Faculty of Science

Course:	CSCI 3070U: Analysis and Design of Algorithms
Course component:	Midterm Solutions
Semester:	Fall 2017

Question 1 (12 marks)

a. (4 marks) Solve the recurrence $T(n) = T\left(\frac{n}{2}\right) + c$ using the master method. Show your work (e.g. case number used, result of $\log_b a$, etc.). Write your answer in theta (Θ) notation.

Rubric:

- 1 mark – calculating $\log_b a$
- 1 mark – identifying the correct case
- 1 mark – demonstrating the applicability of the case
- 1 mark – conclusion in big-theta notation

Solution:

$$\log_b a = \log_2 1 = 0$$

$$f(n) = c \text{ is } \Theta\left(n^{\log_b a}\right) = \Theta(n^0)\text{, so case 2 applies:}$$

$$T(n) = \Theta(n^{\log_b a} \log n) = \Theta(n^0 \log n) = \Theta(\log n)$$

b. (4 marks) Solve the recurrence $T(n) = 8T\left(\frac{n}{2}\right) + n^4 \log_2 n$ using the master method. Show your work (e.g. case number used, result of $\log_b a$, etc.). Write your answer in theta (Θ) notation.

Rubric:

- 1 mark – calculating $\log_b a$
- 1 mark – identifying the correct case
- 1 mark – demonstrating the applicability of the case
- 1 mark – conclusion in big-theta notation

Solution:

$$\log_b a = \log_2 8 = 3$$

$$f(n) = n^4 \log_2 n \text{ is } \Omega\left(n^{\log_b a + \varepsilon}\right) = \Omega(n^{3+\varepsilon})\text{, for } \varepsilon = 1\text{, so case 3 applies:}$$

$$T(n) = \Theta\big(f(n)\big) = \Theta(n^4 \log_2 n)$$

c. (4 marks) Solve the recurrence $T(n) = 9T\left(\frac{n}{3}\right) + n \log_2 n$ using the master method. Show your work (e.g. case number used, result of $\log_b a$, etc.). Write your answer in theta (Θ) notation.

Rubric:

- 1 mark – calculating $\log_b a$
- 1 mark – identifying the correct case
- 1 mark – demonstrating the applicability of the case
- 1 mark – conclusion in big-theta notation

Solution:

$$\log_b a = \log_3 9 = 2$$

$$f(n) = n \log_2 n \text{ is } O\left(n^{\log_b a - \varepsilon}\right) = O(n^{2-\varepsilon}), \text{ for } \varepsilon = 0.1, \text{ so case 1 applies:}$$

$$T(n) = \Theta(n^{\log_b a}) = \Theta(n^2)$$

Question 2 (15 marks)

1. (7 marks) For the following algorithm, find the time-complexity recurrence, T(n,k), which represents the worse-case complexity of the algorithm. Include a T expression for the base case, as well, for completeness.

```
POWER-OF-LIST(L, k)
    if k = 0 then                                        <= 1 repetitions
       base = []                                               O(1)
       for i = 1 to L.length do                                n repetitions
          append 1 to base                                           O(1)
       return base                                             O(1)

    kMinusOne = POWER-OF-LIST(L, k-1)                    T(n,k-1)

    result = []                                          O(1)
    for i = 1 to L.length do                             n repetitions
       currentPower = L[i] * kMinusOne[i]                      O(1)
       append currentPower to result                           O(1)

    return result                                        O(1)
```

Rubric:

- 2 marks – show line-by-line cost
- 4 marks – correct recurrence
- 1 mark – correct base case

Solution:

```
T(n,0) = O(n)
T(n,k) = T(n,k-1) + O(n)
```

2. (8 marks) Using substitution, solve the recurrence for part 1. Show your work, and write your answer in big-O notation. Be sure to find the tightest bound possible.

