1) + T(n - i) + \Theta(n)) \\ &= \sum_{i=1}^n \frac{1}{n}T(i - 1) + \sum_{i=1}^n \frac{1}{n}T(n - i) + \sum_{i=1}^n \frac{1}{n}\Theta(n) \\ &= \frac{2}{n}\sum_{i=1}^{n-1}T(i) + \Theta(n) \end{aligned}$$

1.1.4 d

$$\begin{aligned} &\sum_{k=2}^{n-1} k \lg k \\ &\leq \lg \frac{n}{2} \sum_{k=2}^{\frac{n}{2}} k + \lg n \sum_{k=\frac{n}{2}}^{n-1} k \\ &= \lg n \sum_{k=2}^{n-1} k - \lg 2 \sum_{k=2}^{\frac{n}{2}} k \\ &= \lg n \frac{(n+1)(n-2)}{2} - \frac{(\frac{n}{2}+2)(\frac{n}{2}-1)}{2} \\ &\leq \lg n \frac{n^2}{2} - \frac{n^2}{8} \end{aligned}$$

by Calculus, we have:

$$(\frac{1}{2}x^2 \lg x - \frac{1}{4}x^2)|_1^{n-1} \leq E(T(n)) \leq (\frac{1}{2}x^2 \lg x - \frac{1}{4}x^2)|_2^n$$

1.1.5 e

Proof of $E(T(n)) = O(nlgn)$:

Assume that $\forall k \in [1, n-1], \exists c, E(T(k)) \leq cklgk - \Theta(k)$

For $k = n, E(T(n)) \leq \frac{n}{2}c(lgn\frac{n^2}{2} - \frac{n^2}{4} - \Theta(n^2)) + \Theta(n) \leq cnlgn - \Theta(n)$

Proof of $E(T(n)) = \Omega(nlgn)$:

Assume that $\forall k \in [1, n-1], \exists c, E(T(k)) \geq cklgk + \Theta(k)$

For $k = n, E(T(n)) \geq \frac{n}{2}c(lgn\frac{(n-1)^2}{2} - \frac{(n-1)^2}{4} + \Theta(n^2)) + \Theta(n) \geq cnlgn + \Theta(n)$
 $\rightarrow E(T(n)) = \Theta(nlgn)$

1.2 7.5

1.2.1 a

From counting Theorem, it could be noticed that:

$$p_i = \frac{(i-1)(n-i)}{C_n^3} = \frac{6(i-1)(n-i)}{n(n-1)(n-2)}$$

1.2.2 b

$$\begin{aligned} Pr(i = \text{medium})(\text{normal}) &= \frac{1}{n} \\ Pr(i = \text{medium})(\text{3part}) &= \frac{6(\frac{1}{2}n-1)(n-\frac{1}{2}n)}{n(n-1)(n-2)} = \frac{3}{2} \frac{1}{n} \\ Pr(\text{3part}) - Pr(\text{normal}) &= \frac{1}{2} \frac{1}{n} \end{aligned}$$

1.2.3 c

$$\begin{aligned} \text{Consider } f_{diff} &= \int_{\frac{2}{3}n}^{\frac{2}{3}n} \left(\frac{6(i-1)(n-i)}{n(n-1)(n-2)} - \frac{1}{n} \right) di \\ &= \frac{(-2i^3 + 3(n+1)i^2 - 6ni - (n-1)(n-2)i) \Big|_{i=\frac{1}{3}n}^{i=\frac{2}{3}n}}{n(n-1)(n-2)} \\ \lim_{n \rightarrow \infty} f_{diff} &= \frac{4}{27} \end{aligned}$$

1.2.4 d

Consider we are so lucky that each partition we choose the median:

In the Iteration tree, we have:

$$T(n) = \begin{cases} c & n = 1 \\ 2T(\frac{1}{2}n) + n & n > 1 \end{cases}$$

The $\Omega(nlgn)$ is kept even in best case.

2 8

2.1 8.1-1

n-1 times, since we need n elements to formulate

2.2 8.1-2

$$\Sigma_1^n l g k < \int_1^{n+1} l g k d k = (k l g k - k)_1^n = (n l g n - n) - (0 - 1) = n l g n - n + 1$$

2.3 8.1-3

\leftrightarrow proof at least half of branch is longer than h

Consider a decision tree with $n!/2$ elements

\leftrightarrow proof at least half of branch is longer than h

Consider a decision tree with $n!/n$ elements

\leftrightarrow proof at least half of branch is longer than h

Consider a decision tree with $n!/2^n$ elements, this is not significant enough and could leave only $\Omega(lg \frac{n!}{2^n}) = \Omega(n l g n - n) = \Omega(n l g n)$ elements

2.4 8.2-4

Consider a trim version of counting sort, build the C map up and query directly:

COUNTING-SORT-TRIM(A, k)

```

1  C[]
2  for i = 0 to k
3      C[i] = 0
4  for j = 1 to A.length
5      C[A[j]] ++
6  for m = 1 to k
7      C[m] += C[m - 1]
8  return C[m]
```

DIRECT-QUERT(A, k, a, b)

```

1  C = COUNTING-SORT-TRIM(A, k)
2  if a < 1
3      return C[b]
4  else return C[b] - C[a - 1]
```

2.5 8.3-2

Heapsort is not stable

The scheme would be very similar to counting sort and takes $\Theta(n)$ time

2.6 8.3-4

First, with $O(n)$ time: convert n numbers k_{10} into k_n which has 3 digits.

