National Taiwan University
Department of Electrical Engineering
Algorithms, Fall 2019

Handout #16
November 5, 2019
Yao-Wen Chang, Hui-Ru Jiang, and Chien-Mo Li

Sample Solutions to Midterm #1

Problem 1. (10 pts total)

(a) (5 pts) False. Counterexample:

$$f(n) = \begin{cases} 1, & \text{if } n \text{ is odd} \\ n^4 + 1, & \text{if } n \text{ is even} \end{cases}$$

$$g(n) = n^2 + 1$$

Both $f(n)$ and $g(n)$ are positive functions, but $f(n) \neq O(g(n))$ and $g(n) \neq O(f(n))$.

(b) (5 pts) False. Counterexample: Let $f_i(n) = i \cdot n$. Then, $g(n) = f_n(n) = n^2 \neq O(n)$.

Problem 2. (10 pts total)

(a) (5 pts) Use Master theorem (case 2). $f(n) = \Theta(n^{\log_2 4}) = \Theta(n^2) \Rightarrow T(n) = \Theta(n^2 \lg n)$.

(b) (5 pts)

(1) $T(n) = T(\alpha n) + T(\beta n) + n \geq cn$ for all $n \geq n_0$. Pick $c = 1$ and $n_0 = 2$. Note that $T(\cdot)$ is positive. Thus, $T(n) = \Omega(n)$.

(2) Figure 1 shows the recursion tree. Let $h = \max\{\log_{1/\alpha} n, \log_{1/\beta} n\}$.

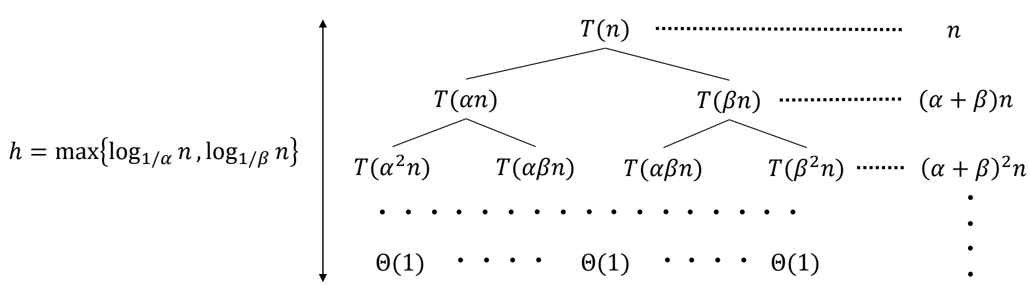

Figure 1: Recursion tree for Problem 2(b).

$$\begin{aligned} T(n) &\leq \sum_{i=0}^{h} (\alpha+\beta)^i n \\ &\leq \sum_{i=0}^{\infty} (\alpha+\beta)^i n \\ &= (1 + (\alpha+\beta) + (\alpha+\beta)^2 + \cdots)n \\ &= \frac{n}{1-(\alpha+\beta)} \end{aligned}$$

Thus, $T(n) = O(n)$.

(3) From (1) and (2), we have $T(n) = \Theta(n)$.

Problem 3. (15 pts total)

(a) Since the skylines are sorted by the x-coordinates of buildings in order from left to right, we can modify the merging procedure of merge sort to combine two skylines.

MergeSkyline(A, B)
Input: A skyline A of size n_1 and a skyline B of size n_2
Output: A combined skyline S
Let $S[1..n_1 + n_2]$ be a new array.
Let $A[i].x$ and $A[i].h$ represent the x-coordinate and the height of the ith building in the skyline A, respectively.
$A[n_1 + 1].x = B[n_2 + 1].x = \infty$
$i = j = 1$
$h_a = h_b = 0$
for $k \leftarrow 1$ **to** $n_1 + n_2$ **do**
  if $A[i].x \leq B[j].x$ **then**
    $h_a = A[i].h$
    $S[k].x = A[i].x$
    $S[k].h = \max(h_a, h_b)$
    $i = i + 1$
  end
  else
    $h_b = B[j].h$
    $S[k].x = (B[j].x$
    $S[k].h = \max(h_a, h_b)$
    $j = j + 1$
  end
end
Then we need to remove consecutive horizontal lines of equal height in the output skyline.
for $k \leftarrow 2$ **to** $S.length$ **do**
  if $S[k].h == S[k-1].h$ **then**
    delete $S[k]$
  end
end
return S

The running time for each iteration of both for loops is $\mathcal{O}(1)$, so the overall running time for the **MergeSkyline** procedure is $\mathcal{O}(n_1 + n_2)$.

(b) We can use Divide-and-Conquer to find the skyline of n buildings.

