18

Name: Covers chapter 1 – 8, 11 – 13, 22 – 23	CS3112 Fall 2017 Final Exam California State University, Los Angeles Instructor: Jungsoo Lim

I pledge by honor that I will not discuss the contents of this exam with anyone.
Signed by jdfnafjksdfkdfjdjnkjfdngjk Date 12/6/17

Part I. (20 pts)

✓ 1. (5 pts) Suppose you have an algorithm that operates on a set of data with n elements. If the recurrence formula that computes the time requirement for the algorithm is given by

$c = \theta(n^c \log n)$ or $n^{\log_b a} \log n$

$c > = n^c$

$c < n^{\log_b a}$

$$T(n) = \begin{cases} 2T\left(\frac{n}{11}\right) + D\sqrt[4]{n} & if\ n > 1 \\ C & if\ n = 1 \end{cases}$$

where D and C are constants, what is the order of complexity Θ of the algorithm?

$a = 2 \quad b = 11 \quad c = n^{1/4}$

$\frac{1}{4} \ ? \ \log_{11} 2$

$0.25 < 0.28$

$c < \log_b a$

$$\boxed{\Theta(n^{\log_{11} 2})}$$

5 2. (5 pts) Suppose you have an algorithm that operates on a set of data with n elements. If the recurrence formula that computes the time requirement for the algorithm is given by

$$T(n) = \begin{cases} T\left(\frac{n}{2}\right) + T\left(\frac{n}{16}\right) + T\left(\frac{n}{32}\right) + Dn & if\ n > 1 \\ C & if\ n = 1 \end{cases}$$

where D and C are constants, what is the order of complexity Θ of the algorithm?

$T\left(\frac{n}{2}\right) + T\left(\frac{n}{16}\right) + T\left(\frac{n}{32}\right) + Dn$

$T\left(\frac{n}{4}\right) + T\left(\frac{n}{32}\right) + \left(\frac{n}{64}\right) + Dn$

$\vdots$

$$\boxed{\Theta(n \log n)}$$

5

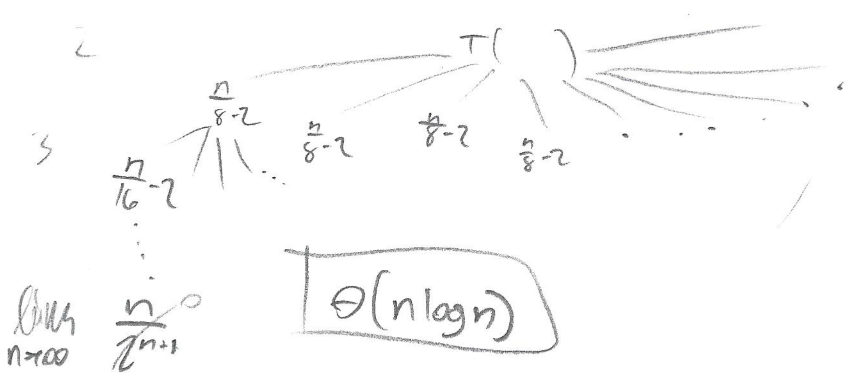
3. (5 pts) Suppose you have an algorithm that operates on a set of data with n elements. If the recurrence formula that computes the time requirement for the algorithm is given by

K:1

$$T(n) = \begin{cases} 9T\left(\frac{n}{8} - 2\right) + Dn\log n & if\ n > 1 \\ C & if\ n = 1 \end{cases}$$

where D and C are constants, what is the order of complexity Θ of the algorithm?

$9T(\frac{n}{8} - 2) + Dn\log n$



$\boxed{\Theta(n\log n)}$

5

4. (5 pts) Suppose you have an algorithm that operates on a set of data with n elements. If the recurrence formula that computes the time requirement for the algorithm is given by

$$T(n) = \begin{cases} T(n-4) + Dn & if\ n > 1 \\ C & if\ n = 1 \end{cases}$$

where D and C are constants, what is the order of complexity Θ of the algorithm?