   Rubric:
   - 1 mark – base case
   - 2 marks – correct structure (assumption for $< k$, inductive step for $= k$)
   - 4 marks – solving the inequality
   - 1 mark – conclusion in big-O notation

   Solution:
   Guess: $T(n,k)$ is $O(nk)$

   Base case: $T(n, 0)$ is $O(n)$
   Therefore, $T(n, 0) \leq cn$, for some constant c

   Assumption: $T(n,x) \leq cnx$, for some constant c, for all $x < k$

   Inductive step: Prove $T(n,k) \leq cnk$

	$T(n,k) = T(n, k-1) + O(n)$	
	$= T(n, k-1) + cn$	(definition of $O(n)$)
	$\leq cn(k-1) + cn$	(by assumption)
	$= cn(k - 1 + 1)$	(factoring cn)
	$= cnk$	(arithmetic)

   Therefore, $T(n,k)$ is $O(nk)$.

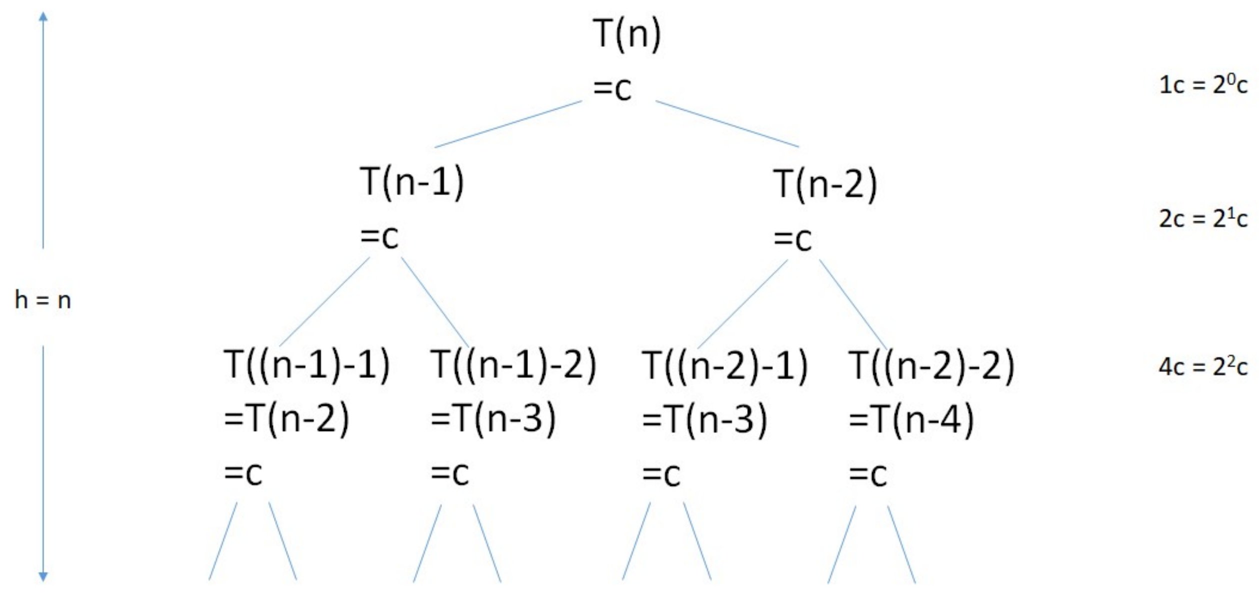
Question 3 (8 marks)

Using the recursion tree method, solve the recurrence $T(n) = T(n-1) + T(n-2) + c$. Show your work, including the height of the tree, and the summation term, which describes the overall size of the tree. Write your final answer in big-O notation.

Rubric:

- 1 mark – tree height
- 1 mark – sum for each tree level
- 1 mark – general term for tree levels
- 3 marks – summation term for the total for all levels
- 2 marks – big-oh notation for the result

Solution:



$$\mathrm{T(n)} = \sum_{i=0}^{n-1} 2^i c = 2^n c \in O(2^n)$$

Question 4 (15 marks)

For this question, you will simulate the execution of the COUNTING-SORT algorithm, discussed in the lectures on the following list for A:

8	2	4	6	1	5	2	1	3	8	2

Be sure that you show the following work as part of your answer:

1. C_0 – The C array, after counting occurrences
2. C_1 – The C array, after calculating the cumulative counts
3. C_2 – The C array, after updating those counts while building the B array
4. B – The resulting sorted array

***Note**: That showing only the B array will not be sufficient to earn any marks for this question.*

Rubric:

