ECE GY 9343 Midterm Exam (2022 Spring)

Name: **NetID:**

Session(Circle one): Session A (Prof. Yong Liu) Session B (Prof. Pei Liu)

Answer ALL questions. Exam is closed book. No electronic aids. However, you are permitted two cheat sheets, two sides each sheet. Any content on the cheat sheet is permitted.

Multiple choice questions may have **multiple** correct answers. You will get **partial credits** if you only select a subset of correct answers, and will get zero point if you select one or more wrong answers.

Requirements for in-person students ONLY

Please answer all questions on the question book. You should have enough space. Since we scan all the submissions in a batch, DON'T write on the back of the pages and don't tear off any page.

Make sure you write your NetID on the bottom of each page (not your N number).

If you need extra scrape papers, we can give to you. However, it will not be graded.

Requirements for remote students ONLY

Each student is required to open Zoom and turn on their video camera, making sure the camera captures your hands and your computer screen/keyboard. The whole exam will be recorded. To clearly capture the video of exam taking, it is recommended that: A student can use an external webcam connected to the computer, or use another device (smartphone/tablet/laptop with power plugged in). Adjust the position of the camera so that it clearly captures the keyboard, screen and both hands. During the exam, you should keep your video on all the time. If there is anything wrong with your Zoom connection, please reconnect ASAP. If you cannot, please email ASAP, or call your proctor.

During the exam, if you need to use the restroom, please send a message using the chat function in Zoom, so we know you left.

Please use a separate page for each question. Clearly write the question numbers on top of the pages. When you submit, please order your pages by the question numbers.

Once exam is finished, please submit a single PDF on Gradescope, under the assignment named Midterm Exam. DEADLINE for submission is 15 minutes after the exam ended. Please remember to parse your uploaded PDF file. Before you leave the exam, its your responsibility to make sure all your answers are uploaded. If you have technical difficulty and cannot upload 10 minutes after exam ended, email a copy to your proctor and the professor.

1. (**20 points**) **True or False**

   (a) **T or F:** $3^n = \Theta(2^n)$;

   (b) **T or F:** When input size is very large, a divide-and-conquer algorithm is always faster than an iterative algorithm that solves the same problem;

   (c) **T or F:** It takes $\Theta(n)$ time to check if an array of length n is sorted or not;

   (d) **T or F:** Bubble-sort is stable;

   (e) **T or F:** Counting sort is NOT in-place;

   (f) **T or F:** An array with the values of $[15, 9, 3, 6, 5, 4, 1]$ is a max heap.

   (g) **T or F:** If any one element in a max-heap with n nodes is changed, the heap property can be restored in $O(n)$ time;

   (h) **T or F:** If we are using the division method for our hashing function, a table size of 128 would be a good choice.

   (i) **T or F:** If chaining is used to handle collisions, the first element inserted to the hash table is always at the head of the chain;

   (j) **T or F:** Complexity to find the min or max node of a binary search tree is a constant.

2. (**4 points**) If $f(n) = \Theta(g(n))$, and $g(n) = \omega(z(n))$, which of the following are true?

   (a) $f(n) = \Omega(g(n))$;

   (b) $f(n) = \Theta(z(n))$;

   (c) $f(n) = \omega(z(n))$;

   (d) $g(n) = O(z(n))$;

   (e) None of the above

3. (**4 points**) Which of the following sorting algorithms run in worst-case time $O(nlogn)$?

   (a) QuickSort;

   (b) HeapSort;

   (c) InsertionSort;

   (d) MergeSort;

   (e) None of the above

4. (**4 points**) Which of the following statements about hash tables are true?

   (a) Hash tables are preferred over direct-address tables due to possible reduction in storage requirement;

   (b) Search for the maximum node always takes $\Theta(n)$;

   (c) When handling collisions using chaining, search for a non-existing node always takes $\Theta(n)$;

   (d) With universal hash functions, a random hash function is picked each time whenever a new item is inserted into the table;

   (e) None of the above

5. (**4 points**) Given a max-heap with height 8 and distinct keys, the 3-rd largest element can be at height of:

   (a) 1;

   (b) 5;

   (c) 6;

   (d) 7;

   (e) 8;

6. (**4 points**) Which of the following about binary search tree is true?

