CS 344 **Problem Set 2**
Fall 2022 **Due:** October 24, 2022, 11:59 p.m.

Name: (Dong Shu) **NetID**: (ds1657)

Honor Code: Students may discuss and work on homework problems in groups, which is encouraged. However, each student must write down their solutions independently to show they understand the solution well enough to reconstruct it by themselves. Students should clearly mention the names of the other students who offered discussions. We check all submissions for plagiarism. We take the honor code seriously and expect students to do the same.

Instruction for Submission: This homework has a total of 100 points, it will be rescaled to 10 points as the eventual score.

We encourage you to use LaTex to write your answer, because it's particularly suitable to type equations and it's frequently used for writing academic papers. We have provided the homework2.tex file for you, you can write your answer to each question in this .tex file directly after the corresponding question, and then compile the .tex file into a PDF file for submission. As a quick introduction to LaTex, please see this *Learn LaTeX in 30 minutes* [1]. Compiling a .tex file into a PDF file needs installing the LaTex software on your laptop. It's free and open source. But if you don't want to install the software, you can just use this website `https://www.overleaf.com/`, which is also free of charge. You can just create an empty project in this website and upload the homework1.zip, and then the website will compile the PDF for you. You can also directly edit your answers on the website and instantly compile your file.

You can also use Microsoft Word or other software if you don't want to use LaTex. If so, please clearly number your answers so that we know which answer corresponds to which question. You also need to save your answers as a PDF file for submission.

Finally, please submit your PDF file only. You should name your PDF file as "Firstname-Lastname-NetID.pdf".

Late Policy: The homework is due on 10/24 (Monday) at 11:59pm. We will release the solutions of the homework on Canvas on 10/27 (Thursday) 11:59pm. If your homework is submitted to Canvas before 10/24 11:59pm, there will no late penalty. If you submit to Canvas after 10/24 11:59pm and before 10/27 11:59pm (i.e., before we release the solution), your score will be penalized by 0.9^k, where k is the number of days of late submission. For example, if you submitted on 10/27, and your original score is 80, then your final score will be $80 * 0.9^3 = 58.32$ for $27 - 24 = 3$ days of late submission. If you submit to Canvas after 10/27 11:59pm (i.e., after we release the solution), then you will earn no score for the homework.

Make best use of picture in Latex: If you think some part of your answer is too difficult to type using Latex, you may write that part on paper and scan it as a picture, and then insert that part into Latex as a picture, and finally Latex will compile the picture into the final PDF

[1] `https://www.overleaf.com/learn/latex/Learn_LaTeX_in_30_minutes`

output. This will make things easier. For instructions of how to insert pictures in Latex, you may refer to the Latex file of our homework 1, which includes several examples of inserting pictures in Latex.

Discussion Group (People with whom you discussed ideas used in your answers if any):

QiHang Wang, TianLe Chen

I acknowledge and accept the Honor Code. Please type your initials below:

Signed: (DS)

---

1. **Warm up with BST.** (10 points) Beginning with an empty binary search tree, what binary search tree is formed when you insert the following values in the order given? (Values are compared in alphabet order, e.g., $A < B$)

