Algorithms: Homework 1

Li-Yuan Wei

March 25, 2023

Problem 1

Consider the following selection sort algorithm: finding the smallest element and exchanging it with $A[1]$, and then finding the second smallest element and exchanging it with $A[2]$. Continue in this manner for the first $n-1$ elements of A. What are the best-case running time and worst-case running time for this algorithm? Simulate what we did for insertion sort to give your conclusion.

Solution. Psuedocode for selection sort:

SELECTION-SORT(A, n)	Cost	Times
1: **for** $i = 1$ **to** $A.length - 1$	c1	n
2: **min** $= i$	c2	$n-1$
3: **for** $j = i+1$ **to** $A.length$	c3	$\sum_{j=2}^{n}(t_j)$
4: **if** $A[j]$ <A[**min**]	c4	$\sum_{j=2}^{n}(t_j-1)$
5: **min** $= j$	c5	$\sum_{j=2}^{n}(t_j-1)$
6: **if** $i \neq$ **min**	c6	$n-1$
7: exchange $A[i]$ with $A[$**min**$]$	c7	$n-1$

The total running time of selection sort is:

$$T(n) = c_1 n + c_2(n-1) + c_3\sum_{j=2}^{n}(t_j) + c_4\sum_{j=2}^{n}(t_j - 1) + c_5\sum_{j=2}^{n}(t_j - 1) + c_6(n-1) + c_7(n-1)$$

We first analyze the **best-case** running time of this algorithm. Let's assume the input array is a sorted sequence in increasing order. Every $A[i]$ we selected in line 2 is already the smallest element in A. Thus, this algorithm will not execute line $5, 7$ under this condition. However, this algorithm will still iterate the whole sequence, which starts from index j, in line 3 because it will need to check if $A[\textbf{min}]$ is the smallest element.

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_3(\frac{n^2+n}{2} - 1) + c_4(\frac{n^2-n}{2}) + c_6(n-1) \\ &= (\frac{c_3}{2} + \frac{c_4}{2})n^2 + (c_1 + c_2 + \frac{c_3}{2} - \frac{c_4}{2} + c_6)n - (c_2 + c_6) \end{aligned}$$

We conclude the **best-case** running time for selection sort is $\Theta(n^2)$ since n^2 is the dominant term in $T(n)$. ■
Additionally, we assume the input array as a decreasing sorted sequence. This is the worst-case scenario because the program will execute line 5 whenever it compares $A[\textbf{min}]$ with $A[j]$ in line 4, and executes exchange in line 7.

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_3(\frac{n^2+n}{2} - 1) + c_4(\frac{n^2-n}{2}) + c_5(\frac{n^2-n}{2}) + c_6(n-1) + c_7(n-1) \\ &= (\frac{c_3}{2} + \frac{c_4}{2} + \frac{c_5}{2})n^2 + (c_1 + c_2 + \frac{c_3}{2} - \frac{c_4}{2} - \frac{c_5}{2} + c_6 + c_7)n - (c_2 + c_3 + c_6 + c_7) \end{aligned}$$

Therefore, we conclude that the **worst-case** running time for selection sort is still $\Theta(n^2)$. ■

Problem 2

What are the best-case running time and worst-case running time for bubblesort? Please refer to the following pseudo-code of bubblesort. Simulate what we did for insertion sort to give your conclusion.

Solution. Psuedocode for bubble sort:

BUBBLE-SORT(A, n)	Cost	Times
1: **for** $i = 1$ **to** $A.length - 1$	c1	n
2: **for** $j = A.length$ **downto** $i + 1$	c2	$\sum_{j=2}^{n}(t_j)$
3: **if** $A[j] < A[j-1]$	c3	$\sum_{j=2}^{n}(t_j - 1)$
4: exchange $A[j]$ with $A[j-1]$	c4	$\sum_{j=2}^{n}(t_j - 1)$

The total running time of bubble sort is:

$$T(n) = c_1 n + c_2 \sum_{j=2}^{n}(t_j) + c_3 \sum_{j=2}^{n}(t_j - 1) + c_4 \sum_{j=2}^{n}(t_j - 1)$$

For the **best-case** running time, we assume the input is a sorted sequence in increasing order. Therefore, $t_j = 1$ in line 4 because there will need no exchange.