(a) the maximum key is always on a leaf node;

(b) in-order tree walk prints out all keys in sorted order;

(c) for any key in the left subtree of a node z is less than or equal to any key in the right subtree of z;

(d) binary search tree is a close-to-complete binary tree;

(e) the worst-case search time in a binary search tree with n nodes is $\Theta(n)$;

7. (**6 points**) Illustrate the operation of insertion-sort on the array [71, 25, 40, 7, 60, 13, 20, 80]

8. (**6 points**) Given an array of $[3, 9, 5, 8, 15, 7, 4, 10, 6, 12, 16]$,

(a) (**3 points**) plot the initial max-heap built from the array;

(b) (**3 points**) plot the max-heap after the maximum is extracted, then a new number 11 is inserted.

7. (6 points) Illustrate the operation of insertion-sort on the array [71, 25, 40, 7, 60, 13, 20, 80]

71	25	40	7	60	13	20	80

25	71	40	7	60	13	20	80

25	40	71	7	60	13	20	80

7	25	40	71	60	13	20	80

7	25	40	60	71	13	20	80

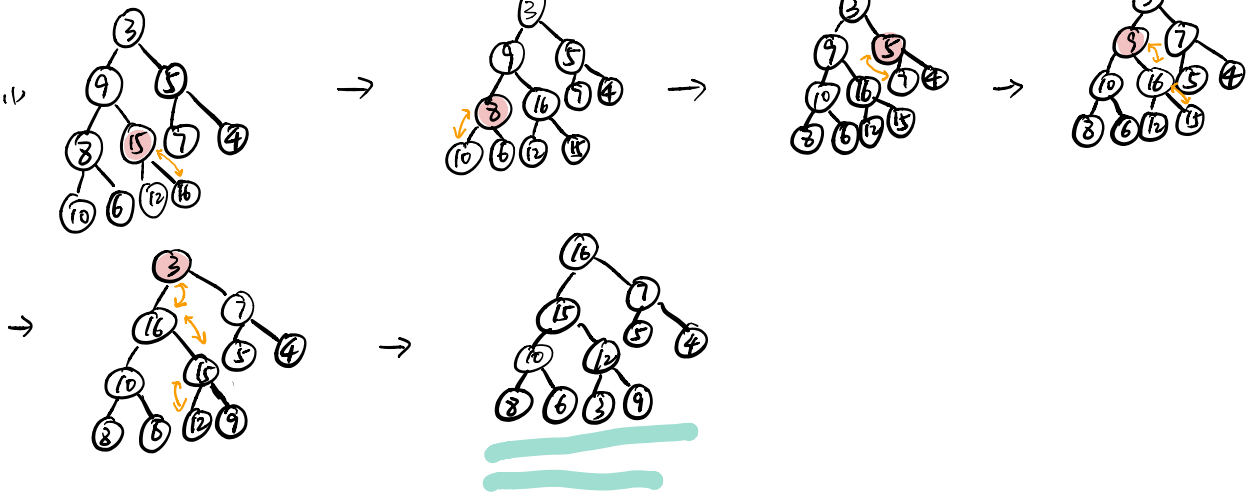
7	13	25	40	60	71	20	80

7	13	20	25	40	60	71	80

8.

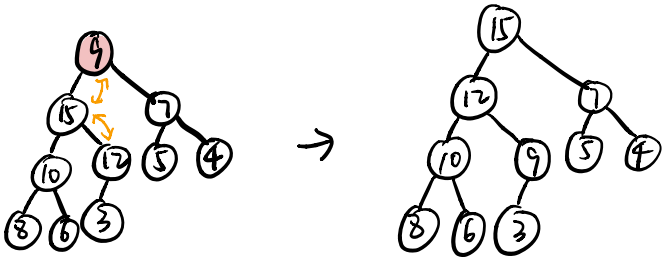
(a)

(b)