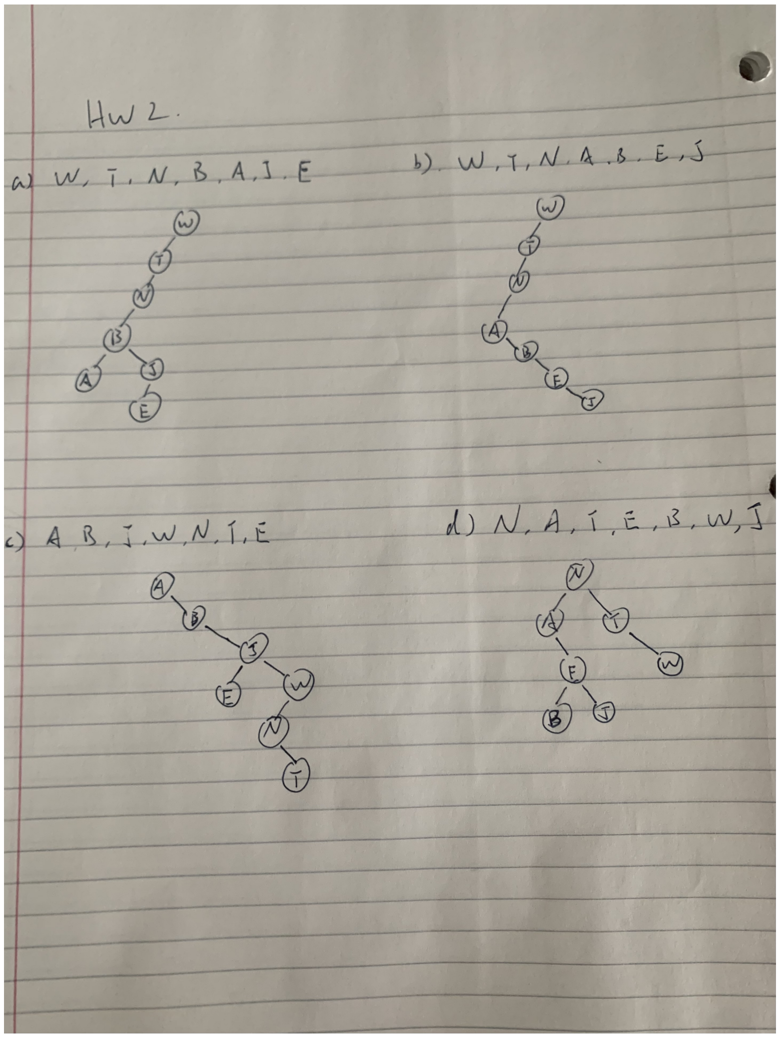
   (a) W, T, N, B, A, J, E

   (b) W, T, N, A, B, E, J

   (c) A, B, J, W, N, T, E

   (d) N, A, T, E, B, W, J

   Answer:

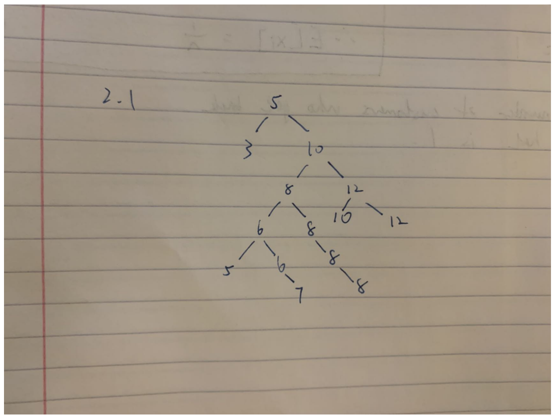


2. **Binary search tree with equal keys.** (40 points) In the class, we assumed that all the keys in a BST are different from each other. However, the BST can still store keys of the same value, in this case, we put a key that is less than a node to its left, and put a key that is greater than **or equal to** a node to its right. Here is the algorithm for inserting a new key z in to a binary search tree T:

   (a) (10 points) To better understand the algorithm, draw the tree generated by inserting the numbers 5, 3, 10, 8, 12, 10, 12, 8, 8, 6, 6, 7, 5, 8 in this given order into an initially empty binary search tree using the above algorithm. Answer:

Algorithm 1: Tree-Insert(T, z)

```
1  y = NIL;
2  x = T.root;
3  while x ≠ NIL
4      y=x;
5      if z.key<x.key
6          x = x.left;
7      else x = x.right
8  z.parent = y;
9  if y==NIL
10     T.root = z;    //tree T was empty
11 elseif z.key<y.key
12     y.left=z;
13 else y.right=z;
```



(b) (10 points) What is the asymptotic runtime of Tree-Insert when used to insert n items with identical keys into an initially empty binary search tree?

Answer:

In the worst case, the tree will lean all the way to the right or left. In this case, the depth of the tree is n. Every time when we insert a node, we need to compare with every node in the tree on line 5. In other world, when we insert the first node, the comparison is 0, because the tree is empty. When we insert the second node, the comparison is 1, because there is only one node in the tree. When we insert n nodes, we need to do n-1 comparisons, because there are n-1 nodes in the tree. Therefore the asymptotic runtime on line 5 will be $0+1+2+...+n-1 = (n(n-1))/2 = O(n^2)$. In the best case, the tree will be balanced or balanced enough. In this case, the depth of tree is $\log n$. Every time when we insert a new node, we don't need to compare with all of them. Therefore, the first level comparison will be $O(\log 1)$ on line 5. When we reach the second level, the comparison will be $O(\log 2)$. When we reach the $\log n$ level, the comparison will be $O(\log n)$. Therefore, the asymptotic runtime will be $O(\log 1 + \log 2 + ... + \log n) = O(\log 1 * 2 * ... * n) = O(\log n!)$.