- 4 marks – C_0
- 4 marks – C_1
- 4 marks – C_2
- 3 marks – B

Solution:

C_0:

2	3	1	1	1	1	0	2

C_1:

2	5	6	7	8	9	9	11

C_2:

0	2	5	6	7	8	9	9

B:

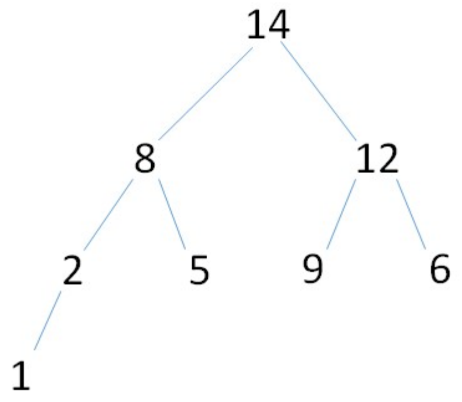
1	1	2	2	2	3	4	5	6	8	8

Question 5 (10 marks)

Below is a max heap, and its array representation. Draw the resulting max heap (both the tree and the array representation) after the following code executes:

```
HEAP-INSERT(10)
HEAP-EXTRACT-MAX()
HEAP-EXTRACT-MAX()
HEAP-EXTRACT-MAX()
```

Note: *Only the final heap tree and array will be marked. Some workspace has been included on the following pages, but please write your final answer at the bottom of the current page.*

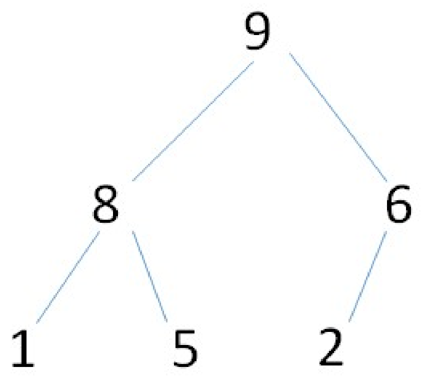


14	8	12	2	5	9	6	1			

Rubric:

- 6 marks – tree representation
- 4 marks – array representation

Solution:



9	8	6	1	5	2					

Question 6 (20 marks)

Write a function (`partitionWithBalancedSums`) in Python, Java, or C++ that takes a list/array/ArrayList/vector (`numbers`) as an argument, and returns an integer containing the dividing point within `numbers` such that the sum of `numbers` up to that index is as close as possible to the sum after that index. Your algorithm must use Dynamic Programming to accomplish its task.

Required strategy:

1. Create a 2D array, such that $\texttt{sums}_{\texttt{ij}}$ - the sum of $\texttt{numbers}_{\texttt{i}}$ .. $\texttt{numbers}_{\texttt{j}}$ in `numbers`.
2. Search this 2D array for the optimal partition point

Sample output:

```
>>> partitionWithBalancedSums([1,6,3,4,14])
3
>>> partitionWithBalancedSums([32,6,11,17,2,12,30,8,27])
4
```

Rubric:

- 4 marks – initializing the 2d array
- 10 marks – calculating the sums for all sums_{ij}
  - Be sure that they do this as efficiently as possible under the strategy
- 6 marks – finding the division point with closest sums

Solution (in Python):

```
def partitionWithBalancedSums(numbers):
    # create a 2D array of sums
    # sums[i][j] - the sum of 'numbers' from numbers[i] to numbers[j]
    # Note:  The table only needs to be filled for i <= j
    sums = []
    for rowNum in range(len(numbers)):
        row = []
        for col in range(len(numbers)):
            row.append(0)
        sums.append(row)

    # calculate the sub-list sums
    for row in range(len(numbers)):
        for col in range(row, len(numbers)):
            previous = 0
            if col > 0:
                previous = sums[row][col-1]
            sums[row][col] = previous + numbers[col]

    # figure out the optimal cutting point
    minDifference = float('inf')
    dividePoint = -1
    for i in range(0, len(numbers)-1):
        # cut after the ith position
        difference = abs(sums[0][i] - sums[i+1][len(numbers)-1])
        if difference < minDifference:
            minDifference = difference
            dividePoint = i

    return dividePoint
```