CS 581 Homework 3 Solution

1

True or False. Justify answer.

1.1

$A \in \mathcal{O}(B) \Rightarrow A = o(B)$.

$$\text{False. } n^2 \in \mathcal{O}(n^2), \text{ However}$$
$$n^2 \notin o(n^2).$$

1.2

$A \in \mathcal{O}(B) \land A \in \Omega(B) \Rightarrow A \sim B$.

$$\text{False. } A \in \mathcal{O}(B) \land A \in \Omega(B) \Rightarrow A \in \Theta(B)$$
$$\Rightarrow c_1 \leq |\lim_{n\to\infty}(\tfrac{A}{B})| \leq c_2$$
$$\text{For some } c_1, c_2 > 0.$$
$$\Rightarrow |\lim_{n\to\infty}(\tfrac{A}{B})| \in \mathbb{R}_{>0}.$$

Since c_1 and c_2 do not have to be $= 1$, the statement is false.

1.3

Worst case time complexity of merge sort is $\mathcal{O}(n^2)$.

True. The time complexity of merge sort can be expressed as $T(n) = 2T(\frac{n}{2}) + \Theta(n)$. This is because merge sort must split the array in two and recurse on both halves, and also merge n elements. Using the master method, the complexity is $T(n) = \mathcal{O}(n \log n) \in \mathcal{O}(n^2)$.

1.4

Worst case time complexity of quick sort is $\mathcal{O}(n^2)$.

True. The worst case in quick sort occurs when the pivot is chosen in order from least to greatest or greatest to least in the file. This causes quick sort to be recursed n times each with $\mathcal{O}(n)$ comparisons. Therefore it is indeed $\mathcal{O}(n^2)$.

1.5

Two sorted lists of size m and n can be merged with only $\mathcal{O}(min(m, n))$ extra space (in linear time).

True. We can merge forwards or backwards depending on if the larger array should appear before or after the smaller array in the output. If the smaller array contains smaller numbers in the output, we merger forwards. If the larger array contains smaller numbers in the output, we merger backwards. In either case, we start using the extra $(min(m, n))$ space, and when it fills up, we start outputting to the larger array (no uninspected element will be overwritten).

1.6

Since merge sort is $\mathcal{O}(n \log n)$ and insertion sort is $\mathcal{O}(n^2)$ on average, we should always use merge sort preferably than insertion sort.

False. Insertion sort has low overhead, which makes it good for small files. It is also better for nearly sorted files.

2

2.1

Why would someone want to shuffle the list before applying quicksort on it?

Randomizing the file before applying quicksort reduces the probability of getting the worst case run-time, because the pivots are randomized and are not likely to be chosen in increasing or decreasing order.

2.2

Name two pivot selection strategies for quicksort other than just using the first or the last item.

- "median-of-three"
- Random pivot selection
- Use a linear selection algorithm to select the median value

2.3

Name one scenario where the stability of sorting matters.

Stable sorting algorithms are useful for stacking when having to sort on multiple indexes. For instance, if you want to sort a file where the elements are first and last names, then you can stable sort by last names, then stable sort by first names and retain the sorted last names.

2.4

What is the recurrence of Strassen's matrix multiplication algorithm?

$$T(n) = \begin{cases} \Theta(1), & \text{if } n = 1. \\ 7T(n/2) + \Theta(n^2), & \text{if } n > 1. \end{cases} \tag{1}$$

3

Given a partition scheme of quicksort, what is the time complexity when all elements in the array are the same and when the array is already sorted? Briefly explain why for both cases.

Using this scheme on an already sorted array yields the worst possible performance for quicksort. This is because the pivot chosen is always the first element of the array, and quicksort is recursively called on an array of size 1 and size $(n-1)$ every recurse. That is, $\mathcal{O}(n)$ comparisons per recursion and n recursions (i.e. $\mathcal{O}(n^2)$).

Using this scheme on an array of identical elements yields $\Theta(n \log n)$. Since the pivot is chosen to be the median of the array each time, you are recursing on two partitions of size $\frac{n}{2}$. The recurrence for this instance is $T(n) = 2T(\frac{n}{2}) + \Theta(n) = \Theta(n \log n)$.

4

Sort array {7, 3, 4, 8, 2, 1, 7, 3, 1, 6, 4, 5, 4} in non-decreasing order using counting sort. Show work.

Index:	1	2	3	4	5	6	7	8
Count:	2	1	2	3	1	1	2	1
Accumulate:	2	3	5	8	9	10	12	13

Iterate the array again, place the element to the position according to the count, decrement the count and move to next element. Sorted Array:

											7	
				3							7	
				3			4				7	
				3			4				7	8
		2		3			4				7	8
	1	2		3			4				7	8
	1	2		3			4			7	7	8
	1	2	3	3			4			7	7	8
1	1	2	3	3			4			7	7	8
1	1	2	3	3			4		6	7	7	8
1	1	2	3	3		4	4		6	7	7	8
1	1	2	3	3		4	4	5	6	7	7	8
1	1	2	3	3	4	4	4	5	6	7	7	8

The Accumulate array decreases each element indexed by the index array that corresponds to the unsorted array. Each decrease step was not shown for simplicity.