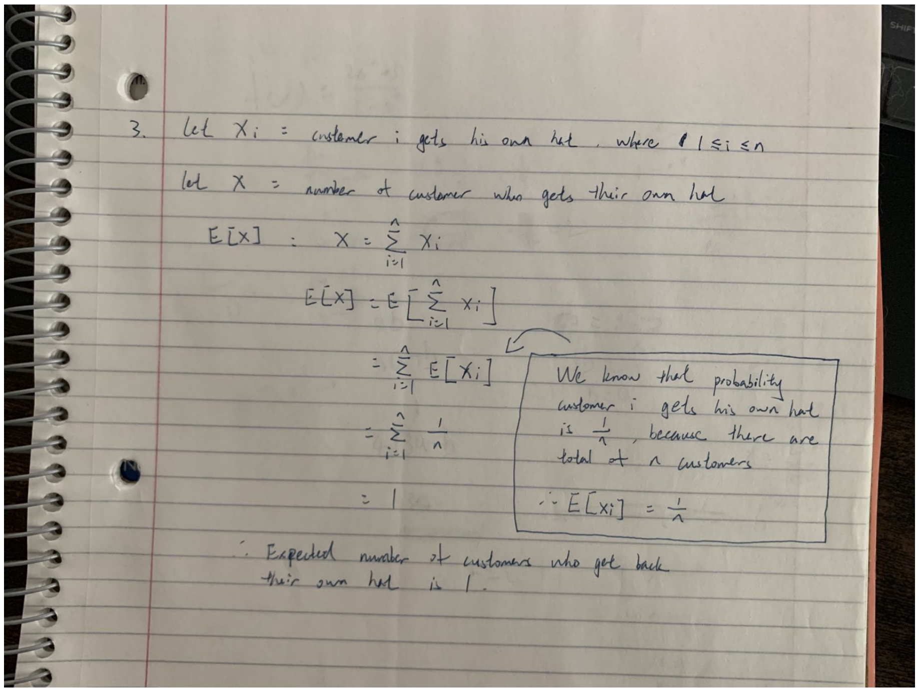
(c) We propose to improve Tree-Insert by testing before line 5 to determine whether z.key==x.key and by testing before line 11 to determine whether z.key==y.key. If equality holds, we implement one of the following strategies (c1 and c2). For each of the strategy (c1 and c2), find the asymptotic runtime of inserting n items with identical keys into an initially empty binary search tree. (The strategies are described for line 5, in which we compare the keys of z and x, and substitute y for x to arrive at the strategies for line 11.)

(c1) (10 points) Keep a list of nodes with equal keys at x, and insert z into the list.
answer:
When we add "if z.key == x.key", we need to substitute y for x and jump out from the while loop and goes to line 11 where we add "elseif z.key == y.key" and insert z to the list.
If we insert n items with identical keys into the tree. In this case, the depth of the tree is at most 1, because every node has same value, resulting in a running time of O(1) for each node comparison in "if z.key == x.key". Since we have n nodes, then we will have O(n). After the while loop, we also have O(1) for each node comparison in "elseif z.key == y.key", and we have n nodes, so O(n). After that the insertion to a list also can be done in constant operation. Therefore, the asymptotic running time will be O(n+n+c) = O(n).

(c2) (10 bonus) Randomly set x to either x.left or x.right. (Give the worst-case runtime and informally derive the expected runtime.)
answer:
For the worse case, if x been randomly set to the node that has no NIL children every time, then the new key needs to compare with all the node in the tree before it jumps out from the while loop. In this case, the running time equals the worst case running time in question b. Therefore, the running time will be $0+1+2+...+n-1 = (n(n-1))/2 = O(n^2)$.
For the expected running time, if x been randomly set to the node that can help balance the tree, then we can say the keys we need to compare are at most $\log n$. Therefore, we have running time equals to the best case in question b, which is $O(\log 1 + \log 2 + ... + \log n) = O(\log 1 * 2 * ... * n) = O(\log n!)$.
[**Note: for (b) (c1) and (c2), we are expecting the runtime represented as a function of** n].

3. **Hat-check problem.** (10 points) Use indicator random variables to solve the following problem, which is known as the hat-check problem. Each of n customers gives a hat to a hat-check person at a restaurant. The hat-check person gives the hats back to the customers in a random order. What is the expected number of customers who get back their own hat?

Answer:



3. let X_i = customer i gets his own hat, where $1 \leq i \leq n$

let X = number of customer who gets their own hat

$E[X]$: $X = \sum_{i=1}^{n} X_i$

$$E[X] = E\left[\sum_{i=1}^{n} X_i\right]$$

$$= \sum_{i=1}^{n} E[X_i]$$

$$= \sum_{i=1}^{n} \frac{1}{n}$$

$$= 1$$

$$\boxed{\begin{array}{l}\text{We know that probability customer } i \text{ gets his own hat}\\ \text{is } \frac{1}{n}\text{, because there are total of } n \text{ customers}\\ \therefore E[X_i] = \frac{1}{n}\end{array}}$$

$\therefore$ Expected number of customers who get back their own hat is 1.

4. **Quicksort with equal element values.** (30 points) The analysis of the expected running time of randomized Quicksort in class (corresponding to Section 7.4.2 in textbook) assumes that all element values are distinct. In this problem, we examine what happens when they are not, i.e., there exist same-valued elements in the array to be sorted.