Second, with $O(d(n+n))$ time (*Lemma 8.3*): Radix sort n 3-digit numbers with each digits take up to n possible values.

DIGITS_CONVERT(X)

```
1  result[]
2  for  $i = 2$  downto 0
3       $result[i] = X/n^i$ 
4       $X = X \bmod n^i$ 
5  return result
```

SORT(A, x)

```
1  result[]
2  for each  $S$  in  $A$ 
3       $S = \text{DIGITS\_CONVERT}(S)$ 
4  RADIX-SORT( $A, x$ )
```

3 9

3.1 9.2-1

once $p == r$, the function return and recursion end.

3.2 9.2-2

It is because $\forall k, X_k = \frac{1}{n}$, giving information on which k would not effect observation

3.3 9.2-3

RANDOMIZED-SELECT-ITER(A, p, r, i)

```
1  while 1
2      if  $i == k$ 
3          return  $A[i]$ 
4      else
5           $q = \text{RANDOM-PARTITION}(A, p, r)$ 
6          if  $i < k$ 
7               $r = q - 1$ 
8          else  $p = q + 1, i = i - k$ 
```

3.4 9.2-4

The worst case is reverse side:

$pivot = 9, 8, 7, 6, 5, 4, 3, 2, 1, 0$

3.5 9.1

3.5.1 a

Sorting: MERGE-SORT(A) in worst case $O(n \lg n)$

Query: CALL-BY-RANK(A, k) i times in worst case $O(i)$, here we assume manip-

ulating $O(n)$ space cost $O(n)$ time.

3.5.2 b

Building: BUILD-MAP-HEAP(A) in worst case $O(n)$

Query: calling EXTRA-MAX(A, k) i times in worst case $O(ilgn)$

3.5.3 c

Selecting: SELECT(A, i) in worst case $O(n)$

Sorting: MERGE-SORT(A') in worst case $O(ilgi)$

3.6 9.2

3.6.1 a

$$\Sigma_1^{k-1} w_i = \Sigma_1^{k-1} \frac{1}{n} = \frac{k-1}{n} < \frac{1}{2}$$

$$\Sigma_{k+1}^n = \frac{n-k}{n} \leq \frac{1}{2}$$

3.6.2 b

WEIGHT-MEDIAN(A)

```

1  w[] = SORT(A).weight
2  n = w.length
3  for i = 1 to n
4      w[i] = w[i] + w[i - 1]
5  return FIND(w[], 1/2)
```

3.6.3 c

SUM($w_1, w_i, lasti, lastsum$)

```

1  if i > lasti
2      return lastsum + NORMAL-SUM( w_{lasti,i} )
3  else return lastsum - NORMAL-SUM( w_{i,lasti} )
```

WEIGHT-MEDIAN-LINEAR(A)

```

1  while 1
2      if sum[w_1, w_i, lasti, lastsum] < 1/2, sum[w_1, w_{i+1}, lasti, lastsum] > 1/2
3          return i
4      else
5          lastsum = sum[w_1, w_i, lasti, lastsum], lasti = i
6          if sum[w_1, w_i] < 1/2
7              i = MEDIAN(A, i, r)
8          else i = MEDIAN(A, p, i)
```

We will experience $\log n$ iteration, but the load is decreasing logarithmically, so the result is linear. Notice the sum is special here, calculating the difference only.

3.7 9.4

3.7.1 a

$k \leq i$ or $k \geq j : 0$
 $i < k < j : \frac{2}{j-i+i}$

3.7.2 b

3.7.3 c

3.7.4 d

4 11

4.1 11.1-2

Consider $vector < bool > A, a.size() = m$, just store the bool value of $key = m$ exist or not.

