National Taiwan University
Department of Electrical Engineering
Algorithms, Fall 2019

Handout #6
October 6, 2019
TAs: Shang-Chien Lin and Yi-Ting Lin

Sample Solutions to Homework #1

1. (10) Exercise 2.1-1 and Exercise 2.3-1

   (a) See Figure 1.

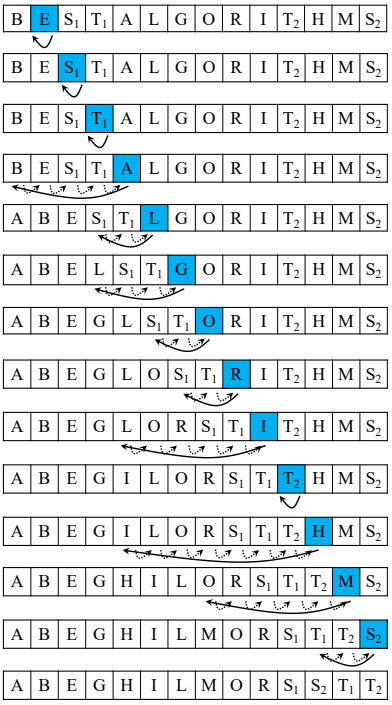

Figure 1: The process steps for Problem 1(a).

   (b) See Figure 2.

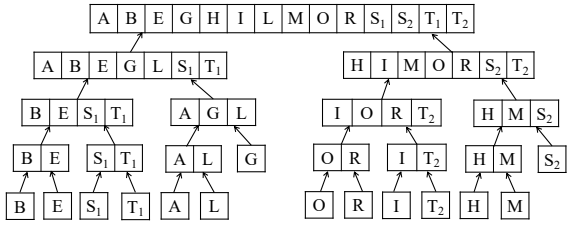

Figure 2: The process steps for Problem 1(b).

2. (20) Exercise 2.2-2 (page 29)

   (a)
   - n: $length[A]$.
   - t_j: # of times that the inequality in line 5 holds at iteration j.
   - **Pseudocode:**

SelectionSort(A)	cost	times
1. **for** $j \leftarrow 1$ **to** $length[A] - 1$ **do**	c_1	n
2. $key \leftarrow A[j]$;	c_2	$n-1$
3. $key\_pos \leftarrow j$;	c_3	$n-1$
4. **for** $i \leftarrow j+1$ **to** $length[A]$ **do**	c_4	$\sum_{j=1}^{n-1}(n-j+1)$
5. **if** $A[i] < key$	c_5	$\sum_{j=1}^{n-1}(n-j)$
6. **then** $key \leftarrow A[i]$;	c_6	$\sum_{j=1}^{n-1} t_j$
7. $key\_pos \leftarrow i$;	c_7	$\sum_{j=1}^{n-1} t_j$
8. $A[key\_pos] \leftarrow A[j]$;	c_8	$n-1$
9. $A[j] \leftarrow key$;	c_9	$n-1$

(b) **Loop invariant:** After performing the jth iteration of the external **for** loop, the subarray $A[1..j]$ contains the j smallest numbers in A, sorted in non-decreasing order.

- **Initialization:** The first iteration finds the smallest number in A and puts it in $A[1]$. Obviously the subarray is sorted, since it contains only one number.
- **Maintenance:** We must show that if the loop invariant was true at the end of iteration j, it will still hold at the end of iteration $j+1$. Thus, we can assume that at the end of iteration j the subarray $A[1..j]$ contained the j smallest numbers in A sorted in non-decreasing order. The next iteration finds the smallest number in $A[j+1..n]$ and puts it in $A[j+1]$. By our assumption, this number is equal to or greater than any of the numbers in $A[1..j]$. Therefore, the new subarray $A[1..j+1]$ is also sorted in non-decreasing order.
- **Termination:** After the last iteration, j equals $n-1$, and the subarray $A[1..j]$ contains the $n-1$ smallest numbers, sorted in non-decreasing order. By the same argument as before, the entire array must be sorted in non-decreasing order.

(c) Because every time we select out the smallest number from the subarray $A[j..n]$, the nth smallest number is the last one in the array. Besides, the nth smallest number is also the largest one in the array. Therefore, we do not need to change its place at all, implying that we need to run for only the first $n-1$ elements.

