CMPSC 465 Data Structures & Algorithms
Fall 2021 Antonio Blanca and David Koslicki

HW 3

1. (20 pts.) Problem 1

(a) After Build-Heap: $[1,5,2,7,6,4,3,9,8,10]$
After 1st iteration of the loop: $[2,5,3,7,6,4,10,9,8,1]$
After 2nd iteration of the loop: $[3,5,4,7,6,8,10,9,2,1]$
After 3rd iteration of the loop: $[4,5,8,7,6,9,10,3,2,1]$

(b) Proof: Assume the element at position $\lfloor \frac{n}{2} \rfloor$ is an internal node, not a leaf. Then, its left child should be at position $2 \cdot \lfloor \frac{n}{2} \rfloor + 1 \geq 2 \cdot \frac{n}{2} - 1 + 1 = n > n-1$, which is the index of the last element in heap. Therefore, there is a contradiction. Thus, the element at position $\lfloor \frac{n}{2} \rfloor$ can't have any children, meaning it must be a leaf. The same reasoning can be applied to other positions in the statement since they are even larger than $\lfloor \frac{n}{2} \rfloor$. So, the nodes at positions $\lfloor \frac{n}{2} \rfloor, \ldots, n-1$ must be the leaves.

(c) Consider an array $[2,3,4,1]$. If we increase i from 0 to $\lfloor \frac{n}{2} \rfloor - 1$, there is no swap in the first iteration. In the second iteration, 3 swaps with 1. Then, the loop and the Build-Heap procedure are finished with the array $[2,1,4,3]$ which doesn't satisfy the min heap property since 2 is greater than its left child 1. So, increasing i from 0 to $\lfloor \frac{n}{2} \rfloor - 1$ can't guarantee that the array we end up obtaining from Build-Heap has the min heap property.

2. (15 pts.) Problem 2

We keep $M+1$ counters, one for each of the possible values of the array elements. We can use these counters to compute the number of elements of each value by a single $O(n)$-time pass through the array. Then, we can obtain a sorted version of x by filling a new array with the prescribed numbers of elements of each value, looping through the values in ascending order. Notice that the $\Omega(n \log n)$ bound does not apply in this case, as this algorithm is not comparison-based.

3. (15 pts.) Problem 3

(a) We get the recurrence $T(n) = T(n/2) + \Theta(1)$ for the running time of Binary Search. Applying the master theorem with $a = 1$, $b = 2$, $d = 0$ and $\log_b a = 0$, we note that we are in the second case, and so $\Theta(n^{\log_2 1} \log(n)) = \Theta(\log(n))$

(b) For Ternary Search the recurrence is $T(n) = T(n/3) + \Theta(1)$. Applying the master theorem with $a = 1$, $b = 3$, and $d = 0$, we have that $\log_3 1 = 0 = d$. So, we are in the second case, and we have that $\Theta(n^{\log_3 1} \log(n)) = \Theta(\log(n))$

4. (20 pts.) Problem 4

Let $m = \lfloor \frac{n}{2} \rfloor$, first we divide the input sequence in two halves ($A_1 = a_1, \ldots, a_m$ and $A_2 = a_{m+1}, \ldots, a_n$). Also, let $T(n)$ be the time it takes to sort the input sequence of n numbers and to count the number of inversions. First, we recursively sort, and count each half, which takes $2T(\frac{n}{2})$. Then, we should merge two halves, and count the number of inversions between the two halves. We can do this as follows. Remember that the two halves are A_1 and A_2, and let B_1, and B_2, be the output of recursive call on A_1 and A_2. Let i be the pointer to array B_1, and j be the pointer to array B_2. Also initialize i and j to zero (for example $i = 0$ points to first element of array B_1). (Merge part:) Compare the first elements of B_1, and B_2, and increase the pointer of the smaller element by one, and put it in output array S.(Count part:) Also, whenever we increase the B_2

pointer by one, we increase the total number of inversions by $\texttt{length}(B_1) - i$, since whenever we increase j (or B_2 pointer) by one, $B_2[j]$ should be less than $B_1[i : \frac{n}{2}]$, which makes $\texttt{length}(B_1) - i$ inversions. To get more details, You can see the pseudocode in algorithm 1.

