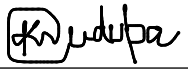
MATH 8650 Advanced Data Structures Fall 2018
Midterm Exam

October 19–21, 2018

- This take-home exam is due by 8:00am, October 21, 2018.Sign, scan, and return this cover sheet with your exam. Scan handwritten responses or type responses in Word or LaTeX, and turn in a PDF file with the answers neatly written and in order. For the programming section, turn in a .py file.
- There are a total of 80 points, 10 for each conceptual problem and 20 for the programming part.
- You may consult the published course notes, your own class notes, or the instructor, but no other sources.
- Mark your answers clearly. *Show your work.* Unsupported correct answers receive partial credit.
- Start each numbered question on a new sheet. Be sure to include your name on *each* page.
- Good luck!

Name: VIVEK KOODLI UDUPA

I certify that I have not received any unauthorized assistance in completing this examination.

Signature: 

Date: Oct 21, 2018

Programming

Implement a Python class `HashTable` with the following specifications:

- For a `HashTable` object `H`,
  - `H = HashTable(n)` creates a hash table with `n` buckets as a Python list with `n` empty entries.
  - `H.insert(k,v)` inserts an object with key `k` and value `v`.
  - `H.delete(k)` removes the object with key `k` if it exists or throws a `KeyError` if not.
  - `H.find(k)` returns the value of the object with key `k` if it exists or throws a `KeyError` if not.
  - `H.list()` lists the key-value pairs in arbitrary order.
- Keys are 10-digit phone numbers and the hash function simply treats the number as an integer and computes its remainder mod `n`.
- There is no collision resolution. If `insert()` produces a hash index that is already occupied, replace the key and value of the existing entry with the new key and value.
- The `list()` method should run in time $O(p)$, where p is the number of live entries in your table. So you need an auxiliary data structure to keep track of which locations contain live entries. Insertions and deletions should take constant time, so your auxiliary data structure should be updatable in constant time. You can use an array to hold copies of keys and indexes into the hash table, and store the index into that array together with the hash-table entry. Append new entries to the end of your array. Delete entries by copying the last entry over the deleted one and popping the last entry.

Conceptual Problems

1. Consider a singly linked list of n nodes, numbered 0, 1, 2, ..., $n-1$ starting at the head. Write a recursive function to delete all the even-numbered nodes.
2. Let $f(n) = n^{3/2} \log n$ and $g(n) = 3n^2$. Prove rigorously using the definition that $f(n)$ is $O(g(n))$ but $f(n)$ is not $\Theta(g(n))$
3. Consider a binary heap with 18 elements stored in an array. Suppose all elements have distinct priorities. We know the element with the smallest key appears in position 0 of the array.
   (a) In what locations could the element with the second smallest key appear?
   (b) In what locations could the element with the largest key appear?

4. Construct an algorithm to reverse the contents of a stack S of n integers stored in an array, so that the top element in the modified stack is the bottom element of the original stack. You may use an auxiliary stack T of the same size and one auxiliary integer variable v and you must return the original stack S with its elements reversed, i.e., you may not return T instead. You may write pseudocode or python. What is the complexity of your algorithm?

5. You are given an array A of nonnegative integers and you need to determine whether there are two indices i and j such that $A[i] = 3 \times A[j]$. For example, if $A = (3, 24, 10, 19, 72)$ the your algorithm should print *yes* because $72 = 3 \times 24$, but if $A = 6, 25, 72, 19, 20)$, your algorithm should print *no*.

   (a) Describe an algorithm (in pseudocode or 2–3 English sentences) for this problem with worst-case running time $O(n^2)$.

   (b) Describe another algorithm with worst-case running time better than $O(n^2)$. Analyze the complexity of your algorithm.

6. The following is the preorder traversal of a binary expression tree.

$$+ \times 16 - 2\ 3\ 9$$

   Draw the expression tree and compute the value of the expression.