(d)
- $T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4\sum_{j=1}^{n-1}(n-j+1) + c_5\sum_{j=1}^{n-1}(n-j) + c_6\sum_{j=1}^{n-1} t_j + c_7\sum_{j=1}^{n-1} t_j + c_8(n-1) + c_9(n-1)$
- **Best case:** If the array is already sorted, all t_j's are 0.
  Quadratic: $T(n) = (\frac{c_4+c_5}{2})n^2 + (c_1+c_2+c_3+c_8+c_9+\frac{c_4-c_5}{2})n - (c_2+c_3+c_4+c_8+c_9) = \Theta(n^2)$
- **Worst case:** If the array is in reverse sorted order, $t_j = j, \forall j$.
  Quadratic: $T(n) = (\frac{c_4+c_5+c_6+c_7}{2})n^2 + (c_1+c_2+c_3+c_8+c_9+\frac{c_4-c_5-c_6-c_7}{2})n - (c_2+c_3+c_4+c_8+c_9) = \Theta(n^2)$

3. (10) Exercise 2.3-7 (page 39)
Use merge sort to sort the n elements in $\Theta(n \lg n)$ time. Then, for each element x_1, use binary search to find if there exists x_2 such that $x_1 + x_2 = x$. The total run time= $\Theta(n \lg n) + n \cdot \Theta(\lg n) = \Theta(n \lg n)$.

4. (15) Problem 2-4 (a), (b), and (d) (page 41–42)

(a) Inversions: (2,1), (3,1), (8,6), (8,1), (6,1).

(b) Array with largest number of inversions: ⟨n,n-1, ...,2,1⟩.
Number of inversions: n(n-1)/2.

(d) We describe an algorithm $COUNT - INV$ that takes as input an array $A[p...r]$ and returns the array A in sorted order as well as the number of inversions in A (i.e., the number of pairs (i,j) with $p \le i \le j \le r$ and $A[i] > A[j]$). The algorithm works as follows.
The procedure $COUNT - MERGE(A,p,q,r)$ is similar to $MERGE(A,p,q,r)$: in addition to merging two sorted subarrays $A[p...q]$ and $A[q+1...r]$ into a single sorted array, it also computes the number of pairs (i,j) with $p \le i \le q$, $q \le j \le r$, and $A[i] > A[j]$. We can consider one such pair (i,j) with $L[i] > R[j]$ before merging. The element $R[j]$ will be put into the array A before element $L[i]$. At that time, all inversion pairs associated with $R[j]$ will be counted, which equals

to $(n_1 - i + 1)$, the number of remaining elements in the array L. When $COUNT - MERGE$ can correctly compute the number of inversion pairs while merging two subarrays, $COUNT - INV$ can return the correct number of inversions of the input array A because the correspondence between inversions and merge-inversions is one-to-one.

```
COUNT − INV(A, p, r)
if p < r
    then q = ⌊(p + r)/2⌋
         a = COUNT − INV(A, p, q)
         b = COUNT − INV(A, q + 1, r)
         c = COUNT − MERGE(A, p, q, r)
return (a+b+c)
```

```
COUNT − MERGE(A, p, q, r)
n1 = q − p + 1
n2 = r − q
let L[1..n1 + 1] and R[1..n2 + 1] be new arrays
for i = 1 to n1
    L[i] = A[p + i − 1]
for j = 1 to n2
    R[j] = A[q + j]
L[n1 + 1] = ∞
R[n2 + 1] = ∞
inversions = 0
i = j = 1
for k = p to r
    if L[i] ≤ R[j]
        A[k] = L[i]
        i = i + 1
    else
        inversions = inversions + (n1 − i + 1)
        A[k] = R[j]
        j = j + 1
return inversions
```

Since we only add an additional constant-time operation to each iteration of the for loop in the merging procedure, the worst-case runtime is the same as that of Merge-Sort, which is $\Theta(nlogn)$.

5. (10) Exercise 3.1-1 (page 52)
   Let $h(n)$ denote $max\{f(n), g(n)\}$. Since $f(n) \leq h(n)$ and $g(n) \leq h(n)$, we have $f(n) + g(n) \leq 2 \cdot h(n)$ or $h(n) \geq 1/2 \cdot (f(n) + g(n))$ for all $n \geq 1$. Hence, $h(n) = \Omega(f(n) + g(n))$.

   Since $f(n)$ and $g(n)$ are asymptotically nonnegative, there exists a constant n_0 such that $f(n) \geq 0$ and $g(n) \geq 0$ for $n \geq n_0$. For $n \geq n_0$, we therefore have $h(n) \leq f(n) + g(n) = 1 \cdot (f(n) + g(n))$. Hence, $h(n) = O(f(n) + g(n))$. We conclude that $h(n) = \Theta(f(n) + g(n))$.

