National Taiwan University of Science and Technology
Department of Electrical Engineering

Algorithm Design and Application
Sample Solution to Homework #1

1. (a) See Figure 1.
   (b) See Figure 2.
   (c) See Figure 3 for the first iteration of Qricksort.
   (d) See Figure 4.

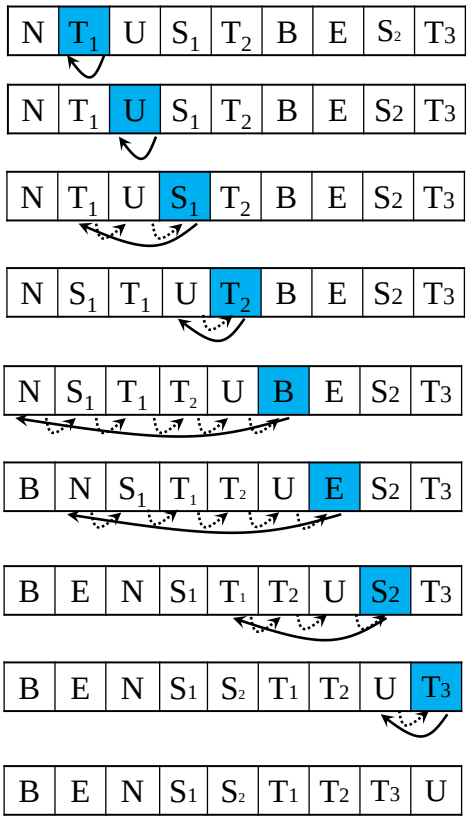


Figure 1: Problem 1(a).

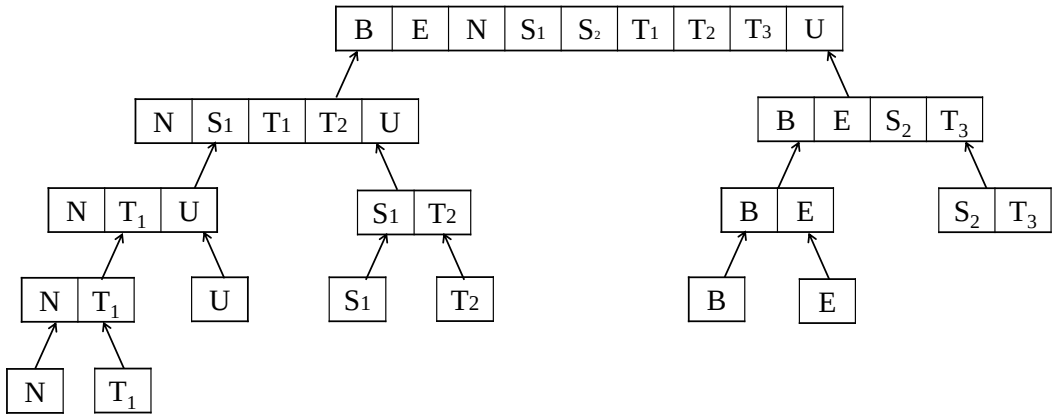
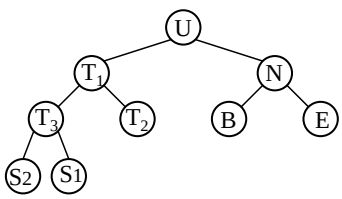
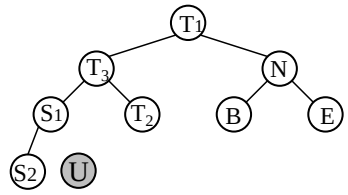


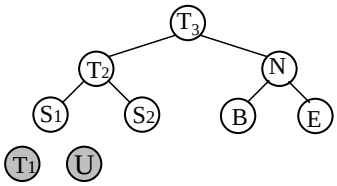
Figure 2: Problem 1(b).



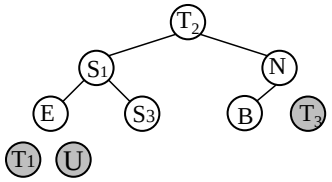
(a) Heap built by Build-Max-Heap



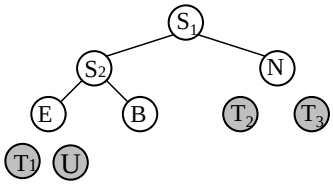
(b) First exchange followed by Max-Heapify



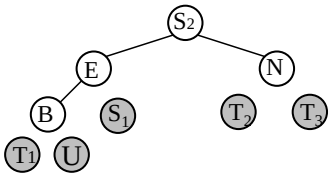
(c)



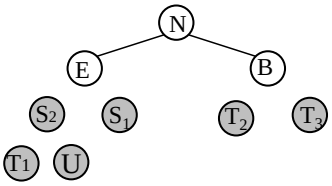
(d)



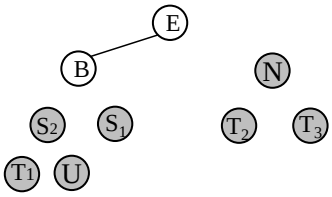
(e)



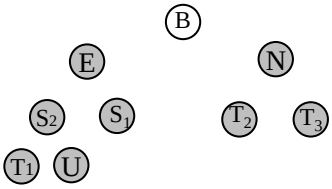
(f)



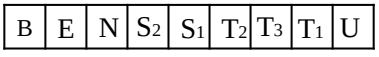
(g)



(h)



(i)



(j) Sorted sequence

Figure 3: Problem 1(c).