   The Quicksort algorithm relies on the following partition algorithm, which finds a pivot randomly, and then put all the numbers less than or equal to the pivot in the left, and put all the numbers greater than the pivot in the right, and then return the pivot location, as well as the left and right sublists for recursive calls.

Algorithm 2: Partition(A, p, r)

```
1 q=RANDOM(p,r);  //generate a random number in the range of [p,r]
2 L=empty list, R=empty list;
3 for each element a in A except A[q]:
4     if a≤A[q]:
5         append a to L;
6     else append a to R;
7 A=append(L,A[q],R);
8 returen A, q;
```

   In this algorithm, the list to be sorted is A, and we use p and r to denote the left-most and right-most indices of the currently processing subarray, respectively. For example, in the initial call of the Quicksort algorithm, we will let $p = 0$ and $r = n-1$, which correspond to the whole original array. The PARTITION(A, p, r) procedure returns an index q such that each element of $A[p : q-1]$ is less than or equal to $A[q]$ and each element of $A[q+1 : r]$ is greater than $A[q]$.

(a) (10 points) Suppose there are n elements and all element values are equal. What would be randomized Quicksort's running time in this case? answer:
$O(n^2)$, because in this algorithm we are moving the equal value to the left of the pivot. In this case, the algorithm will compare every elements. So we need $O(n^2)$ operations.

(b) (10 points) Modify the PARTITION procedure to produce a procedure PARTITION$'(A, p, r)$, which permutes the elements of $A[p : r]$ and returns two indices q and t, where $p \leq q \leq t \leq r$, such that:

i. all elements of $A[q : t]$ are equal,

ii. each element of $A[p : q - 1]$ is less than $A[q]$, and

iii. each element of $A[t + 1 : r]$ is greater than $A[q]$

Like PARTITION, your PARTITON$'$ procedure should take $O(r - p)$ time.

answer:

Algorithm 3: PARTITION'(A, p, r)

```
q = RANDOM(p,r)
Left = []
Right = []
Equal = []
for each element $i$ in $A$ do
    if $A[i] < A[q]$ then
        Left.append(A[i])
    else if $A[i] > A[q]$ then
        Right.append(A[i])
    else Equal.append(A[i])
A = [Left, Equal, Right]
q_position = Left.length() - 1
q = A[q_position]
t_position = q_position + Equal.length() - 1
t = A[t_position]
return $q, t$
```

(c) (10 points) Now you have a PARTITION$'(A, p, r)$ algorithm, based on this algorithm, provide the pseudocode of a QUICKSORT$'(A)$ algorithm which calls PARTITION$'(A, p, r)$ as a subroutine, so that it recurses only on partitions of elements not known to be equal to each other. [Hint: it will looks very similar to our Quicksort pseudocode in the lecture slides, you only need to make minor modifications to recurse on the partitions produced by PARTITION$'(A, p, r)$.]

answer:

Algorithm 4: QUICKSORT'(A)

```
if $length(A) \leq 1$ then
    return
$p$ = RAMDOM(A)
$r$ = A.length
A[q], A[t] = PARTITION'(A, p, r)
L = all elements from p to q
R = all elements from t to r
QUICKSORT'(L)
QUICKSORT'(R)
```

5. **Hashtables.** (10 points) A frequently used hash function to hash integer numbers is:

$$h_{a,b}(x) = ((ax + b) \mod p) \mod n \tag{1}$$

which we have introduced in our class. It works in the following way:

To hash an integer x in $\{0, 1, \cdots, |U|-1\}$ to a bucket in $\{0, 1, \cdots, n-1\}$, we first find the smallest prime $p \geq |U|$. Then, let a be a random number in $\{1, 2, \cdots, p-1\}$, and let b be a random number in $\{0, 1, \cdots, p-1\}$, then x will be hashed to bucket $h_{a,b}(x) = ((ax + b) \mod p) \mod n$.

Prove that the above is a Universal Hash Family.

answer:

We know that in order to be a universal hash family, the probability of a collision have to upper bounded by 1/n.
The function for the probability of a collision is $P(h(x) - h(y) = 0)$. And we know that $h(x) = ((ax + b) \mod p) \mod n$ and $h(y) = ((ay + b) \mod p) \mod n$. So $h(x) - h(y) = ((a(x - y) + b) \mod p) \mod n$. We know that x and y are two arbitrary numbers, so $a(x - y) + b$ is also a arbitrary number. After we module this arbitrary number with p, the result will be an arbitrary number between 0 to p-1. So if we module this arbitrary number between 0 to p-1 with n, the result will also be an arbitrary number between 0 to n-1. In this case, we will have n choices from 0 to n-1. Therefore our probability that the result equals to 0 is 1/n. In this case, we know that our probability of collision is 1/n. So we proved $h_{a,b}(x) = ((ax + b) \mod p) \mod n$ is a universal hash family.