NAME: VIVEK KOODLI UDUPA

CONCEPTUAL PROBLEM

(1) Consider a singly linked list of n nodes, 0,1,.. n-1. Write a recursive function to delete all the even-nodes.

```
class Node :
    def __init__(self, data = None, next = None) :
        self.data = data
        self.next = next
    def getNextNode(self) :
        return self.next
    def setNextNode(self, newNext) :
        self.next = newNext
    def getData(self) :
        return self.data

class LinkedList :
    def __init__(self) :
        self.head = None
    def push(self, data) :
        newNode = Node(data)
        newNode.setNextNode(self.head)
        self.head = newNode
```

NAME: VIVEK KOODLI UDUPA

```
def printLis(self):
    current = self.head
    while current:
        print(current.data, "  ", end="  ")
        current = current.getNextNode()
    print()
```

#Recursive Function ↴

```
def deleteEven(self, prev=None, current=None):
    if (prev==None and current==None):
        prev = None
        current = self.head
    if (current==None):
        return
    elif (current.data %. 2 ==0):
        if (prev==None):
            self.head = current.getNextNode()
            prev = self.head
        else:
            prev.setNextNode(current.getNextNode())
    if (current.getNextNode()==None):
        current = None
    else
        current = current.getNextNode()
    else: prev = current
          current = current.getNextNode();
```

Recursive Function Call → `Self.deleteEven(prev, current)`

NAME: VIVEK KOODLI UDUPA

Algorithm to remove even Numbers from Linked List

Step 1 : Define two variables "previous" and "current" to keep track of Nodes. Initialize previous to None and current to Head of the Linked List.

Step 2 : Check if we are at the end of the List.
IF NOT, Continue

Step 3 : Check if the current Node is even by using the Modulus 2. Even numbers return Zero.

Step 4 : If the number is even, then we delete the Node by NOT POINTING AT IT.
Change the previous pointer to point at the Next item.

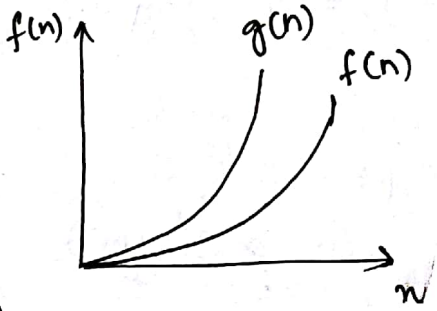
Step 5 : If the Number was odd in step 3, continue to the next node. without skipping over the node.

Step 6 : Recursively perform steps 2 through 5 until the end of Node is reached.

Step 7 : Once all the Nodes are checked and even nodes are deleted, print the final result.

② Let $f(n) = n^{3/2}\log(n)$ and $g(n) = 3n^2$. Prove that $f(n)$ is $O(g(n))$ but $f(n)$ is not $\Theta(g(n))$

Part I) $f(n)$ is $O(g(n))$ is true if $f(n) \leq c\,g(n)$ for $c > 0$ and $n \geq n_0$.



Basically this means that $g(n)$ is the upper-bound of $f(n)$

given $f(n) = n^{3/2}\log(n)$ $\quad g(n) = 3n^2$

$= n^{1.5}\log(n)$ $\quad g(n) = 3.(n^{1.5})(n^{0.5})$

$f(n) \leq c.g(n)$

$n^{1.5}\log(n) \leq 3c.(n^{1.5})(n^{0.5})$

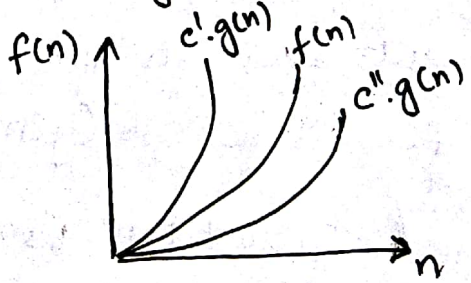
$\log(n) \leq 3c\,n^{0.5}$ — ①

Equation ① is true for all $c > 0$ and $n \geq 1$ $\therefore$ $f(n)$ is $O(g(n))$

part II) $f(n)$ is not $\Theta(g(n))$

$f(n)$ is $\Theta(g(n))$ if $c'.g(n) \leq f(n) \leq c''.g(n)$

If we can show that $f(n) < c''.g(n)$ then $f(n)$ will not be $\Theta(g(n))$



Since $f(n)$ is bounded by $c'.g(n)$ and $c''.g(n)$, and we proved in Equation ① that $c'.g(n) \geq f(n)$ we must show that ~~$f(n) \leq$~~ $f(n) \not\geq c''.g(n)$