$$\begin{aligned} T(n) &= c_1 n + c_2(\frac{n^2+n}{2} - 1) + c_3(\frac{n^2-n}{2}) \\ &= (\frac{c_2}{2} + \frac{c_3}{2})n^2 + (c_1 + \frac{c_2}{2} - \frac{c_3}{2})n - c_2 \end{aligned}$$

The **best-case** running time for bubble sort is $\Theta(n^2)$. ■

Additionally, the **worst-case** scenario for bubble sort is to exchange everytime the program executes line $3, 4$.

$$\begin{aligned} T(n) &= c_1 n + c_2(\frac{n^2+n}{2} - 1) + c_3(\frac{n^2-n}{2}) + c_4(\frac{n^2-n}{2}) \\ &= (\frac{c_2}{2} + \frac{c_3}{2} + \frac{c_4}{2})n^2 + (c_1 + \frac{c_2}{2} - \frac{c_3}{2} - \frac{c_4}{2})n - c_2 \end{aligned}$$

The **worst-case** running time for buble sort is still $\Theta(n^2)$. ■

Problem 3

Compare the following functions, and list them in a non-decreasing order:
$n, \lg n, \lg\lg n, 5^{\lg n}, n!, n\lg n, \lg n^2, \lg^2 n, e^n$

Solution. $\lg\lg n < \lg n < \lg n^2 < \lg^2 n < n < n\lg n < 5^{\lg n} < e^n < n!$
$\lg n^2 = 2\lg n$
$5^{\lg n} = n^{\lg 5} \approx n^{2.321}$
$\lg^2 n = \lg n \lg n > \lg n$
$\lg(n!) = \Theta(n\lg n)$
$\lg(e^n) = n\lg e$ ■

Problem 4

(a) Is $5n^2 + 2n = O(n^2)$? If so, please give its c and n_0 according to the definition.
(b) Is $\log_3^2 n = o(\sqrt[3]{n})$? If so, please prove it.

Solution. (a) Yes, $5n^2 + 2n = \mathcal{O}(n^2)$. From the definition of Big $\mathcal{O}$, we need to find postive constants c, n_0 to satisfy $5n^2 + 2n \le cn^2, \forall n \ge n_0$.

$$\begin{aligned} f(n) &= 5n^2 + 2n \le cn^2 = cg(n) && \text{divide both functions by } n^2 \\ &= 5 + 2/n \le c \end{aligned}$$

From the above function, we have $n_0 = 1, c = 7$ to show that $f(n) = \mathcal{O}(n^2)$. ■

Solution. (b) Let $f(n) = \log_3^2 n, g(n) = \sqrt[3]{n}$.

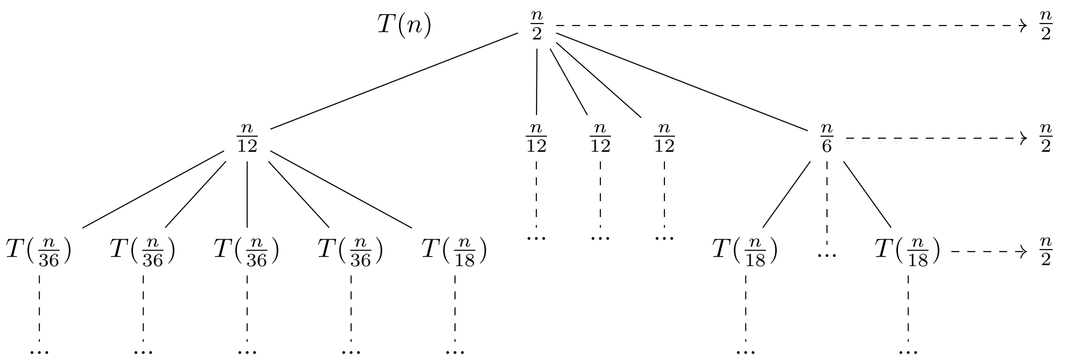
$$\begin{aligned} \lim_{n\to\infty} \frac{f(n)}{g(n)} &= \lim_{n\to\infty} \frac{\log_3^2 n}{\sqrt[3]{n}} && \text{Applying L'Hospital's Rule} \\ &= \lim_{n\to\infty} \frac{f'(n)}{g'(n)} = \lim_{n\to\infty} \frac{2\log_3 n}{n\ln 3 \cdot \frac{1}{3}n^{-\frac{2}{3}}} = \lim_{n\to\infty} \frac{6\log_3 n}{n^{\frac{1}{3}}\ln 3} \\ &= \lim_{n\to\infty} \frac{f''(n)}{g''(n)} = \lim_{n\to\infty} \frac{6}{n\ln 3 \cdot \ln 3 \frac{1}{3}n^{-\frac{2}{3}}} = \lim_{n\to\infty} \frac{18}{n^{\frac{1}{3}}\ln^2 3} = 0 \end{aligned}$$