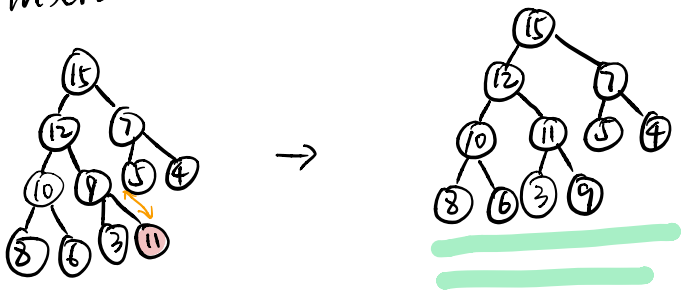


(b)

Max - extract



Insert. 11



9 (12 points)

9.1 (4 points)

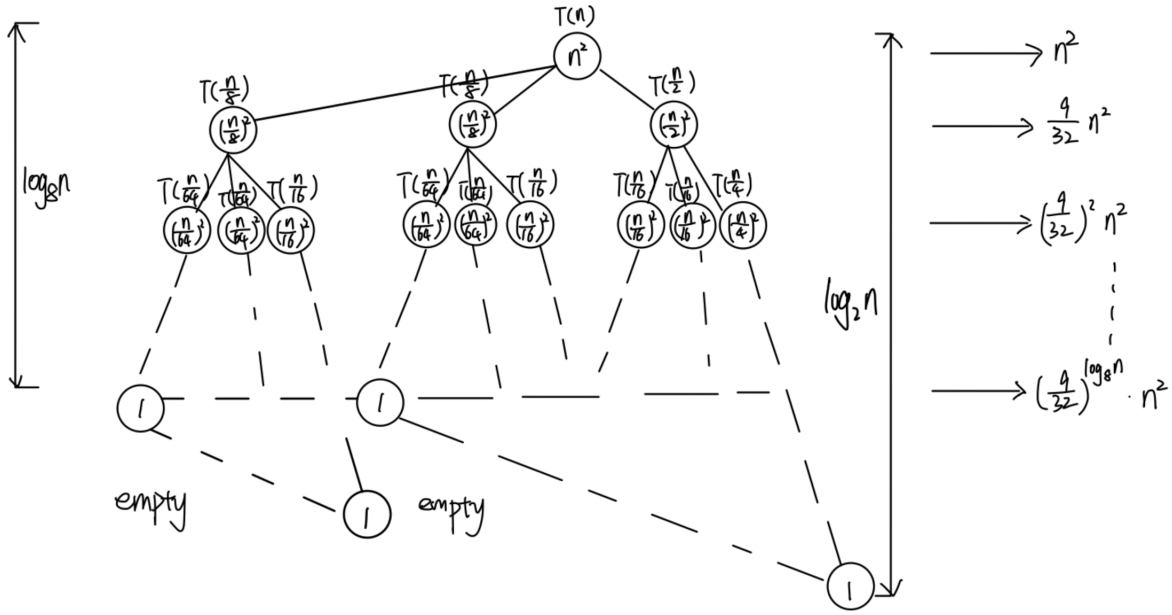


Figure 1: asymmetric and inbalanced recursion tree

The recursion tree is asymmetric and inbalanced: the recursions that divide the problem size by 8 reach the bottom the fastest at depth $log_8 n$, whereas the recursions that divide the problem size by 2 reach the bottom at depth $log_2 n$. Other recursions that keep dividing the problem size by combinations of 8 and 2 are in the middle. Note that the cost at each depth is reduced by a factor of $\frac{9}{32}$ before reaching the depth $log_8 n$. In other words, the merging cost at depth k is $n^2 * \frac{9}{32}^k$. Then we can bound $T(n)$ from above and below by

$$upper = (1 + \frac{9}{32} + \frac{9}{32}^2 + \frac{9}{32}^3 + \cdots + \frac{9}{32}^{log_2 n})n^2 \quad (1)$$

$$lower = (1 + \frac{9}{32} + \frac{9}{32}^2 + \frac{9}{32}^3 + \cdots + \frac{9}{32}^{log_8 n})n^2 \quad (2)$$

$$lower \leq T(n) \leq upper \quad (3)$$

As $n \to \infty$, both the upper and lower bounds tend to $\frac{32}{23}n^2$, which implies $T(n) = \Theta(n^2)$.