$f(n) = n^{1.5}\log(n)$ $\quad$ $g(n) = 3n^2$

$c''.g(n) = c''.3n^2$

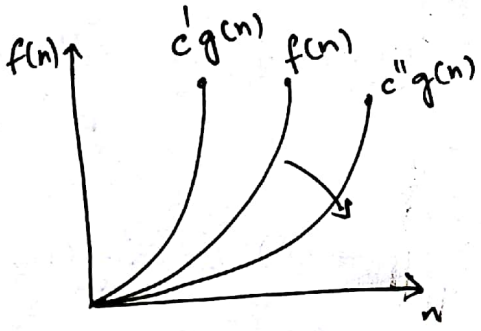
To prove:

$f(n) \not\geq c''.g(n)$ [$f(n)$ IS NOT GREATER THAN $c''.g(n)$]

$n^{1.5}\log(n) \not\geq 3c''.(n^{1.5})(n^{0.5})$

$\log_2(n) \not\geq 3c''.\sqrt{n}$ ~~—②~~

$\log_2(n) < 3c''.\sqrt{n}$ — ②



If we substitute $n = 4$ and $c'' = 1$ in ②

$\log_2(4) < 3.(1).\sqrt{4}$

$2 < 6$

we can see that $f(4) = 2$ and $c''g(4) = 6$. ~~Thus~~ ~~w~~ $f(n)$ went below its lower bound of $c''.g(n)$. Thus we can say $f(n)$ is not $\Theta(g(n))$

Thus we proved that $f(n)$ is $O(g(n))$ and $f(n)$ is NOT $\Theta(g(n))$

(3.) Consider a binary heap with 18 elements. we know that the element with the smallest key appears in position 0. of the array.

(a) In what location could the element with the second smallest key appear?

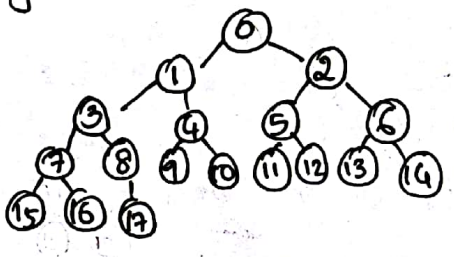
In a Minimum binary heap, the smallest element is always the root. The second smallest element can be either of the children of the root node. Thus in the array, second smallest number appears either in position 1 or position 2.

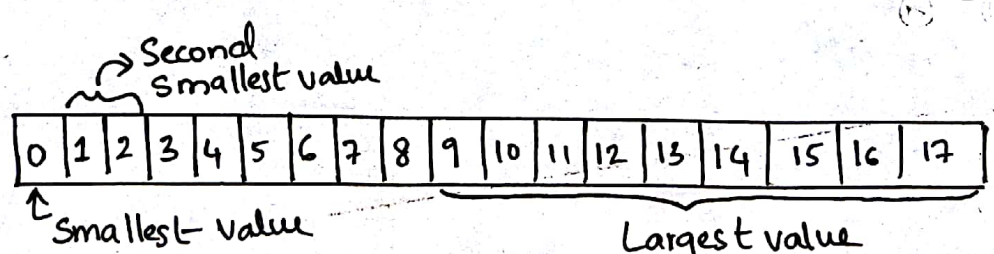
(b) In a Minimum binary heap, the largest element will be at any one of the leafs of the binary tree.

∴ for a tree of size n, largest element will be at any of the last n/2 position.

∴ For an array of 18 elements, Largest will be in one of the following position.

[9, 10, 11, 12, 13, 14, 15, 16, 17]





Note: Index starts at Zero.
∴ 18 elements = 0 to 17

Name: Vivek Koodli Udupa

④ Construct an algorithm to reverse the contents of a Stack S of n integers. You may use an auxilary stack and one auxilary variable.

To reverse the contents of a stack of n integers, we will ~~fo~~ use the following algorithm.

Step 1 : Take a stack S with n integers, Set count to zero.

Step 2 : pop the last entry of the stack ~~in~~ and store that value in the auxilary variable V.

Note: Stack follows First In Last Out order.

Step 3 : pop rest of the elements of the stack into the auxilary stack T.

Step 4 : Now Stack T has (n-1) values in reverse order. V has one value. Now set another variable to keep count of the (~~pop~~) pop done on stack T. When the pop count = Main count, pop the element from V. All these pop's are pushed into main stack S.

Step 5 : Incriment the main count.

