Midterm Exam

Joseph Jabour

CSE 8833 Algorithms
Prof. Banicescu

Problem 1:

We consider the merge sort algorithm that uses a divide-and-conquer design
strategy. The running time of a merge sort algorithm is given below:
• T(n) = (2 if n = 2; 2T(n2) + n if n = 2k, k > 1)
a) Give the name of the type of recursion (linear homogeneous, nonlinear recursion, etc..)
defined by T(n).
b) Prove that the expression giving the complexity of the merge sort algorithm with which
you are already familiar is also a solution to the above recurrence. Hint: use mathematical
induction.

Answer

a)

This particular use case of Merge Sort is a recurrent type of linear nonhomogeneous recurrence.

b)

To prove the expression, we need to prove that $T(n) = \Theta(nlogn)$
Basis: T(2) = 2log2 = 2 = T(n). This is true
Inductive step: Assume for $n = 2^k$ and k>1, we assume that $T(2^k) = 2^k log 2^k$, which will show that $T(2^k) = k2^k$

Assume true for $T(2^k) = 2T(2^k/2) + 2^k = 2T(2^{k-1}) + 2^k$

To prove that this is true for $T(2^{k+1})$:

$T(2^{k+1}) = 2T(2^k) + 2^{k+1}$
$= 2[2^k log 2^k] + 2^{k+1}$
$= 2^{k+1}k + 2^{k+1}$
$= (k+1)2^{k+1}$
Therefore, $T(n) = \Theta(nlogn)$ QED

Problem 2

(4 points) Tower of Hanoi: Consider moving a stack of disks with different sizes with three
pegs. Initially, all disks are stored and placed on the first peg. One can only move the

topmost disk from a peg to another peg. At any time, disks on each peg must be sorted as well. How many steps does it take to move all disks from one peg to another? [Hint: You must first express the solution to the number of steps expressed in a recursive form; then, you need to find the solution to this recursive expression].

Answer

The recursive form of this problem is
$T(n) = 2T(n-1) + 1$

To move n disks from the starting position to where it needs to do, you must move every disk above it to a location which won't be their final location, n-1, which requires T(n-1) steps.
Then you must move the largest disk from its starting position to it's target peg, which requires 1 step.
Then, you must move the n-1 disks from where they are to their final location, which will take T(n-1) steps.

Problem 3

Prove by induction that the number of nodes of a full binary tree with depth d is: $2^{d+1} - 1$

Answer

In a binary tree, the root node is represented as having a depth of 0. This can prove our base case
The number of nodes in the root level is 1, which is shown by inserting 0 for d into our formula
$2^{d+1} - 1$
$2^{0+1} - 1$
$2^1 - 1 = 1$

Our hypothesis will be that our formula holds for any arbitrary k where k>0 and k<=d. Any tree has at most $2^{d+1} - 1$ nodes in it.

If we consider a rooted binary tree T of depth k+1, and let T_L be the subtree on the left child of root T and T_R represent the subtree to the right, then because the depth of our binary tree is at least 1, the root has at least one child. The path from the root of T to any leaf has to go through the root of T_L, which is the left child of root of T. This means that the depth of both T_L and T_R are at most k. By inductive hypothesis, these subtrees have at most $2^{d+1} - 1$ each

---

Problem 4

Let A be a heap of size n. Give the description of the most efficient algorithm for the following tasks:
a) Find the sum of all elements.
b) Find the sum of the largest logn elements.

Answer

a)
To describe what I would do to find a sum of all elements in a heap of size n,

- I would first create a variable "sum" to store the sum of all elements and set it to 0.
- I would have to label the root node of the heap, and start a loop that moves through the heap until it passed each elements
- For each time the loop finds an element, I would add its value to the sum
- Upon finishing, the variable would contain a sum of all elements

```
def summarizeElements(elements):
    sum = 0
    for i in elements:
        sum = sum + i

    return sum
```

b)

To find the sum of the largest log(n) elements, I would use a similar technique of summarizing all the elements.
However, I would use a max heap from the given heap using a bottom up approach, which takes O(n) time.
Then, perform log(n) iterations of:

- Remove the maximum elements from the max heap.
- Add it to the 'sum' variable

This will result in the 'sum' variable containing the sum of the largest log(n) elements in the heap.

Problem 5

a) Give the names of the major design strategies of algorithm design.
b) List the similarities and the differences between the greedy and dynamic programming strategies

Answer

a)
There are many major design strategies in algorithm design. Some are as follows

- Divide and Conquer
- Greedy Algorithms

- Brute Force
- Dynamic Programming
- Linear Programming
- Reduction
- Heuristic Methods

b)
The biggest difference between dynamic programming and the greedy method are that the greedy method makes optimal choices for the most current given moment, in the hopes that it finds a solution to the overall global problem.
Dynmaic programming will store solutions into a data table and retrieve the results later in an attempt to reduce time complexity. It still may have to solve a problem at the moment in the instance it does not have a solution stored, in which case it will solve the solution in the moment similar to the greedy method.

Problem 6

List the specific uses of each: the breadth first search and the depth first search.

Answer

Breadth First Search (BFS) is more useful when only a single answer is needed (or node needs to be found). This is also more useful when the depth of the tree may vary
Depth First Search (DFS) will traverse the entire tree, and is mroe useful when the target needed is far from the root. DFS is also more memory efficient

---

Problem 7

Scheduling jobs on a single machine is considered an optimization problem for maximizing profitability for two cases: (i) minimize the average time that a job spends in the system, and (ii) maximize the profit on jobs that can be scheduled before a deadline. Give the proof (or at least describe the outline of a proof) that the greedy algorithm (i) for minimizing the average time a job spend in the system is optimal. You may consider P1, P2, .., Pn jobs in the system with corresponding service times S1, S2, .., Sn. [Hint: We explained the proof in detail in class].

Answer

The greedy algorithm is optimal in minimizing the average time a job spends by nature of the greedy algorithm. It will naturally find a short term solution to the problem, in this case, picking a job with the shortest run time.

If you have P1, P2, etc jobs with service times S1, S2, etc, then you can extend the schedule to include the (k+1)th job so you still have an optimal schedule for the first K+1 jobs. Creating a short term optimal solution creates an optimal solution to the whole problem.

Problem 8

The following instance for the knapsack problem is given to you to be solved using a greedy strategy. There are three plausible solutions: (i) selection in decreasing value, (ii) selection in increasing weight (iii) selection in decreasing value per weight. Show these plausible solutions in a table and verify that the general solution offered by the greedy algorithm for the knapsack problem is the optimal solution in this particular case also. [Hint: problem quite similar to the one in your handout]

Answer

Assume a knapsack and a set of items with values (V1, V2, etc) and weights (W1, W2, etc)

If you assume all objects can fit into the knapsack, the greedy algorithm will definitely find the optimal solution as it will always find the optimal solution for each smaller situation

Let v_i be the value v of object i and w_i be the weight w of object i
Let X = (x_i.. x_n) be the solution found by the greedy algorithm

n = 5, W = 100

```
w   10 20  30    40 50
v   20 30  68    40 60
v/w 2  2/3 15/34 1  5/6
```

As shown in the table above, the greedy algorithm should be able to determine the optimal solution given each instance of a problem.