$T(n-4) + Dn$

$n-4$ Dn

↓

$n-8$ Dn

↓

$n-16$ Dn

⋮ ⋮

$\frac{n}{4}$ n

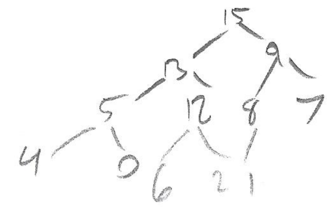
$\boxed{\Theta(\frac{n^2}{4})}$

Part II (30)

```
Parent(i)
        return floor(i/2)

Heap-Increase-Key (A, i, key)
if key < A[i]
        error "new key is smaller than current key"
A[i] = key
While i > 1 and A[Parent(i)] < A[i]
        Exchange A[i] with A[Parent(i)]
        i = Parent(i)

Max-Heap-Insert(A, key)
    A.heap-size = A.heap-size + 1
    A[A.heap-size] = – ∞
    Heap-Increase-Key(A, A.heap-size, key)
```



★Insert 10

✓ 1. (5) Run the operation of Max-Heap-Insert(A, 10) on the heap A= {15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1} $\boxed{-\infty}$

Max - Heap-Insert (A, 10)

A.heapSize = 13

A[13] = -∞

Heap-Increase-key(A, 13, 10)

Heap-Increase-key(A, 13, 10)

if key < A[13]

error

A[13] = key //8 < 10

while 13>1 and A[6] < A[13] ①

exch. A[13] w/ A[6] //{15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1, 10}

i = 6

{15, 13, 9, 5, 12, 10, 7, 4, 0, 6, 2, 1, 8}

② {15, 13, 9, 5, 12, 10, 7, 4, 0, 6, 2, 1, 8}

parent(6) returns 6/2 = 3

While 6>1 & A[3] < A[6]

//9 < 10 ✓

exch. A[6] w/ A[3]

i = 3

while 3>1 & A[1] < A[3]

// 15 < 10 ✗

//While loop breaks

③ {15, 13, 10, 5, 12, 9, 7, 4, 0, 6, 2, 1, 8}

{15, 13, 10, 5, 12, 9, 7, 4, 0, 6, 2, 1, 8}

4 2. (5) Why should we set $A[A.heap\text{-}size]$ to be – ∞ (negative infinity) in Max-Heap-Insert?

we do not want to leave an index with no value. Since the heapsize increases by one because of A.heapsize = A.heapsize + 1, we must set the added index assigned to a value.

-10

3. (10) Write a pseudocode for the procedure Heap-Decrease-Key(A, i, key).

Heap-Decrease-Key(A, i, key)

-10

4. (10) Write a pseudocode for the procedure Min-Heap-Insert(A, key).

Min-Heap-Insert(A, Key)

- set min as root

Part III (20)

Hire-Assistant(n)

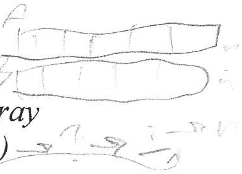
```
    best = 0          // candidate 0 is a least-qualified dummy candidate
    for i = 1 to n
        interview candidate i
        if candidate i is better than candidate best
            best = i
            hire candidate i          //fire the previous best
```

-10 ~~1~~. (10) In Hire-Assistant (5), if five (5) candidates are presented in a random order, what is the probability that you hire exactly three times?

$\boxed{\frac{3}{n!}}$

-10 ~~2~~. (10) Suppose that instead of swapping element A[j] with a random element from the subarray A[i…n], we implement the Permute-By-Cycle algorithm to generate uniform random permutation.

```
PERMUTE-BY-CYCLE(A)
    n = A.length
    let B[1...n] be a new array
    offset = RANDOM(1, n )
    for i = 1 to n
        dest = i + offset
        if dest > n
            dest = dest – n
        B[dest] = A[i]
    return B
```



Does this algorithm produce a uniform random permutation?
If it does, explain why. If not, explain why not.

Yes, because there is only one random number (offset) being used.

Part IV (30)

Suppose we want to find all prime numbers less than or equal to n, we can use the following efficient algorithm instead of brute force algorithm:

Find-All-Primes(n)

Let A be an array of Boolean values, indexed by integers 2 to n, initially all set to true.