SEARCH(A, key)

```
1  if A(key)
2      return key
3  else return NIL
```

INSERT(A, key)

```
1  A(key) = 1
```

DELETE(A, key)

```
1  A(key) = 0
```

4.2 11.2

4.2.1 a

Consider for a ball i fall into a specific bucket $Pr(i) = \frac{1}{n}$
 Then consider Binomial Distribution, $Pr(k) = C_n^k Pr(i)^k (1 - Pr(i))^{n-k}$

4.2.2 b

Consider random picking a slot, the probability of that slot is maximum is $Pr_{max} = \frac{1}{n}$, and it contains k elements Q_k . for conditional probability, we have:

$$Pr_k = Pr_{i=k|max} = \frac{Pr(i=k \cap max)}{Pr_{max}} \leq \frac{Pr(i=k)}{Pr_{max}} = nQ_k$$

4.2.3 c

Proof:

$$\begin{aligned}
Q_k &= \left(\frac{1}{n}\right)^k \left(\frac{n-1}{n}\right)^{n-k} C_n^k \\
&= \frac{(n-1)^{n-k}}{n^n} \frac{\Pi_0^{k-1} n-k}{k!} \\
&\leq \frac{n^n}{n^n} \frac{1}{k!} \\
&= \frac{e^k}{k^k} \frac{1}{k^{\frac{1}{2}(1+\Theta(\frac{1}{n}))}} \\
&\leq \frac{e^k}{k^k}
\end{aligned}$$

4.2.4 d

Proof for Q_{k_0} :

$$\begin{aligned}
Q_{k_0} &= \frac{e^{\left(\frac{c l g n}{l g l g n}\right)}}{\left(\frac{c l g n}{l g l g n}\right)^{\frac{c l g n}{l g l g n}}} \\
&= \frac{n^{\frac{c l g \frac{c}{c}}{l g l g n}}}{\frac{c l g l g l g n}{n}} = n^{\frac{c l g \frac{c}{c} + c l g l g l g n}{l g l g n} - c}
\end{aligned}$$

It would not take effort to notice that since $\lim_{n \rightarrow \infty} \frac{c l g \frac{c}{c} + c l g l g l g n}{l g l g n} = 0$

$\forall c > 3 + \epsilon, Q_{k_0} = O(\frac{1}{n^3})$

And $P_k \leq n Q_k \rightarrow P_k = O(\frac{1}{n^2})$

4.2.5 e

$$E(M) = \sum_{M=1}^n M Pr(M) < n Pr(M > \frac{c l g n}{l g l g n}) + \frac{c l g n}{l g l g n} Pr(M \leq \frac{c l g n}{l g l g n})$$

A stronger conclusion to note:

$$\begin{aligned}
E(M) &= \sum_{M=1}^n M Pr(M) < M Pr(M > \frac{c l g n}{l g l g n}) + \frac{c l g n}{l g l g n} Pr(M \leq \frac{c l g n}{l g l g n}) \\
&\leq \int_{\frac{c l g n}{l g l g n}}^{\infty} \frac{1}{n} dn + 1 * \frac{c l g n}{l g l g n} \\
&= l g \left(\frac{c l g n}{l g l g n} \right) + \frac{c l g n}{l g l g n} \\
&= O\left(\frac{c l g n}{l g l g n}\right)
\end{aligned}$$

5 15

5.1 15.1-1

$$2^n - 1 = \sum_{j=0}^{n-1} 2^j$$

5.2 15.1-2

Do not know how!

5.3 15.1-3

BOTTOM-UP-CUT-ROD(p, n, c)

```
1   $r[] = c$ 
2  for  $j = 1$  to  $n$ 
3      for  $i = 1$  to  $j$ 
4           $r[i] = \max(p[i] + r[j - i] - c)$ 
5  return  $r[n]$ 
```

5.4 15.1-4

MEMOIZED-CUT-ROD(p, n, m, s)

```
1  if  $m[n] > -1$ 
2      return  $m[n]$ 
3  else
4      for  $i = 1$  to  $n$ 
5           $m[n] = \max(p[i] + r[n - i])$ 
6           $s[n] = i$ 
7      return  $m[n]$ 
```

5.5 15.1-5

See Code

5.6 15.2-1

See Code

5.7 15.2-2

See Code

5.8 15.2-3

Assume that $\forall k \leq n - 1, T(k) \geq c2^k$

Then $T(n) = \sum_{k=1}^{n-1} T(k)T(n-k) = (n-1)c^22^n > c2^n$

So $T(n) = \Omega(n), \omega(n)$

5.9 15.2-4

See Figure 1

5.10 15.2-5

For each level $h(i) = i(n-i)$

For tree $T(n) = 2\sum_{i=1}^{n-1} i(n-i)$

Ex 15.2.4

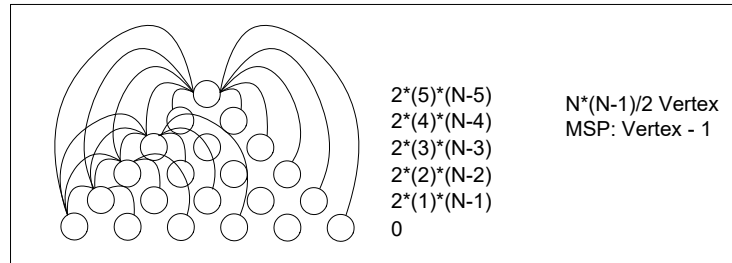


Figure 1: 15.2-4

$$\begin{aligned}
 &= \frac{3n^3 + 3n^2}{3} - \frac{2n^3 + 3n^2 + n}{3} \\
 &= \frac{n^3 - n}{3}
 \end{aligned}$$