Algorithm 1 sort, and count

```
Input: an array A = [a_1, a_2, ..., a_n], and its length n
function SORT-COUNT(A, n)
    if n == 1 then return A[0], 0
    end if
    A_1 ← [a_1, ..., a_⌊n/2⌋], A_2 ← [a_⌊n/2⌋+1, ..., a_n]
    B_1, cnt_1 ← sort-count(A_1, ⌊n/2⌋)
    B_2, cnt_2 ← sort-count(A_2, ⌈n/2⌉)
    i ← 0
    j ← 0
    S ← an array of length n
    k ← 0
    cnt ← 0
    while i < length(B_1) and j < length(B_2) do
        if B_2[j] ≥ B_1[i] then
            S[k] = B_1[i], i ← i + 1
        else
            S[k] = B_2[j], j ← j + 1, cnt ← cnt + length(B_1) − i
        end if
        k ← k + 1
        if i = length(B_1) then
            S[k : n − 1] = B_2[k : n − 1]
        end if
        if j = length(B_2) then
            S[k : n − 1] = B_1[k : n − 1]
        end if
    end while
    return S, cnt_1 + cnt_2 + cnt
end function
```

Now since the merge part takes at most n iteration of while loop, we can say the running time for merge is $O(n)$. So, the recursive relation would be:

$$T(n) = 2T(\frac{n}{2}) + O(n) \tag{1}$$

If we use the master theorem for the above recursive relation, we have $T(n) = O(n \log n)$

5. (20 pts.) Problem 5

All recurrences can be solved by unrolling the recurrence and carefully arranging the terms, following the level-by-level approach from lectures (which is equivalent), or using various form of the Master Theorem. However, substitution or induction method is sometimes more convenient to showing the recurrence:

(a) In class, we saw that if $T(n) = a \cdot T(n/b) + O(n^d)$ for some $a > 0, b > 1$ and $d >= 0$, then:

$$T(n) = \begin{cases} O\left(n^d\right) & \text{if } d > \log_b a \\ O\left(n^d \log n\right) & \text{if } d = \log_b a \\ O\left(n^{\log_b a}\right) & \text{if } d < \log_b a \end{cases}$$

Using master Theorem ($a = 9, b = 3, d = 2$), we get $T(n) = O(n^2 \log n)$.

(b) $T(n) = 2T(n/5) + \sqrt{n} = O(\sqrt{n})$ by the Master theorem ($a = 2, b = 5, d = \frac{1}{2}$).

(c) $T(n) = T(n-1) + c^n = \sum_{i=1}^{n} c^i + T(0) = c(\frac{c^n - 1}{c-1}) + T(0) = O(c^n)$.

(d) $T(n) = T(n-1) + n^c = \sum_{i=1}^{n} i^c + T(0) = O(n^{c+1})$.

(e) $T(n) = 5T(n/4) + n = O(n^{\log_4 5})$ by the Master theorem.

(f) Unrolling, we have

$$T(n) = 1.5^n + 1.5^{n/2} + 1.5^{n/4} + 1.5^{n/8} + \ldots$$

Then $T(n) \leq \sum_{i=0}^{n} 1.5^i = O(1.5^n)$. Hence, $T(n) = O(1.5^n)$.

(g) $T(n) = 49T(n/25) + n^{3/2} \log n = O(n^{3/2} \log n)$. By unrolling we have:

$$\begin{aligned} \sum_{i=0}^{\log_{25} n} \left(\frac{49}{25^{3/2}}\right)^i n^{3/2} \log\left(\frac{n}{25^i}\right) &< \sum_{i=0}^{\log_{25} n} \left(\frac{49}{125}\right)^i O(n^{3/2} \log n) \\ &= O(n^{3/2} \log n) \sum_{i=0}^{\log_{25} n} \left(\frac{49}{125}\right)^i \\ &= O(n^{3/2} \log n) O(1) \\ &= O(n^{3/2} \log n) \end{aligned} \tag{2}$$

Note that in latter equation we used the fact that the geometric series will be $O(1)$, since $\frac{49}{125} < 1$.

(h) By induction we will show that $T(n) = O(n)$. Assume for $k \leq n-1$, $T(k) \leq ck$, where $c \geq 8$. Then we have:

$$T(n) = T(\frac{n}{2}) + T(\frac{n}{4}) + T(\frac{n}{8}) + n \leq \frac{7}{8}cn + n \leq cn \tag{3}$$

As a result, we proved that $T(n) \leq 8n$, which means that $T(n) = O(n)$.