// initialized each index to be true

```
for p = 2 to $\sqrt{n}$    → 4
if A[p] is true
    j = p2
    while j < n
        A[j] = false
        j =j + p
Output: all p such that A[p] is true.
```

Overview of Find-All-Primes(n)

1. Create an array of Boolean indexed consecutive integers from 2 through *n*: (2, 3, 4, ..., *n*) and initialize them to be true.
2. Initially, let *p* equal 2, the smallest prime number.
3. Enumerate the multiples of *p* by counting to *n* from 2*p* in increments of *p*, and mark them in the array as false (these will be 2*p*, 3*p*, 4*p*, ...; the *p* itself should not be marked).
4. Find the first number greater than *p* in the array that is not marked as false. If there was no such number, stop. Otherwise, let *p* now equal this new number (which is the next prime), and repeat from Step 3.
5. When the algorithm terminates, the numbers remaining not marked to be false in the array are all the primes below *n*.

-2

1. (10) Run the **Find-All-Primes (20)** and output the array A which contains the results.

2[3, 5, 7, 11, 13, 17, 19]

-10

2. (10) What is the running time of **Find-All-Primes(n)** algorithm?
(Hint: It has been proven that there are approximately $^{i}/_{\log i}$ primes for i.)

$o(n^2)$

-10

3. (10) The algorithm requires an array of size n. As n grows, the size of array grows as well. Thus, the algorithm is not scalable. How can we mitigate this issue?

setting a max input value can mitigate this issue.

Extra Credit

1. (10 pts) Design an algorithm how to multiply two complex numbers a + bi and c + di using only three multiplications of real numbers. Your algorithm should take a, b, c, and d as input and produce the real component ac – bd and the imaginary component ad + bc separately.

MULTIPLY-COMPLEX-NUMBER(a, b, c, d)

$(a+bi)(c+di)$

Foil = $ac + adi + cdi + dbi^2$

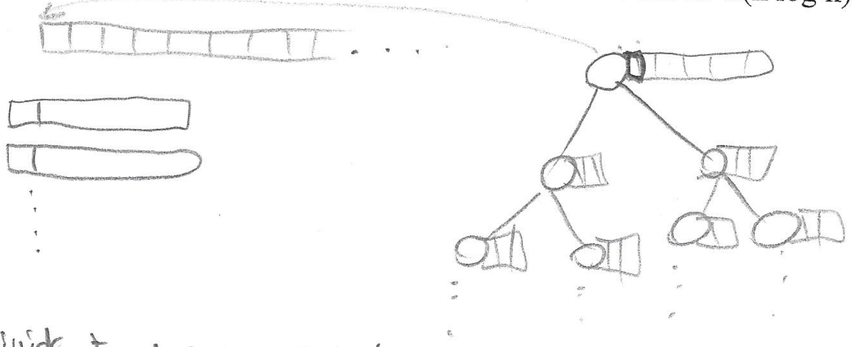
Foil2 = $ac + adi + cdi + db$

$M_1 = ac - db$
$M_2 = (ad + cb)i$
$M_3 = M_2 - M_1$

M_1
M_2
$M_3 = M_2 - M_1$?

$(a+bi)(c+di)$
$ac + adi + bci + dbi^2$
$ac + adi + cbi - db$
$(ac-db) + (ad+cb)i$

2. (10) Suppose that you are given a sequence of n elements to sort. The input sequence consists of n/k subsequences, each containing k elements. The elements in a given subsequence are all smaller than the elements in the succeeding subsequence and larger than the elements in the preceding subsequence. Thus, all that is needed to sort the whole sequence of length n is to sort the k elements in each of the n/k subsequences. Design an algorithm that sorts the list in O(n log k) running time.



1. Divide Input into sub lists
2. Take the head of each sublist and have it be a node
3. Use nodes to create a min heap tree
4. After the min heap is created, pop the root of the tree into a new list but leave the remaining list.
5. Then take the following index to get a new root.
6. run min heap
7. reapeat steps 2-6