5.11 15.2-6

Assume that $\forall k \leq n-1, N(k) = k-1$

Then $N(n) = N(n-1) + 1$

So $N(n) = n-1$

5.12 15.3-1

running through: $T(n) = n * P_n^n = n * n! > 4^n$

running recursion: $T(n) = 2 \sum_{i=1}^{n-1} 4^i + n = \frac{8}{3} 4^{n-1} + n \leq 4^n$

running through takes longer

5.13 15.3-2

no overlapping subproblem call

5.14 15.3-3

Yes

5.15 15.3-4

Do not know how!

5.16 15.4-1

See code

5.17 15.4-2

See code

5.18 15.4-3

See code

5.19 15.5-1

A Preorder Traverse of BST

PRE-ORDER-PRINT-AID($i, j, root$)

```
1  if  $root[i, j] - 1 - i \geq 0$ 
2      k  $root[i, root[i, j] - 1]$  is the left child of k  $root[i, j]$ 
3      PRE-ORDER-PRINT-AID( $i, root - 1, root[i, root[i, j] - 1]$ )
4  else d  $i - 1$  is the left child of k  $root[i, j]$ 
5  if  $j - root[i, j] - 1 \geq 0$ 
6      k  $root[root[i, j] + 1, j]$  is the right child of k  $root$ 
7      PRE-ORDER-PRINT-AID( $root + 1, j, root[root[i, j] + 1, j]$ )
8  else d  $i - 1$  is the right child of  $root$ 
```

PRE-ORDER-PRINT($root$)

```
1  k  $root[1, n]$  is the root
2  PRE-ORDER-PRINT-AID( $1, n, root$ )
```

5.20 15.5-3

Asymptotically there would be no change to the running time, just the constant cn^3 increase

Time spent on w would increase from $\Theta(n^2)$ to $\Theta(n^3)$

5.21 15.1

LSP(s, t, G)

```
1   $r = G.size()$ 
2   $DPS[r] = 0$ 
3   $DPr[r] = path(s, t)$ 
4   $max = -\infty$ 
5  for  $i = 1$  to  $r$ 
6       $max(DPS[j] + DPr[r - j] + what)$ 
7  return  $max$ 
```

5.22 15.1

5.23 22.1-1

for both out-degree and in-degree $\Theta(V + E)$ time
both take $\Theta(V)$ memory

5.24 22.1-2

```
1 → 2 → 3 → NIL
2 → 1 → 4 → 5 → NIL
3 → 1 → 6 → 7 → NIL
4 → 2 → NIL
5 → 2 → NIL
6 → 3 → NIL
7 → 3 → NIL
0 - 1 - 1 - 0 - 0 - 0 - 0
1 - 0 - 0 - 1 - 1 - 0 - 0
1 - 0 - 0 - 0 - 0 - 1 - 1
0 - 1 - 0 - 0 - 0 - 0 - 0
0 - 1 - 0 - 0 - 0 - 0 - 0
0 - 0 - 1 - 0 - 0 - 0 - 0
0 - 0 - 1 - 0 - 0 - 0 - 0
```

5.25 22.1-3

```
TRANSPOSE(Adjlist)
1  new AdjlistPrime
2  for each node in Adjlist
3      for each subnode in Adjlist(node)
4          AdjlistPrime(subnode).insert(node)
5      Adjlist = AdjlistPrime
```

For adjacent list: just traverse every node and rebuild one
 $\Theta(E + V)$ for time and space complexity, hard to do it inplace

```
TRANSPOSE(Adjmatrix)
1  for each pair(i, j) in upper left Adjmatrix
2      SWAP(Adjmatrix[i, j], Adjmatrix[j, i])
```

For adjacent matrix: just transpose the matrix
 $\Theta(V^2)$ for time and $\Theta(1)$ for space

5.26 22.1-4

use an adjacent matrix as aid.

5.27 22.2-3

line 2 $\rightarrow u.ifgrey = 0$
line 5 $\rightarrow s.ifgrey = 1$
line 14 $\rightarrow v.ifgrey = 1$

5.28 22.2-4

take $\Theta(V^2)$ time and $\Theta(V^2)$ space, since we need to search every column to find adjacent list.

line 12 \rightarrow **for** : *each* $v \in M[u]$
line 13 \rightarrow **if** : $v == \mathbf{true}$ **and** $v.color == white$