6. (15) Problem 3-3 (pages 61–62) (partial)

   (a) $2^{2^{n+1}} > 2^n > n^{\lg \lg n} = (\lg n)^{\lg n} > (\lg n)! > n^3 > n^2 = 4^{\lg n} >$
   $\lg(n!) > 2^{\sqrt{2 \lg n}} > \sqrt{\lg n} > 2^{\lg^* n} > \lg^* n > n^{1/\lg n} = 1$

(b) The following $f(n)$ is nonnegative, and for all functions $g_i(n)$ in part (a), $f(n)$ is neither $O(g_i(n))$ nor $\Omega(g_i(n))$.

$$f(n) = \begin{cases} 2^{2^{n+2}} & \text{,if } n \text{ is even,} \\ 0 & \text{,if } n \text{ is odd.} \end{cases}$$

7. (5) Exercise 4.2-1 (pages 82)
The first matrices are

$$\begin{array}{ll} S_1 = 6 & S_6 = 8 \\ S_2 = 4 & S_7 = -2 \\ S_3 = 12 & S_8 = 6 \\ S_4 = -2 & S_9 = -6 \\ S_5 = 6 & S_{10} = 14 \end{array}$$

The products are

$$\begin{aligned} P_1 &= 1 \cdot 6 = 6 \\ P_2 &= 4 \cdot 2 = 8 \\ P_3 &= 6 \cdot 12 = 72 \\ P_4 &= -2 \cdot 5 = -10 \\ P_5 &= 6 \cdot 8 = 48 \\ P_6 &= -2 \cdot 6 = -12 \\ P_7 &= -6 \cdot 14 = -84 \end{aligned}$$

The four matrices are

$$\begin{aligned} C_{11} &= 48 + (-10) - 8 + (-12) = 18 \\ C_{12} &= 6 + 8 = 14 \\ C_{21} &= 72 + (-10) = 62 \\ C_{22} &= 48 + 6 - 72 - (-84) = 66 \end{aligned}$$

The result is

$$\begin{pmatrix} 18 & 14 \\ 62 & 66 \end{pmatrix}$$

8. (10) Exercise 4.3-7 (page 87)
Substitution proof with the assumption $T(n) \le cn^{\log_3 4}$:

$$\begin{aligned} T(n) &= 4T(n/3) + n \\ &\le 4c(n/3)^{\log_3 4} + n \\ &= 4c(n^{\log_3 4}/3^{\log_3 4}) + n \\ &= 4c(n^{\log_3 4}/4) + n \\ &= cn^{\log_3 4} + n \\ &\not\le cn^{\log_3 4} \end{aligned}$$

The substitution proof with the assumption $T(n) \le cn^{\log_3 4}$ fails.

Assuming the lower-order term to be subtracted is dn, then the substitution proof with assumption $T(n) \le cn^{\log_3 4} - dn$ will be

$$\begin{aligned} T(n) &= 4T(n/3) + n \\ &\le 4c(n/3)^{\log_3 4} - (4/3)dn + n \\ &= cn^{\log_3 4} + (1 - (4/3)d)n \\ &\le cn^{\log_3 4} - dn \end{aligned}$$

By choosing $d \ge 3$, the substitution proof with the assumption $T(n) \le cn^{\log_3 4} - dn$ works.

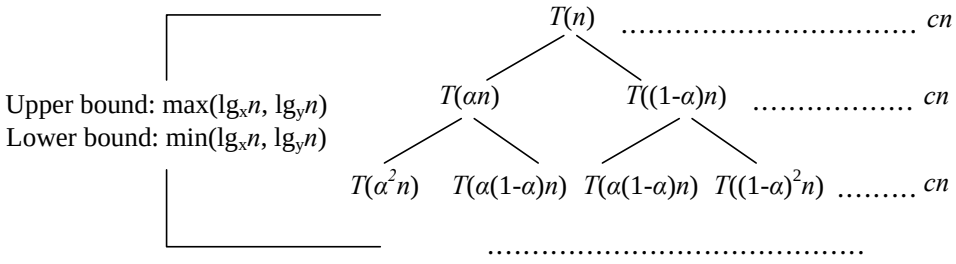

Figure 3: The recursion tree for Problem 9.

9. (10) Exercise 4.4-9 (page 93)
   Let $x = \frac{1}{\alpha}$ and $y = \frac{1}{1-\alpha}$. Construct the recurrence tree as Figure 3. We can obviously see that $T(n) = \Theta(n \lg n)$.