9.2 (4 points)

The upper bound:
IH: $T(k) \leq dk^2$ for all $k < n$, then consider T(n),

$$T(n) = T(\frac{n}{2}) + 2T(\frac{n}{8}) + n^2 \leq d * \frac{n}{2}^2 + 2d * \frac{n}{8}^2 + n^2 \tag{4}$$
$$\leq dn^2 \tag{5}$$

Then we can get

$$d \geq \frac{32}{23} \tag{6}$$

If we set $d \geq \frac{32}{23}$ then for n, $T(n) \leq dn^2 \Rightarrow T(n) = O(n^2)$

The lower bound:
IH: $T(k) \geq ck^2$ for all $k < n$, then consider T(n),

$$T(n) = T(\frac{n}{2}) + 2T(\frac{n}{8}) + n^2 \geq c * \frac{n}{2}^2 + 2c * \frac{n}{8}^2 + n^2 \tag{7}$$
$$\geq cn^2 \tag{8}$$

Then we can get

$$c \leq \frac{32}{23} \tag{9}$$

If we set $c \leq \frac{32}{23}$, then for n, $T(n) \geq cn^2 \Rightarrow T(n) = \Omega(n^2)$
Bound by the upper and the lower, we can get $T(n) = \Theta(n^2)$.

9.3 (4 points)

$$T(n) = 4T(\frac{n}{2}) + n^2 + n^{1.5} \tag{10}$$

Use master method: $a = 4, b = 2, f(n) = n^2 + n^{1.5}$
In case 2: $f(n) = \Theta(n^2)$ is applied. Then we can get

$$T(n) = \Theta(n^{log_b a} logn) \tag{11}$$
$$= \Theta(n^2 logn) \tag{12}$$

10. (6 points) **Prove** or **disprove** the following properties of asymptotic notations:

(a) (3 points) $n + \sqrt{n} = \omega(n)$, where $\omega(\cdot)$ stands for little-ω;

(b) (3 points) If $f(n) = \omega(g(n))$, then $g(n) = o(f(n))$, where $o(\cdot)$ stands for little-o.

(a)

Disprove

$if\ we\ take\ c = 2, we\ know\ that\ for\ all\ n > 0, n + \sqrt{n} < 2n$
$so\ there\ doesnt\ exist\ n_0 \geq 0, for\ any\ constant\ c > 0, such\ that$
$n + \sqrt{n} > c * n\ satisfied$

So disproved

(b)

Prove

$f(n) = \omega(g(n)) \Leftrightarrow then\ for\ any\ constant\ c_1 > 0,$
$there\ exists\ n_0 \geq 0, \quad such\ that\ f(n) > c_1 g(n), for\ all\ n \geq n_0$

$Therefor, for\ any\ c > 0, we\ can\ find\ c_1 = \frac{1}{c} > 0, there\ exists$
$n_0 \geq 0, \quad such\ that\ f(n) > \frac{1}{c} g(n),$
$then\ g(n) < cf(n), \quad for\ all\ n \geq n_0, satisfied$

So proved

11. (**10 points**) The following is an iterative algorithm for binary search for a `target` value in an array `nums` with `n` numbers.

(a) (**2 points**) analyze the worst-case complexity of the algorithm;

(b) (**8 points**) prove the correctness of the algorithm using Loop-Invariant by clearly stating the `Loop-Invariant` statement, and proving `initialization`, `maintenance`, and `termination` steps.

```
int binarySearch(int nums[], int n, int target)
{  int low = 0, high = n - 1;
   while (low <= high)
   {   int mid = (low + high)/2;
       if (target == nums[mid]) {
          return mid;
       }
       else if (target < nums[mid]) {
          high = mid - 1;
       }
       else {
          low = mid + 1;
       }
   }
   return -1;
}
```

Answer:

a). The worst-case performance happens when $nums[mid] \neq target$ in all iterations. Since the length of the active range $[low, high]$ reduces by half after each iteration, the maximum number of iterations is $\log_2(n)+1$. So the worst-case performance is $\Theta(\log n)$.

b). **Loop-Invariant Statement:** If `target` is in `nums[0···n-1]`, it must be in the range of `nums[low···high]`.
Initialization: When `low=0` and `high=n-1`, automatically true.