FindSkyline(B, p, r)
if $p == r$ **then**
  Let $S[2]$ be a new array.
  $S[1].x = B[p].L$
  $S[1].h = B[p].H$
  $S[2].x = B[p].R$
  $S[2].h = 0$
  return S
end
else if $p < r$ **then**
  $q = \lfloor \frac{p+r}{2} \rfloor$
  S_1 = **FindSkyline(**B, p, q)
  S_2 = **FindSkyline(**$B, q + 1, r$)
  S = **MergeSkyline(**S_1, S_2)
  return S
end

We can give the skyline in $\mathcal{O}(1)$ time for the base case that there is only one building. The running time for **MergeSkyline** at each level requires $\mathcal{O}(n)$, so the recurrence for **FindSkyline** is $T(n) = 2T(n/2) + cn$. Therefore, the overall running time to find the skyline of n buildings is $\mathcal{O}(n \log n)$, which is the same as merge sort.

(c) We first show the loop invariant holds for the **MergeSkyline** procedure:
The loop invariant property is:
At the start of each iteration of the first for loop, the subarray $S[1..k-1]$ contains the $k-1$ smallest coordinates of $A[1..n_1+1]$ and $B[1..n_2+1]$ in sorted order. Moreover, $A[i]$ and $B[j]$ are the smallest coordinates of their arrays that have not been copied back into S.
Initialization: Prior to the first iteration of the loop, we have $k = 1$ and the new skyline array $S[k-1]$ is empty. $A[i].x$ and $B[j].x$ correspond to the smallest x-coordinates of these two skylines. **Maintenance**: At each iteration of the loop, $S[k-1].h$ represents the height which starts at $S[k-1].x$ and whose endpoint is not yet determined. $A[i].x$ and $B[j].x$ correspond to the next smallest x-coordinates greater than $S[k-1].x$ at which the height of their original skylines change. So the height of the new skyline we are producing will be $S[k-1].h$ until $x \geq \min(A[i].x, B[j].x)$. Suppose $A[i].x \leq B[j].x$, which means the height of skyline A at $x = A[i].x$ changes to $A[i].h$, and the height of skyline B at $x = A[i].x$ is still $B[j-1].h$. Thus the height of the new skyline $S[k].h$ at $S[k].x = A[i].x$ should be $\max(A[i].h, B[j-1].h)$. Since we can construct correct x-coordinate and height for the new skyline at each iteration, and the loop invariant will hold as k and i, j increase.
Termination: At termination, $k = n_1 + n_2 + 1$. By the loop invariant, $S[1..n_1 + n_2]$ is the skyline constructed from $A[1..n_1]$ and $B[1..n_2]$ and sorted by their x-coordinates. Yet there may be some redundant pairs whose height are the same as their previous ones. We need to scan the new skyline again to remove these redundant pairs.

For **FindSkyline**, the correctness of the base case is trivial since the skyline of a single building ($k = 1$) can be constructed from its triplet (L, H, R). Assume **FindSkyline** can produce the skyline for $k < n$ buildings. Besides, the **MergeSkyline** procedure gives us the correct combination of two skylines. So we can divide the original problem into subproblems and solve them recursively, and we will get the correct skyline of n buildings in the end.

Problem 4. (10 pts total)

(a) (2 pts)

Iteration	A	i	j
1	3 2 15 6 2 17 11 22	1	7
2	3 2 2 6 15 17 11 22	3	5
3	3 2 2 6 15 17 11 22	5	4
	return $j = 4$		

(b) (3 pts) Traversing n elements will have $\Theta(n)$ time complexity.

(c) (2 pts)

```
Quicksort_H (A, p, r)
if p < r
    q = H-Partition(A, p, r)
    Quicksort_H(A, p, q)
    Quicksort_H(A, q + 1, r)
```

(d) (3 pts) No. Here is a counterexample: $A =< 1_1, 1_2 > \Rightarrow A =< 1_2, 1_1 >$. (We mark the two 1's as 1_1 and 1_2)

Problem 5. (10 pts total)

(a) (5 pts)

index	1	2	3	4	5	6	7	8	9	10	11
step1	7	1	5	11	10	14	3	9	8	2	13
step2	7	1	5	11	13	14	3	9	8	2	10
step3	7	1	5	11	13	14	3	9	8	2	10
step4	7	1	14	11	13	5	3	9	8	2	10
step5	7	13	14	11	10	5	3	9	8	2	1
step6	14	13	7	11	10	5	3	9	8	2	1

(b) (3 pts)

```
Heap-Decrease-Key(A, i, key)
    if key > A[i]
        error "new key is larger than current key"
    A[i] = key
    Max-Heapify (A, i)
```

(c) (2 pts)

index	1	2	3	4	5	6	7	8	9	10	11
step1	100	90	80	70	65	60	55	1	2	3	5
step2	100	6	80	70	65	60	55	1	2	3	5
step3	100	70	80	6	65	60	55	1	2	3	5

Problem 6. (10 pts total)

(a) (5 pts)

```
Order-Statistics (A, i)
    n = A.length
    Let B[0..n − 1] be a new array
    for j = 0 → n − 1
        make B[j] an empty list
    for j = 0 → n
        insert A[j] to list B[⌊nA[j]⌋]
    cnt = 0
    for j = 0 → n − 1
        if cnt + B[j].length ≥ i
            sort list B[j] with insertion sort
            return B[i − cnt]
        cnt = cnt + B[j].length
```

(b) (3 pts) When all numbers are in the same bucket, the insertion sort takes $\Theta(n^2)$-time. As a result, the worst-case running time of the order statistics is $\Theta(n^2)$.