There are a few slightly different implementations of counting sorts. This is the one the textbook uses.

5

Sort array {789, 123, 456, 567, 234, 11, 1, 2} in non-decreasing order using radix sort. Show work.

Sorted Array:

One's:	11	01	02	123	234	456	567	789
Ten's:	01	02	11	123	234	456	567	789
Hundred's:	001	002	011	123	234	456	567	789

Each digit can take 10(k=10) values, there are 3(d=3) digits and 8(n=8) numbers. So we have 10 bins, after the first pass (sort by one's), each bin (empty bins are omitted) has the following configuration:

- bin 1: 11, 01
- bin 2: 2
- bin 3: 123
- bin 4: 234
- bin 6: 456
- bin 7: 567
- bin 9: 789

Then restore elements to the list from each bin. The elements in the same bin should remain the same order (like a queue). This process iterates the bins and thus takes $\Theta(k)$ time. The later two passes are similar, the whole process takes $\Theta(d(n+k))$ time.

6

Use master theorem to put asymptotic bounds on each of the following recurrences.

6.1

$T(n) = 7T(\frac{n}{2}) + n^2 \log^2 n$

$$\text{First case: } n^2 \log^2 n \in \mathcal{O}(n^{\log 7 - \epsilon})$$
$$\therefore T(n) \in \Theta(n^{\log 7})$$

6.2

$T(n) = 7T(\frac{n}{3}) + n^2 \log^3 n$

$$\text{Third Case: } n^2 \log^3 n \in \Omega(n^{\log_3 7 + \epsilon}).$$
$$\text{If } 7(\tfrac{n}{3})^2 \log^3(\tfrac{n}{3}) \leq cn^2 \log^3 n \text{ for some } 0 < c < 1. \text{ Then } T(n) \in \Theta(n^2 \log^3 n).$$

$$7(\tfrac{n}{3})^2 \log^3(\tfrac{n}{3}) = \tfrac{7}{9}n^2 \log^3(\tfrac{n}{3}).$$

Since $\log(\frac{n}{3}) < \log(n)$ for $n > 1$, $\frac{7}{9}n^2 \log^3(\frac{n}{3}) \leq cn^2 \log^3 n$ holds true.

$$\therefore T(n) \in \Theta(n^2 \log^3 n)$$

6.3

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

$$\text{Second Case: } 1 \in \Theta(n^{\log 1}) = \Theta(1)$$

$$\therefore T(n) \in \Theta(\log n)$$

6.4

$T(n) = T(n^{0.5}) + 1$

$$\text{Let } n = 2^k \text{ Then}$$

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

$$\text{Let } T'(k) = T(2^k). \text{ Then}$$

$$T'(k) = T'(k/2) + 1.$$

$$\text{This is the Second Case: } 1 \in \Theta(1)$$

$$\therefore T(n) \in \Theta(\log k) = \Theta(\log \log n).$$

7

Consider the regularity condition $af(\frac{n}{b}) \leq cf(n)$ for some constant $c < 1$, which is part of case 3 of the master theorem. Give an example of constants $a \geq 1$ and $b > 1$ and a function $f(n)$ that satisfies all the conditions in case 3 of the master theorem except the regularity condition.

$$T(n) = T(\tfrac{n}{2}) + n(sin(n - \tfrac{\pi}{2}) + 2)$$

This dissatisfies the regularity condition, since

$\frac{n}{2}(sin(\frac{n}{2} - \frac{\pi}{2}) + 2) \leq cn(sin(n - \frac{\pi}{2}) + 2) \equiv c \geq \frac{\frac{1}{2}(sin(\frac{n}{2} - \frac{\pi}{2}) + 2)}{sin(n - \frac{\pi}{2}) + 2}$ cannot be satisfied for large n.

An example would be $n = 2\pi k$ where k is odd.

$$\Rightarrow c \geq \frac{\frac{1}{2}(sin(\pi k - \frac{\pi}{2}) + 2)}{sin(2\pi k - \frac{\pi}{2}) + 2}$$

$$\equiv c \geq (\frac{1}{2})(\frac{1+2}{-1+2})$$

$$\equiv c \geq \frac{3}{2} > 1.$$

Other possible answers can be constructing some function that has different order of growth for different xs. For example let $f(n) = n^2$ for even numbers and $f(n) = n^3$ for odd numbers. We still let $b = 2$, and a can be either 1 or 2, as long as $\log_b a < 2$. For such a $f(n)$ and ns such that n is even number and $n/2$ is odd, $a(n/2)^3$ is greater than n^2 asymptotically, so no positive $c < 1$ can exist.