The limit of $\frac{f(n)}{g(n)}$ as n approaches to ∞ is 0. Therefore, we proved that $\log_3^2 n = o(\sqrt[3]{n})$. ■

Problem 5

Use recursion tree to find the tight bound for the following recurrence: $T(n) = 4T(n/6) + T(n/3) + n/2$

Solution. Recursion tree of $T(n) = 4T(n/6) + T(n/3) + n/2$



The minimum height of this recurrsion tree is $\log_6 n$, on the other hand, the maximum height of this recurrsion tree is $\log_3 n$. Thus, we can get an approximate running time for our recurrence $\frac{n}{2}\log_6 n \leq T(n) \leq \frac{n}{2}\log_3 n$. As listed above, we concluded that the running time of this recurrence is $\Theta(n \lg n)$. ■

Problem 6

Use the master method to solve the following recurrence:
(a) $T(n) = 4T(n/2) + n$
(b) $T(n) = 4T(n/2) + n^2$
(c) $T(n) = 4T(n/2) + n^3$

Solution. (a) Based on master method, we have $a = 4, b = 2, f(n) = n$, which implies that $n^{\log_b a} = n^{\log_2 4} = n^2 = \Theta(n^2)$. Since $f(n) = n = \mathcal{O}(n^{2-\epsilon})$, for any constant $0 < \epsilon \leq 1$, we can solve this recurrence by applying case 1 of master method. The solution is $T(n) = \Theta(n^{\log_b a}) = \Theta(n^{\log_2 4}) = \Theta(n^2)$. ■

Solution. (b) We have $a = 4, b = 2$, and our $f(n) = n^2 = \Theta(n^{\log_b a}) = \Theta(n^{\log_2 4}) = \Theta(n^2)$. Thus, we can directly apply case 2 of master method to solve this recurrence. The solution is $T(n) = \Theta(n^{\log_b a}\lg n) = \Theta(n^2 \lg n)$. ■

Solution. (c) We have $a = 4, b = 2$, and our $f(n) = n^3$. First, we assume that we can solve this recurrence with case 3 because $f(n) = n^3 = \Omega(n^{(\log_b a)+\epsilon}) = \Omega(n^{(\log_2 4)+\epsilon}) = \Omega(n^{2+\epsilon})$, for $\epsilon = 1$. In addition, we verify that $af(n/b) = 4(n/2)^3 = 1/2(n^3) \leq 1/2(n^3) = cf(n)$ for $c = 1/2$ is true. As a result, we can conlude that the solution of this recurrence is $T(n) = \Theta(f(n)) = \Theta(n^3)$. ■

Problem 7

Can the master method be applied to the recurrence $T(n) = 4T(n/2) + n^2 \lg n$? Why or why not?

Solution. We have $a = 4, b = 2, f(n) = n^2 \lg n$ from our recurrence, where we have $n^{\log_b a} = n^{\log_2 4} = n^2$. $f(n) = n^2 \lg n = \Omega(n^{2+\epsilon})$, where we cannot find a constant $\epsilon > 0$. In addition, we cannot find any constant $c < 1$ that satisfies $af(n/b) = 4(n/2)^2 \lg n/2 = n^2 \lg n/2 \leq cn^2 \lg n$. To verify our statement,

$$\begin{aligned}
n^2 \lg n/2 &\leq cn^2 \lg n && \text{divide both by } n^2 \\
\lg n/2 &\leq c \lg n && \lg(ab) = \lg a + \lg b = \lg(n2^{-1}) = \lg n + \lg 2^{-1} \\
\lg n - \lg 2 &\leq c \lg n && \text{divide both by } \lg n \\
1 - \frac{\lg 2}{\lg n} = 1 - \frac{1}{\lg n} &\leq c && (1)
\end{aligned}$$