(c) (2 pts) If the numbers are uniformly distributed, there are only a few numbers in one bucket, so the running time of insertion sort can be ignored. Since the for loop takes $\Theta(n)$-time, the running time of the order statistics is $\Theta(n)$.

Problem 7. (14 pts total)

(a) (3 pts) See Figure 2(a).

(b) (3 pts) See Figure 2(b).

(c) (4 pts) See Figure 2(c).

(d) (4 pts) See Figure 2(d).

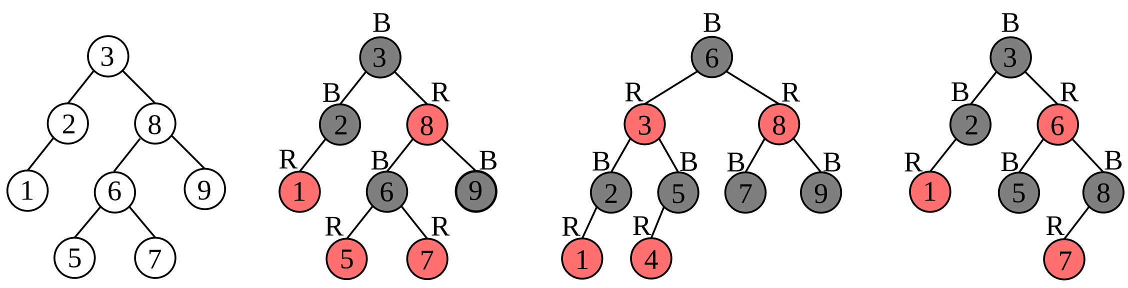

Figure 2: Sample trees for Problem 7.

Problem 8. (8 pts total)

(a) $a = 1.50$, $b = 1.30$, $c = 2.05$, $d = 2.00$, $f = 2.60$
$p = 3$, $q = 3$, $r = 2$, $s = 3$, $t = 3$

(b) The expected search cost of the tree is
$3 \cdot 0.10 + 2 \cdot 0.10 + 1 \cdot 0.20 + 2 \cdot 0.05 + 4 \cdot 0.05 + 4 \cdot 0.10 + 3 \cdot 0.20 + 3 \cdot 0.10 + 3 \cdot 0.10 = 2.6$.

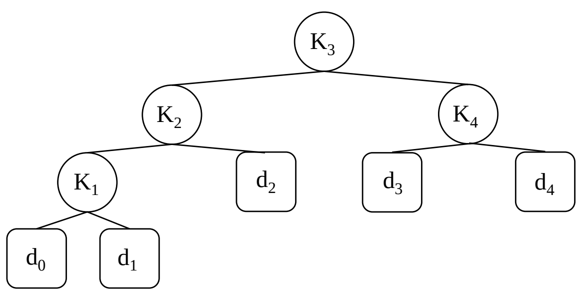

Figure 3: The optimal binary search tree for Problem 8(b).

Problem 9. (5 pts total)
$N(1, n) = N(2, k-1) + 1 + N(k+1, n)$

Problem 10. (13 pts total)

(a) (7 pts)

$$\begin{aligned} Z_{i+j} &== shuffle(X_i, Y_j) \text{ iff} \\ Z_{i+j-1} &== shuffle(X_{i-1}, Y_j) \text{ and } Z_{i+j} == X_i, \text{ or} \\ Z_{i+j-1} &== shuffle(X_i, Y_{j-1}) \text{ and } Z_{i+j} == Y_j. \end{aligned}$$

(b) (4 pts)

```
Is_Shuffle(X, Y, Z, m, n)
Let s[1..m, 1..n] be a new table
for i = 1 to n
    s[i, 0] = TRUE
for j = 1 to n
    s[0, j] = TRUE
for i = 1 to m
    for j = 1 to n
        if (s[i − 1, j] == TRUE and Z_{i+j} == x_i) or (s[i, j − 1] == TRUE and Z_{i+j} == y_j)
            s[i, j] = TRUE
        else
            s[i, j] = FALSE
return s[m, n]
```

(c) (2 pts) The running time of Is_Shuffle is $O(mn)$.