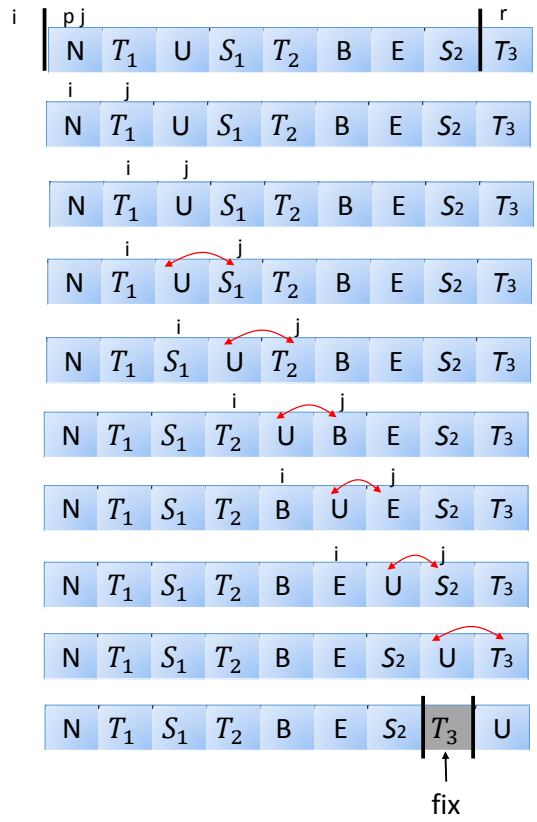


Figure 4: Problem 1(d).

2. $n! > n.2^n > 2^n > 1.2^n > n^{\lg\lg n} = (\lg n)^{\lg n} > n^2 > n\lg n > n > (\sqrt{3})^{\lg n} > \lg^2 n > 1000000$

3. Let $x = \frac{1}{\alpha}$ and $y = \frac{1}{1-\alpha}$. Construct the recurrence tree as Figure 5. We can obviously see that $T(n) = \Theta(nlgn)$.

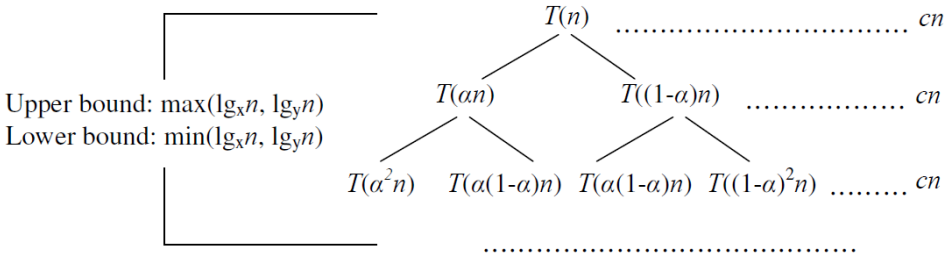


Figure 5: Problem 3.

4. 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:

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

5. (a) $T(n) = 2T(n/4) + 1 = \Theta(n^{1/2})$. Since $\log_4 2 = 1/2$, we have that $1 = O(n^{\log_4 2 - \varepsilon})$ for some const $\varepsilon > 0$. Thus, case1 of the master theorem applies, and $T(n) = \Theta(n^{1/2})$.

   (b) $T(n) = 2T(n/4) + (\sqrt{n}) = \Theta(n^{1/2} \lg n)$. Since $f(n) = (\sqrt{n}) = \Theta(n^{\log_b a}) = \Theta(n^{1/2})$,Thus, case2 of master theorem applies, and $T(n) = \Theta(n^{1/2} \lg n)$.