10. (10) Exercise 4.5-4 (page 97)
    Apply the master theorem:
    $n^{\log_b a} = n^2$, $f(n) = n^2 \lg n$, and we cannot find $\epsilon > 0$ such that $f(n) = \Omega(n^{\log_b a + \epsilon})$. That means we cannot apply the master method to solve this recurrence.

    Solve by iteration:

$$\begin{aligned}
T(n) &= 4T(\frac{n}{2}) + n^2 \lg n \\
&= 16T(\frac{n}{4}) + n^2 \lg n + n^2 \lg \frac{n}{2} \\
&= O(\sum_{k=0}^{\lg n} (n^2 lg \frac{n}{2^k})) \\
&= O(\sum_{k=0}^{\lg n} (n^2 \lg n - kn^2)) \\
&= O(n^2 \lg n(\lg n + 1) - \frac{n^2}{2} \lg n(\lg n + 1)) \\
&= O(\frac{n^2}{2} \lg n(\lg n + 1)) \\
&= O(n^2 \lg^2 n)
\end{aligned}$$

11. (15) Problem 4-1 (a), (c), and (e) (page 107)

    (a) $T(n) = 2T(n/2) + n^4 = \Theta(n^4)$. Using master theorem, $f(n) = n^4 = \Omega(n^{\log_b a + \varepsilon}) = \Omega(n^{1+\varepsilon})$ for some const $\varepsilon > 0$, and $af(n/b) = n^4/8 \le cf(n) = cn^4$ holds for any $1/8 \le c < 1$. Thus, case3 of master theorem applies, and $T(n) = \Theta(n^4)$.

    (c) $T(n) = 16T(n/4) + n^2 = \Theta(n^2 \lg n)$. Since $f(n) = n^2 = \Theta(n^{\log_b a}) = \Theta(n^2)$, case2 of master theorem applies.

    (e) $T(n) = 7T(n/2) + n^2 = \Theta(n^{\lg 7})$. Since $2 < \log_2 7 < 3$, we have that $n^2 = O(n^{\log_2 7 - \varepsilon})$ for some const $\varepsilon > 0$. Thus, case1 of the master theorem applies, and $T(n) = \Theta(n^{\lg 7})$.

12. (15) Problem 4-3 (a), (c), and (e) (page 108)

    (a) Apply the master theorem: $n^{\log_b a} = n^{\log_3 4} = n^{1.261\ldots}$, $f(n) = n \lg n$, and $f(n) = O(n^{\log_b a - \epsilon})$ for some $\epsilon > 0$. Thus, $T(n) = \Theta(n^{\log_3 4})$.

    (c) Using master theorem, $f(n) = n^2\sqrt{n} = \Omega(n^{\log_b a + \varepsilon}) = \Omega(n^{2+\varepsilon})$ for some const $\varepsilon > 0$, and $af(n/b) \le cf(n)$ holds for some constant $c < 1$. Thus, case3 of master theorem applies, and $T(n) = \Theta(n^2\sqrt{n})$.

(e) Using the iteration method, we get (if n is a power of 2)

$$\begin{aligned} T(n) &= \frac{n}{\lg n} + \frac{n}{\lg n - 1} + ... + \frac{n}{1} + 2 \cdot T(1) \\ &= n \cdot \sum_{i=1}^{\lg n} \frac{1}{i} + O(1) \\ &= \Theta(n \lg \lg n). \end{aligned}$$

Consider the integration of $\frac{1}{i}$, we can use $\int_1^{\lg n+1} \frac{1}{i}$ as the lower bound and $\int_1^{\lg n+1} \frac{2}{i}$ as the upper bound.

13. (15)
Suppose that the repeated parts of these n words are composed of L different lines of text. We can write the script as follows:

```
line 1 = <text of line 1 here>
line 2 = <text of line 2 here>
...
line L = <text of line L here>
For i = 1, 2, ..., L
   For j = 1, 2, ..., i
     Sing lines j through 1
   Endfor
Endfor
```

Now, the nested `For` loops have length bounded by a constant c_1, so the real space in the script is consumed by the text of the lines. Each of these lines in the script has length at most c_2 (where c_2 is the maximum line length c plus the space to write the variable assignment).

The number of different lines L is bounded by the condition that each line has only one word, while the total words are $1 + 2 + ... + L = \frac{1}{2}L(L+1) = n$. Thus we have $L = O(\sqrt{2n})$. Plugging this into our bound on the length of the script, we have $f(n) \leq c_1 + c_2 \cdot L = O(\sqrt{n})$.

14. (40) DIY.