

Faculty of Science

**Course**: CSCI 3070U: Analysis and Design of Algorithms

**Instructor:** Randy J. Fortier

**Course component:** Midterm Exam

**Weight:** 25%

**Duration:** 80 minutes

# Question 1 (4 marks)

1. (2 marks) Solve the recurrence T(n) = 4T(n/2)+ n/lg n using any method.

Method: Master method, case 1

Solution: Θ(n2)

1. (2 marks) Solve the recurrence T(n) = 2T(n/4)+ n0.51 using any method.

Method: Master method, case 3

Solution: Θ(n0.51)

# Question 2 (10 marks)

Using the recursion tree method, solve the following recurrence: T(n) = T(n-1) + T(n-2) + c. Express your answer as a simplified summation (or a more direct result, if possible).

C 1c = 20c

C C 2c = 21c

C C C C 4c = 22c

Height of tree: n

# Question 3 (10 marks)

Determine a recurrence for the recursive solution above, T(n), and solve the recurrence using substitution to find the running time complexity (in terms of disc moves) of the recursive solution to the Towers of Hanoi problem.

T(n) = 2T(n-1) + 1

Guess: 2n – 1

For k = 0:

T(0) = 20 – 1 = 1-1 = 0

It takes zero moves to move zero discs, true!

Assume: T(k) = 2k – 1, for all k < n

Prove: T(n) = 2n – 1

T(n) = 2T(n - 1) + 1

=2(2n-1 - 1) + 1

=2\*2n-1 - 2\*1 + 1

=2n - 2 + 1

=2n - 1

# Question 4 (10 marks)

Formally, determine the running time complexity of the following algorithm. Show your work.

FIND-MAX(list, start, end):

1 if start >= end then Θ(1)

2 return list[start] Θ(1)

3 middle = start + (end – start) / 2 Θ(1)

4 max1 = FIND-MAX(list, start, middle) T(n/2)

5 max2 = FIND-MAX(list, middle+1, end) T(n/2)

6 if max1 > max2 then Θ(1)

7 return max1 Θ(1)

8 else Θ(1)

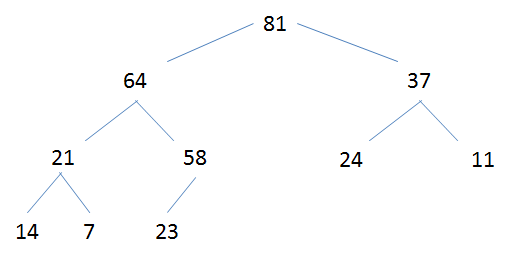
9 return max2 Θ(1)

T(n) = 2T(n/2) + Θ(1)

By the master method, case 1: T(n) = Θ(n)

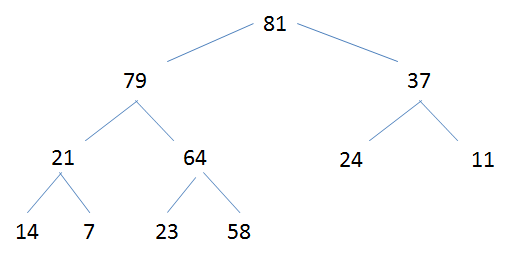
# Question 5 (10 marks)

Below is a max heap, and its array representation. Draw the resulting max heap (and array representation) after a call to INSERT(79).



|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 81 | 64 | 37 | 21 | 58 | 24 | 11 | 14 | 7 | 23 |

Solution:



|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 81 | 79 | 37 | 21 | 64 | 24 | 11 | 14 | 7 | 23 | 58 |

# Question 6 (8 marks)

Including an example for each, describe the conditions of applicability for following algorithm design techniques.

1. Dynamic programming

Example: Calculating Fibonacci numbers by storing smaller Fibonacci numbers in a memo table as they are computed toward a larger Fibonacci number

1. Subproblems always have the same solution
2. Overlapping subproblems (the same subproblems are calculated multiple times while solving the problem)
3. Greedy algorithms

Example: Filling up a container with fuel, where each fuel has its ability to cook food (a variation of the fractional knapsack problem).

1. Subproblems always have the same solution
2. Overlapping subproblems
3. A series of locally optimal choices must lead to a globally optimal solution

# Question 7 (8 marks)

***Note****: 2-3 sentences should be enough for this question. Extra content increases your risk of saying something incorrect, which will cost you marks even if you have the correct answer elsewhere. I do not cherry pick the correct answers out of a list.*

How is it possible that comparison sorting requires Ω(n lg n) operations, yet linear sort using Θ(n) operations? What mechanism do several linear sort algorithms use to overcome this limitation?

Comparison sorting uses comparison operators (e.g. <), which make binary decisions (decisions with exactly two possible outcomes). Linear sorting takes advantage of other operators that are not binary. One common such operator is array indexing ([]), which (in constant time) can make an n-ary decision, where n is the size of the array.

# Question 8 (20 marks)

Design a greedy algorithm, and write the pseudocode for it, to solve the following problem.

*A bike messenger, Jackie, injured her leg muscle, yet she still needs to deliver packages to a number of buildings. Some of the packages to be delivered are heavy, and some are light. Some of the buildings are close together, and some are far apart. Jackie compiled a table, T, of travel times (in hours) such that tij is the time it takes to bike from building i to building j. When her company drops off the packages, it includes a list W, such that wi is the mass of package i in kilograms. Also included is a destination list D, such that di is the destination building for package i. Given D, W and T, determine a route that tries to minimize the number of kg-hours Jackie has to spend riding her bike. Jackie can begin and end her deliveries at any building. Assume the company drives her, her packages, and her bike to the first building and picks her up at the last building when finished.*

*Note: A kg-hour is the stress of riding the bike for one hour, carrying 1 kg of packages. Jackie’s weight, and the bike’s weight, are not considered for this problem.*

Hint: Dropping off heavy items early in the trip will help reduce the strain, but not at the expense of driving for a longer time.

MINIMIZE-STRAIN(D, W, T):

for start in D do

subpathCost ← MINIMIZE-STRAIN(D – {start}, W, T)

cost ← subpathCost + wi\*ti

if cost < minCost then

minCost = cost

endif

endfor

return minCost