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

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

6. See Figure 6.

7. See Figure 7.

8. (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	$n-1$

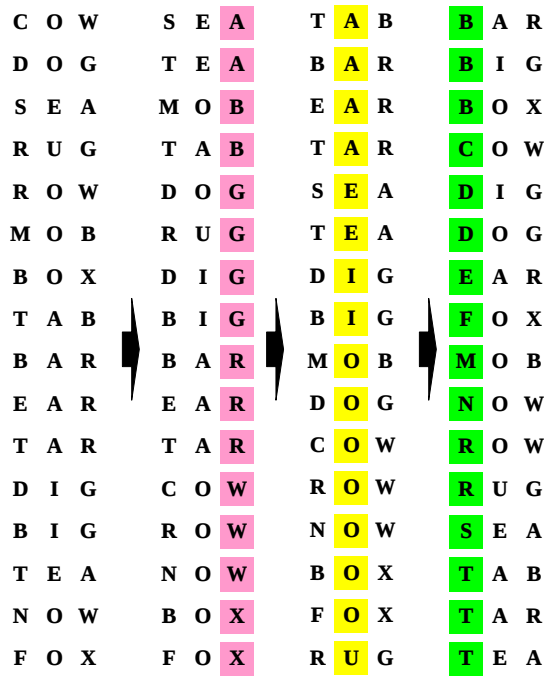


Figure 6: The operation of Radix-Sort.

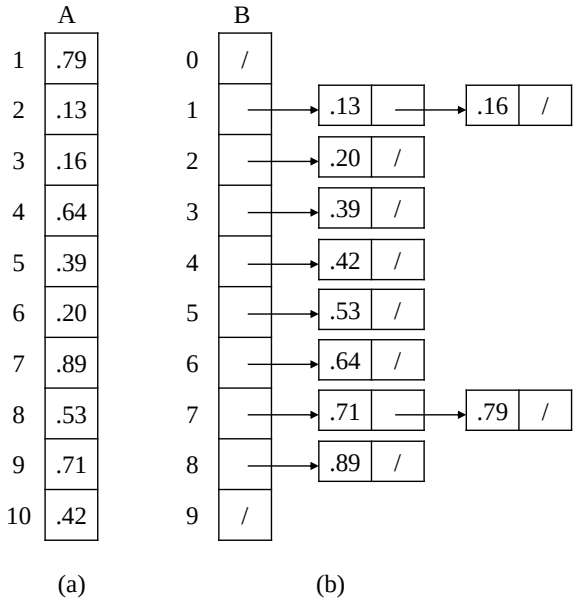


Figure 7: The operation of Bucket-Sort (Problem 6).(a) The input array $A[1...10]$. (b) The array $B[0...9]$ of sorted lists (buckets). The sorted output consists of a concatenation in order of the lists $B[0], B[1], ..., B[9]$.

(b) 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.

(c)
* $T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=1}^{n-1}(n-j+1) + c_5 \sum_{j=1}^{n-1}(n-j) + c_6 \sum_{j=1}^{n-1} t_j + c_7 \sum_{j=1}^{n-1} t_j + c_8(n-1) + c_9(n-1)$
* **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: $T(n) = (\frac{c_4+c_5+c_6+c_7}{2})n^2 + (c_1+c_2+c_3+c_8+c_9+\frac{c_4-c_5-c_6-c_7}{2})n - (c_2+c_3+c_4+c_8+c_9) = \Theta(n^2)$

9. This problem can be solved by a binary-search approach. Let p be the median of $X[1..n]$, and q be the median of $Y[1..n]$. If $p > q$, then the median of the two arrays is in $X[1..\frac{n}{2}]$ or $Y[\frac{n}{2}..n]$ If $p \leq q$, then the median of the two arrays is in $X[\frac{n}{2}..n]$ or $Y[1..\frac{n}{2}]$. Therefore, we can use recursion to solve this problem. For any instance with two sorted arrays $X[1..n]$ and $Y[1..n]$, we first compare the median of them, and then follow the mentioned rules to divide the two arrays into half of the original size. After dividing, we can let the new smaller arrays as the new instance and apply the same strategy recursively.

   Obviously, this binary-search approach divide the problem size into half size every recursion. Thus we have the time complexity, $T(n) = T(n/2) + \theta(1) = O(\lg n)$

10. To show that $(n+a)^b = \Theta(n^b)$, we need to find the constants c_1, c_2, and $n_0 > 0$ such that $0 \leq c_1 n^b \leq (n+a)^b \leq c_2 n^b$ for all $n \geq n_0$. It can be found that $n + a \leq n + |a| \leq 2n$ when $|a| \leq n$, and $n + a \geq n - |a| \geq 1/2n$ when $|a| \leq 1/2n$. Thus, when $n \geq 2|a|$,

$$0 \leq 1/2n \leq n + a \leq 2n$$

Since $b > 0$,

$$0 \leq (1/2n)^b \leq (n+a)^b \leq (2n)^b$$
$$0 \leq (1/2)^b n^b \leq (n+a)^b \leq 2^b n^b$$

Thus, $c_1 = (1/2)^b$, $c_2 = 2^b$, and $n_0 = 2|a|$ can satisfy $0 \leq c_1 n^b \leq (n+a)^b \leq c_2 n^b$.