From (1), as n approaches to ∞, (1) will become $1 \leq c$, which did not satisfy the conditions in case 3. Therefore, we have concluded that we cannot solve this recurrence with master method. ■

Problem 8

Use the idea of changing variables to solve the following recurrence: $T(n) = 3T(\sqrt[3]{n}) + \lg n$

Solution. Let $m = \lg n, n = 2^m, n^{\frac{1}{3}} = 2^{\frac{m}{3}}$.

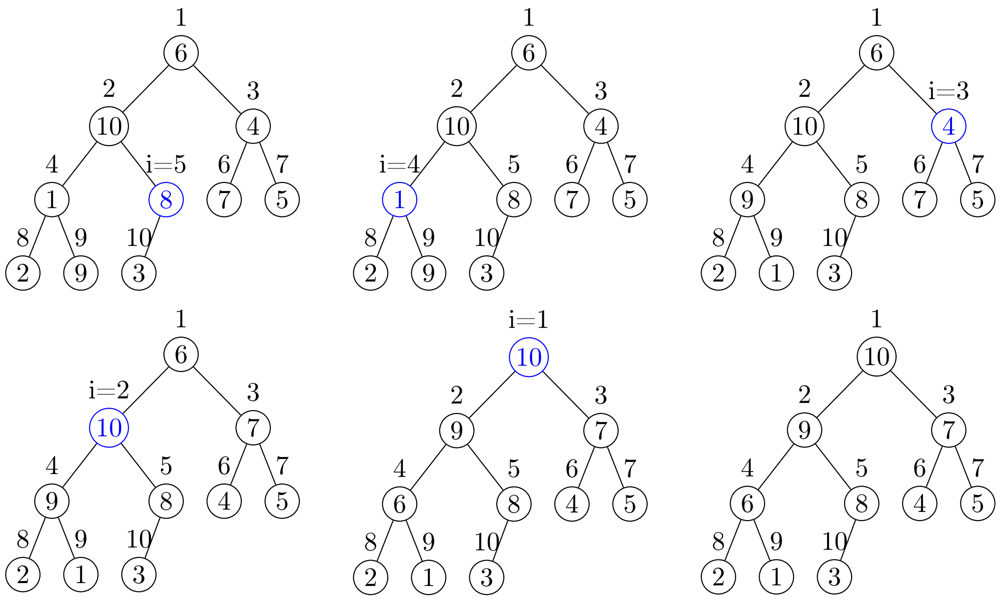
$$\begin{aligned}
&T(n) = 3T(\sqrt[3]{n}) + \lg n && \text{replace } n \text{ with } 2^m \\
\equiv\ &T(2^m) = 3T(2^{\frac{m}{3}}) + m && \text{replace } T(2^m) \text{ with } S(m) \\
\equiv\ &S(m) = 3S(\frac{m}{3}) + m && \text{apply master method from this recurrence} \qquad (1)
\end{aligned}$$

From (1), we have $a = 3, b = 3, f(m) = m$. Since $f(m) = m = \Theta(m^{\log_3 3}) = \Theta(m)$, we can apply case 2 of master method. Hence, the solution is $S(m) = \Theta(m \lg m)$. However, we want to solve $T(n)$, not $S(m)$. By applying $m = \lg n$ back to $S(m)$, we can have the solution for our original recurrence $T(n) = \Theta(\lg n \lg \lg n)$. ■