Step 6 : Follow Step 2 through Step 5 ~~8~~ till the ~~St~~ reversed Stack is Stored in Stack S. This takes (n-1) iterations

Python implementation:

```
class Stack :
    def __init__(self):
        self.items = []

    def push(self, data):
        self.items.insert(0,data)

    def pop(self):
        return self.items.pop(0)

    def show(self):
        n = self.size()
        for i in range(n):
            print(self.items[i], "  ", end=" ")
        print()

    def size(self):
        return len(self.items)

#Functions
def reverse(r):
    #Initialize a Temporary stack
    T= Stack()

    n = r.size()
    count = 0
    while (count < n-1):
        r,T,v = switch1(n,r,T)
        r,T = switch2(n, count, r, T, v)
```

PTO

```
        count = count + 1
    return 0

def switch1(n, orig, T)
def switch1(n, S, T):
    for sc in range(n):
        if (sc == 0):
            V = S.pop()
        else:
            T.push(S.pop())
    return S,T,V

def switch2(n, count, S, T, V):
    for sc in range(n):
        if (sc == count):
            S.push(V)
        else:
            S.push(T.pop())
    return S,T
```

In the above code, "count" is used to count the total iterations.
S → original stack T → auxilary stack. V → auxilary int variable
Function switch1 puts elements from S into T and V
Function switch2 puts elements back into S.
we will consider a small example for better understanding the algorithm.

P.T.O

Name : Vivek Koodli Udupa

Example: We will consider a simple array of 5 elements [0, 1, 2, 3, 4]. We will use the algorithm discussed to reverse the above mentioned array.

(~~Remember~~ ~~S~~) Note: Stack follows first In Last out so popping elements from S and pushing them into T will look like:

S = [0, 1, 2, 3, 4] After the S = [ ]
T = [ ] operation T = [4, 3, 2, 1, 0]

Count	original Stack (S)	Auxilary (V)	Temp. Stack (T)
0	0 1 2 3 4 [1] 2 3 4 0 ←	→ 0	4 3 2 1
1	[2] 3 4 1 0 ←	→ 1	~~0~~ ~~4~~ 3 2
2	[3] 4 2 1 0 ←	→ 2	0 1 4 3
3	4 3 2 1 0 ⇓	→ 3	0 1 2 4
	[4 3 2 1 0]	Successfully Reversed!!	

As we can see above, in 4 (count = 3) iterations, we reversed the original stack. We need to pop all n elements from S to T and then back from T to S. And we must do this (n-1) times

∴ Complexity = ~~O(n)~~ ~~O(n)~~ $O(n * n * (n-1))$ = ~~$O(n^3)$ $O(n^2)$~~

$$\boxed{\simeq O(n^3)}$$

⑤ You are given an array A of non-negative integers and you need to determine whether there are two indices i & j such that A[i] = 3 x A[j].

a) Describe an algorithm with worst case running time $O(n^2)$

The most basic approach to the given problem would be to have two loops and iterate through every element and check for the given condition.

This method takes $O(n^2)$ as there are 2 loops iterating through all n- elements.

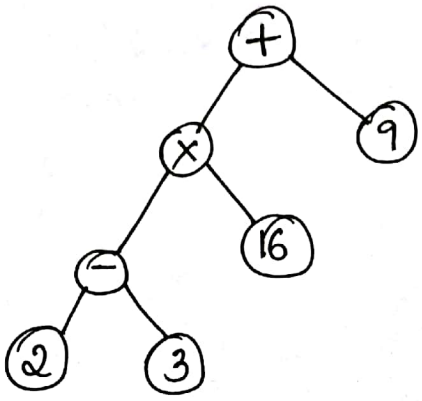
b) Describe another algorithm with worst-case running time better than $O(n^2)$

Given that the ~~(arrg)~~ array contains only non-negative integers, we can sort the array. Sorting ~~takes~~ has a complexity of ~~O(logn)~~ $O(n \log n)$ [Merge Sort / quick Sort].

With a sorted array, we can skip all the indices prior to the index in consideration and check for the remaining values. This way the complexity is definitely below $O(n^2)$ and on or above $O(n \log n)$.

(6) The following is the pre-order traversal of a binary expression tree.

+ X 16 − 2 3 9



((2 - 3) x 16) + 9

(-1 x 16) + 9

-16 + 9

= -7