COMP.4040 **Mid-term 2** Sp19

Name (Print): Dang Nhi Ngo

Problem Number(s)	Possible Points	Earned Points
1 (1)	10	10
1 (2)	15	15
2	20	20
3 (1)	10	0
3 (2)	25	25
4 (1)	14	8
4 (2)	6	3
	TOTAL POINTS 100	

Exam Time: 50 minutes, 5 problems (8 pages, including this page)

- Print your name on this page and the last page, put your initials on the rest of the pages.
- This exam is close book and notes, and **Closed** neighbors, mobile devices, and internet. Only two-page of cheat sheets (prepared by yourself) are allowed.
- If needed, use the back of each page or the last page.
- Show your work to get partial credits.
- Show your rational if asked. Just giving an answer can't earn full credits.
- You may use any algorithm (procedure) that we learned in the class, assuming nothing changed in the algorithm.
- Keep the answers as brief and clear as possible.

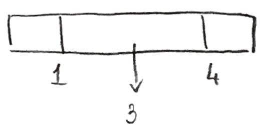
Name (Print): Dang Nhi Ngo

1. (25 points) **QuickSort**

Consider the quick sort algorithm in our textbook on page 171 as shown below

```
QUICKSORT(A, p, r)
  if p < r
      q = PARTITION(A, p, r)
      QUICKSORT(A, p, q - 1)
      QUICKSORT(A, q + 1, r)
```

(1) (10 points) For an array A with n distinct elements, how often can we expect to see a split that's 4-to-1 (or 1-to-4) or better? Assume the pivot is equally likely to end up anywhere in the sub-array after partitioning. Explain your answer.



$\alpha = 1/5 \Rightarrow 1 - \alpha = 4/5$ $\quad 0 \leqslant \alpha \leqslant \frac{1}{2}$

$T(n) = T(\frac{n}{5}) + T(\frac{4n}{5}) + \Theta(n)$

$Pr\{A\} = \frac{3n}{5} \times \frac{1}{n} = \frac{3}{5} = 0.6$

So 60% that we can see a split that's 4-to-1 (or 1-to-4) or better

(2) (15 points) Assume that the algorithm always produces a 4-to-1 split, provide a tight bound on the running time of quicksort in this case. Show your answer <u>in recurrence and then solve the recurrence</u>. You do not need to prove the answer with the substitution method.

$\alpha = 1/5 \Rightarrow 1 - \alpha = 4/5$ $\quad 0 \leqslant \alpha \leqslant 1/2$

$T(n) = T(\frac{n}{5}) + T(\frac{4n}{5}) + \Theta(n)$

By Theorem: $T(n) = \Theta(n\lg n)$

Name (Print): Dang Nhi Ngo

2. (20 points) Use indicator random variables to solve the following problem:
There are n people at a circular table in a restaurant. On the table there are n different appetizers arranged on a big Lazy Susan (which is a turntable or rotating tray placed on a table or countertop to help distribute food). Each person starts eating the appetizer directly in front of him or her. Then a waiter spins the Lazy Susan so that everyone is faced with a random appetizer.

What is the expected number of people that end up with the appetizer that they had originally? You must define necessary random variable and/or indicator random variable clearly. Also, be sure to show the detailed steps of calculating the final result. Specify the rule or lemma used when applicable.

X : number of people that end up with the appetizer that they had originally

X_i : Indicator Random Variable associated with the event that person i ends up with the appetizer that he had originally

$X_i = I\{$ person i ends up with the appetizer that he had originally $\}$

$= \begin{cases} 1 & \text{if they do} \\ 0 & \text{if they don't} \end{cases}$

By Lemma: $E[X_i] = Pr(X_i$ ends up with the appetizer that he had originally$)$

$= 1/n$

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

Therefore, the expected number of people that end up with the appetizer that they had originally is 1

Name (Print): Long Hai Ngo

3. (35 points) **Heap and its application**

(1) (10 points) We have learned that the running time of MAX-HEAPIFY on a node is O(*lgn*) for a binary heap *A* that has *n* (n equals to A.length) nodes. The following algorithm BUILD-MAX-HEAP calls MAX-HEAPIFY O(n) times. This seems to suggest that the running time for BUILD-MAX-HEAP is O(*n lgn*). Is this an asymptotic tight upper bound? Briefly explain your answer. You don't need to formally approve it.

```
BUILD-MAX-HEAP(A)
  A.heap-size = A.length
  for i = ⌊A.length/2⌋ downto 1
      MAX-HEAPIFY(A, i)            O(lgn)
```

Yes, this is an asymptotic tight upper bound.