Problem 9

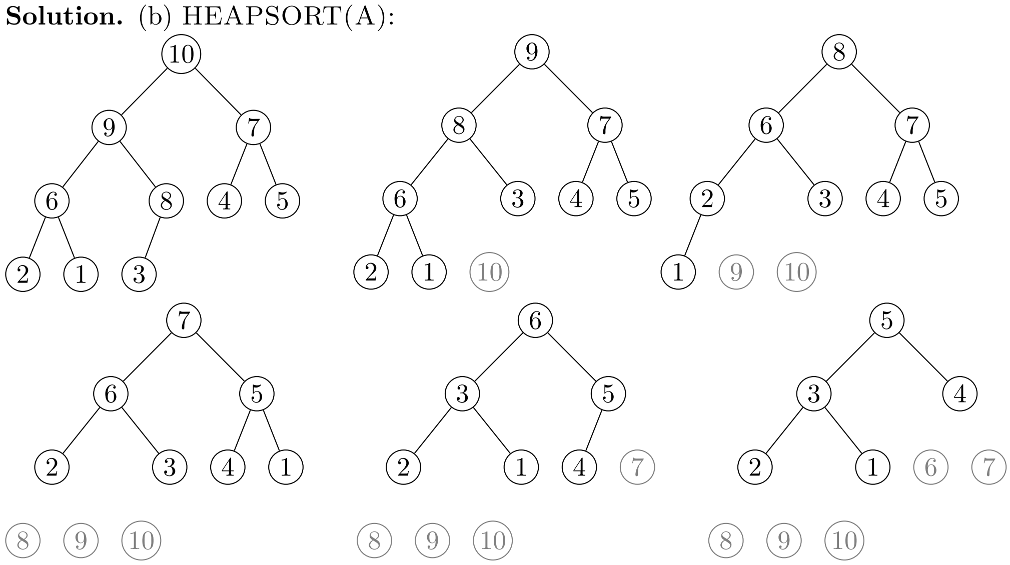
(a) Given an array $A[1..10] =< 6, 10, 4, 1, 8, 7, 5, 2, 9, 3 >$, what is the resulting heap using the BUILD-MAX-HEAP(A) procedure listed in the textbook? You should show your result step by step, using Figure 6.3 in the textbook as a model. (b) Use Figure 6.4 as a model, illustrate the operation of HEAPSORT on the heap you built in (a).

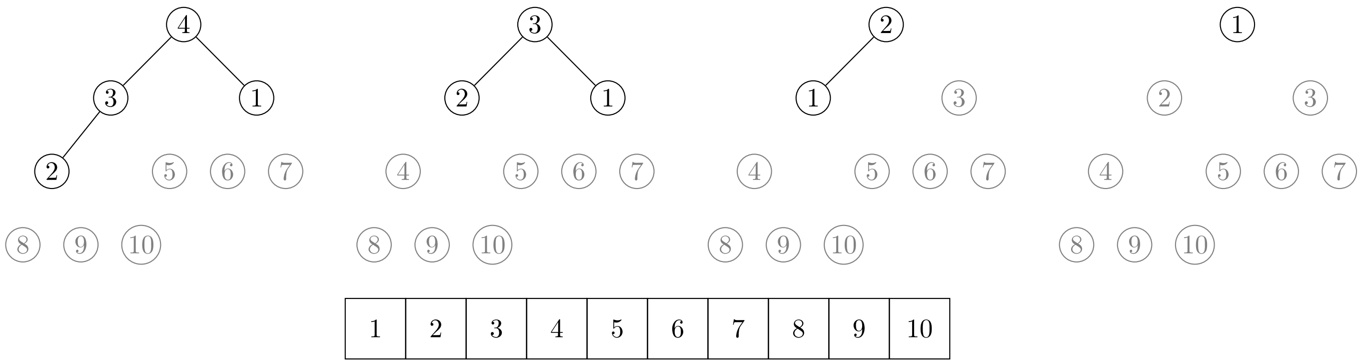
Solution. (a) Result of each step of BUILD-MAX-HEAP(A), starting from left to right, top to bottom:



■

Solution. (b) HEAPSORT(A):





Problem 10

The operation HEAP-DELETE(A, i) deletes the item in node i from heap A. Give an implementation of HEAP-DELETE that runs in $O(\lg n)$ time for an n-element max-heap.

Solution. Psuedocode for heap delete:

```
HEAP-DELETE(A, i)
if i > A.heap-size
    error A[i] does not exist
key = A[i]
A[i] = A[A.heap-size]
A.heap-size = A.heap-size − 1
if A[i] < key
    MAX-HEAPIFY(A, i)
else
    key1 = A[i]
    A[i] = key
    HEAP-INCREASE-KEY(A, i, key1)
```