(i)
- First method (Unrolling).
  Let us consider the upper bound and suppose $T(n) \leq T(3n/5) + T(2n/5) + cn$ for a suitable constant $c > 0$. Note by unrolling that the size of each sub-problem at level k is of the form $S_{k,i} = \left(\frac{3}{5}\right)^i \left(\frac{2}{5}\right)^{k-i} n$ for some $i = 0, \ldots, k$. Moreover, there are $\binom{k}{i}$ sub-problems of size $S_{k,i}$ at level k. The height of the recurrence tree is $\log_{5/2} n$. Hence,

$$T(n) \leq \sum_{k=0}^{\log_{5/2} n} \sum_{i=0}^{k} \binom{k}{i} c \left(\frac{3}{5}\right)^i c \left(\frac{2}{5}\right)^{k-i} n = \sum_{k=0}^{\log_{5/2} n} c^2 n = O(n \log n).$$

- Second method (Induction).
  As the induction step, assume that for all $k < n$, we have $T(k) \leq c_1 k \log k$. We want to show that there exists c_1 such that for all large enough n, $T(n) \leq c_1 n \log n$. Since both $\frac{2n}{5}$, and $\frac{3n}{5}$ are less than n according to induction assumption we have: $T(\frac{2n}{5}) \leq c_1 \frac{2n}{5} \log \frac{2n}{5}$, and $T(\frac{3n}{5}) \leq c_1 \frac{3n}{5} \log \frac{3n}{5}$.

Now using recurrence relation for $T(n)$ we can write:

$$\begin{aligned}
T(n) \leq T(\frac{3n}{5}) + T(\frac{2n}{5}) + cn &\leq c_1 \frac{3n}{5} \log \frac{3n}{5} + c_1 \frac{2n}{5} \log \frac{2n}{5} + cn \\
&= c_1 \frac{3n}{5} \log \frac{3}{5} + c_1 \frac{3n}{5} \log n + c_1 \frac{2n}{5} \log \frac{2}{5} + c_1 \frac{2n}{5} \log n + cn \\
&= c_1 n \log n + c_1 n (\frac{3}{5} \log \frac{3}{5} + \frac{2}{5} \log \frac{2}{5}) + cn \\
&= c_1 n \log n + cn - c_1 n (\frac{3}{5} \log \frac{5}{3} + \frac{5}{2} \log \frac{2}{5})
\end{aligned} \tag{4}$$

Now, note that if we let $c_1 = \frac{c}{\frac{3}{5} \log \frac{5}{3} + \frac{5}{2} \log \frac{2}{5}}$, and replace it on latter equation, we get $cn - c_1 n (\frac{3}{5} \log \frac{5}{3} + \frac{5}{2} \log \frac{2}{5}) = 0$, which means that $T(n) \leq c_1 n \log n$. So, induction step is complete, and we proved that $T(n) = O(n \log n)$

(j) Note by unfolding that

$$T(n) = n^{\frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{2^k}} T(n^{\frac{1}{2^k}}) + 10kn.$$

(This can also be proved, for example, by induction.) Now if we let $k = \log_2 \log_2 n$, we have $n^{\frac{1}{2^k}} = 2$ which is constant. Since $\frac{1}{2} \leq \frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{2^k} \leq 1$, we have $T(n) = \Theta(n \log \log n)$.

Rubric:

Problem 1, total points 20

(a) 6 points
3 points: provide correct resulting array for Build-Heap
1 point: provide correct resulting array for the 1st iteration
1 point: provide correct resulting array for the 2nd iteration
1 point: provide correct resulting array for the 3rd iteration

(b) 7 points
4 points: explain the child issue
3 points: explain the child issue is applicable to all the elements in the specified positions

(c) 7 points
3 points: provide a valid counterexample for increasing i from 0 to $\lfloor \frac{n}{2} \rfloor - 1$
4 points: explain why this increasing i from 0 to $\lfloor \frac{n}{2} \rfloor - 1$ fails on this counterexample

Problem 2, 15 pts

- 8 points for proof of time complexity
- 7 points for the justification of why $\Omega(n \log n)$ doesn't apply to the case where M is small and the time is linear time.

Problem 3,15 pts

- 4 points for recurrence $T(n) = T(n/2) + \Theta(1)$, 4 points for running time $\Theta(n)$
- 3.5 points for recurrence $T(n) = T(n/3) + \Theta(1)$, 3.5 points for running time $\Theta(n)$

Problem 4, 20 pts.

The algorithm includes two recursive call, and then a merge and count part.
Designing a correct merge and count procedure: 5
The correctness of algorithm: 5
Deriving the recurrence relation: 5
Showing the run-time from the recurrence relation: 5

Problem 5, 30 pts

There are total of 10 parts, each part is worth 3 pts. For the parts that can be solve by master theorem, only correct answer get the full points, and there is not partial credit for these parts. For, the parts that master theorem can not be applied, unrolling correctly is worth 1.5 pts, and showing the final answer is worth 1.5 pts.