Running time for Build-Max-Heap is O(nlgn)

$$T(n) = \frac{n}{2} O(lgn)$$
$$= O(nlgn)$$

Name (Print): DangNhi Ngo

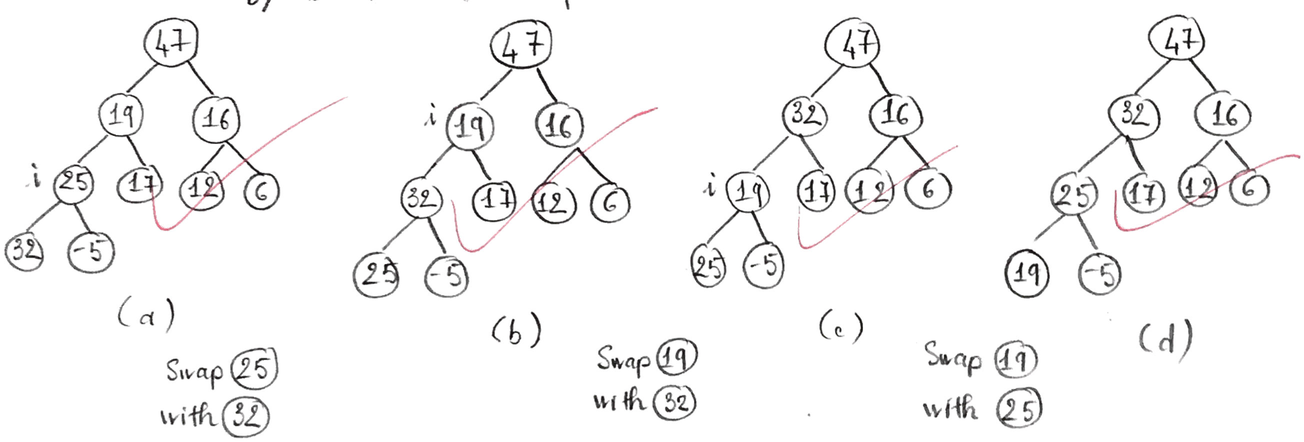
(2). (25 points) Consider the given array <47, 19, 16, 25, 17, 12, 6, 32, -5>

(a) Is this a **binary max-heap**? Justify the answer.

(b) If your answer is yes in (1), show the heap **in its binary tree view**.
If your answer is no, make it a max-heap using the appropriate algorithm(s) that we learned <u>in the textbook and class</u>. Show the steps of how to get the max-heap. And then show the final answer **in an array view**.

a/ No, this is not a binary max-heap. Because at the left-side, the child node 32 is larger than the parent node 25.

b/ Build a max-heap



Final answer in an array view:

⟨47, 32, 16, 25, 17, 12, 6, 19, -5⟩

Name (Print): Dang Nhi Ngo

4. (20 points) **Design and Analysis of an Algorithm**

Consider an unsorted array A of n integers; design an efficient algorithm that accepts A, n and s as the inputs and determines if the array contains two integers such that they add up to a specific target number s. That is: if we can find A[i] + A[j] == s ($1 \leq i, j \leq n, i \neq j$), the algorithm should return TRUE, otherwise return FALSE.

Design requirement:

- the *efficient* algorithm you are going to design should provide an **O(nlgn)** running time, rather than an $O(n^2)$ running-time solution.
- To keep your answers brief, you may use any algorithms that we have learned from lectures and the textbook as subroutines (this means you do NOT need to re-write those algorithms).

(1) (14 points) Algorithm Pseudocode *(please use textbook conventions)*: 8

Algorithm:

+ Sort the array A in increasing order
+ Compare sum of two integers with the specific target number s

$\theta(n\lg n)$ Merge Sort (A, 1, n)

```
Target Number (A, n, s)
    Merge Sort (A, 1, n)
    sum = 0
    for i = 1 to n - 1
        for J = i + 1 to n
            sum = A[i] + A[J]
            if (sum == s)
                return true
    return false
```

$\theta(n)$

this is $\Theta(n^2)$

$\theta(1)$

```
FindSum (A, n, s)
    // sort first
        MERGE_SORT (A, 1, n)        ] O(nlgn)
        or HEAP_SORT (A)
    // find a pair
    i = 1 , J = n
    while (i < J)
        if A[i] + A[J] == s
            return TRUE
        else if A[i] + A[J] > s
            i = J - 1
        else i++
    return FALSE
```

Name (Print): Dang Nhi Ngo

(2) (6 points) What is the running time of the algorithm that you designed? Justify your answer.

3

X

$$T(n) = \Theta(n\lg n) + \Theta(n) + \Theta(1)$$
$$= \Theta(n\lg n)$$

Name (Print): Dang Nhi Ngo

(You may use this page to write answers if needed. Please mark the problem number clearly)