Maintenance: For any iteration, given the statement is true before the iteration, there are three possibilities,

(a) `target==nums[mid]`, there is no change in `low` and `high`, statement remains to be true.

(b) `target<nums[mid]`, since the array is sorted, if `target` is in the range of `nums[low···high]` before the iteration, then it must be in the range of `nums[low···mid-1]`, after setting `high=mid-1`, the statement remains to be true;

(c) `target>nums[mid]`, since the array is sorted, if `target` is in the range of `nums[low···high]` before the iteration, then it must be in the range of `nums[mid+1···high]`, after setting `low=mid+1`, the statement remains to be true;

Termination: The while loop terminates under two situations:

(a) `target==nums[mid]` in some iteration, the `target` is found, `mid` is its position, the returned value `mid` is correct;

(b) `low>high` after all iterations complete, the range of `nums[low···high]` is empty, based on the loop-invariant, `target` is not in `nums`, the returned value -1 is correct.

12. (**9 points**) In Randomized Quicksort (with Lomuto's partition) for an array of n distinct numbers.

(a) (**4 points**) after the first call of randomized-partition, what is the probability that the i-th ranked number is in the left partition, for any $1 \leq i \leq n$?

(b) (**5 points**) suppose the second partition call is for the right partition resulted from the first partition call, what is the probability that the i-th ranked number is in the same partition as the $(i+k)$-th ranked number after the first two partition calls?

Answer:

a) Let the rank of the randomly selected pivot be Y, the i-th ranked number will be in the left partition if and only if $Y \geq i+1$, the probability is

$$P\{i\text{-th ranked number in the left partition}\} = P\{Y \geq i+1\} = \frac{n-i}{n}$$

b) after the first partition call, the i-th ranked number and $(i+k)$-th ranked number will both be in the left partition if the first chosen pivot rank is larger than $i+k$; they will both end up in the right partition if the first pivot rank is less than i;

Since the second partition call only works on the right partition, so the probability that the two numbers will be in the same partition after two calls can be calculated as the sum of the probability that both in the left partition after the first call, and the probability that both in the right partition after the first call & stay in the same partition after the second call.

For the second case, the right partition length is $j \geq n-i+1$, the probability that i and $i+k$ will be in the same partition after the second call is that the second pivot is not chosen from $k+1$ numbers with rank i through rank $i+k$ (inclusive), which happens with probability of $\frac{j-k-1}{j}$. The the final probability is

$$P\{i \text{ and } i+k \text{ stay in the same partition after two calls}\} = \frac{n-i-k}{n} + \frac{1}{n}\sum_{j=n-i+1}^{n-1} \frac{j-k-1}{j}.$$

13. (**11 points**) We have a hash table with m slots and collisions are resolved by chaining. Suppose n distinct keys are inserted into the table. Each key is equally likely to be hashed to each slot.

(a) (**5 points**) Let X_i be the random variable of the position of the i-th ($1 \leq i \leq n$) inserted key in its chain after all keys have been inserted (counting the position from the head of the chain), what is the probability mass function of X_i, i.e., calculate $P(X_i = k)$, $\forall k \geq 0$?

(b) (**3 points**) What is the expected value of X_i?

(c) (**3 points**) What is the expected number of elements examined when searching for a randomly selected key?

Answer: a). X_i depends on how many keys inserted after the i-th key are hashed into the same slot as the i-th key, each of which happens with probability of $\frac{1}{m}$. Therefore, X_i follows the binomial distribution with $n-i$-trial and $p = \frac{1}{m}$, so its p.m.f is

$$P(X_i = k) = \binom{n-i}{k} \frac{1}{m^k} (1 - \frac{1}{m})^{n-i-k};$$

b). The expected value is $E[X_i] = \frac{n-i}{m}$

c). The average number of elements examined for a successful search is

$$1 + \frac{1}{n} \sum_{i=1}^{n} E[X_i] = 1 + \frac{1}{n} \sum_{i=1}^{n} \frac{n-i}{m} = 1 + \frac{n-1}{2m}$$