Yash Patole

In class Midterm Exercise:

Problem 1

Using the very definition of Big-omega notation, prove that n3 logn is Ω(n3). You must use the definition and finding the constants in the definition to receive credit.

Answer:

there is a constant c>0 and an integer n0>0 such that n3 logn>= c * n3.

c=1 and n0 =2, then the inequality holds.

Problem 2

Given that T(n) = 1 if n=0 and T(n) = T(n-1)+ 2n otherwise; show, by induction, that T(n) = 2n+1 -1. Show all three steps of your induction explicitly.

Answer:

The base case T (0) = 2-1 =1 which is correct.

Therefore, T(n) = T(n-1) +2n = 2n-1+1 -1 +2n = 2n+1 -1.

Hence, it is correct for any n less than n-1, it is true for n.

Problem 3

Is the bucket-sort algorithm in-place? Why or why not?

Answer:

Bucket-sort does not use a constant amount of additional storage. It uses O(n + N) space. Therefore, the bucket-sort algorithm is not in place.

Problem 4

Recall the Extendable Array Implementation from Sections 1.4.1 and 1.4.2. Now consider an extendable table that supports both add and remove methods. Suppose we grow the underlying array implementing the table by doubling its

capacity any time we need to increase the size of this array, and we shrink the underlying array by half any time the number of (actual) elements in the table dips below N/4, where N is the current capacity of the array. Use amortization method with cyber dollars to show that a sequence of n add and remove methods, starting from an array with capacity N = 1, takes O (n) time. You must use amortization Technique with cyber dollars to receive any credit.

Answer:

Assuming that the add, remove, and copy operations cost one cyber dollar each, we charge three cyber dollars for adding an element to the array, conserving two cyber dollars for future copies when the array is increased. In addition, we charge an additional 3 cyber dollars when it is added for any future removal. In the worst-case scenario, n add operations cost 3n cyber dollars, whereas n remove operations cost 3n cyber dollars. It should be noted that reducing the Array from N to 1 is equivalent to extending the array from 1 to N. As a result, the total number of operations costs 6n, which equals O (n).

Problem 5

Pseudo code the Algorithm findAllElements(k),which returns all the items with keys equal to k in a balanced search tree, and show that it runs in time O(log n + s), where n is the number of elements stored in the tree and s is the number of items returned.

Answer:

```
Algorithm findAllElements(k,v,c):
Input: The search key, k, a node, v, of the binary search tree T and a container c
Output: An enumeration of the elements found
if v is an external node then
return c.elements()
if k = key(v) then
  c.addElement(v)
return findAllElements(k,T.rightChild(v),c)
else if k < key(v) then
  return findAllElements(k,T.leftChild(v))
else
return findAllElements(k,T.rightChild(v))
```

In the worst case the find traverse the height of the tree recursively for logn and store s elements, hence O (logn +s).

Problem 6

Consider an n-by-n matrix M whose elements are 0's and 1's such that in any row, all the 1's come before any 0's in that row. Assuming A is already in memory, describe an algorithm running in O(n) time for finding the row of M that contain the most of 1's.

Answer:

To count the number of ones in A, we look for the final one in the first row at position k.

In the following row, at position k: if the value is zero, we proceed to the next row at location k. If the value is 1, we look for the previous one and set k to be the new k. To find the final k, we repeat this method till the last row.

This takes O(n) time because we made an n comparison at position k and updated its value at most times.

Problem 7

Develop an algorithm that computes the kth smallest element of a set of n distinct integers in O (n + k log n) time.

Answer:

Create a heap and then call removeMinElement k times.

It takes O(n) time to construct a heap. Taking away the min components takes O(klogn), for a total of O(n+ klogn).

Problem 8

Suppose a social network, N, contains n people, m edges, and c connected components. What is the exact number of times that each of the methods,

makeSet, union, and find, are called in computing the connected components for N using Algorithm 7.2?

Answer:

The method makeSet is called n times.

The find method is used 2m times (one for each edge) and once to print all find(x) for each node x, for a total of 2m+n times.

If we suppose that the find methods were called again in the Union (find(x), find(y)) instruction, which is unlikely, then we must add another 2(n-c) to the sum, which becomes 2m+2(n-c)+n.

The union method is only used to build a tree in the connected component, which has the number of vertices minus one. Because we have c components, the Union is referred to as (nc) times.

Problem 9

Let A be a collection of objects. Describe an efficient method for converting A into a set. That is, remove all duplicates from A. What is the running time of this method?

Answer:

Assuming the objects have a total order, we begin by sorting the objects of A. Then we may go over the sorted sequence and eliminate all duplicates. Sorting takes O(nlogn) time, and removing duplicates takes n time. As a result, this is an O(nlogn)-time algorithm.

If no total order is defined for the objects, we can only determine whether two objects are identical. In this scenario, we bucket sort them so that similar things are in the same bucket.

Then, for each bucket, we choose one object to form a set.

For comparison, the running time is 1+2+3+...+(n-1)= n*(n-1)/2= O(n2).

Problem 10

Consider a single machine scheduling problem, where we are given a set, T, of tasks specified by their start times and finish times, as in the task scheduling problem, except now we have only one machine and we wish to maximize the number of tasks that this single machine performs. Design a greedy algorithm for this single machine scheduling problem.

Answer:

The Greedy option considers tasks by increasing completion times. As a result, we arrange the tasks in ascending order based on their completion times. Now we go through the sorted tasks and schedule the things that are not in conflict.

Problem 11

In the Art Gallery Guarding Problem we are given a line L that represent a long hallway in an art gallery. We are also given a set X= {x0, x1, x2, ..., x n-1 } of real numbers that specify the positions of paintings in this hallway. Suppose that a single guard can protect all the paintings within a distance at most 1 of his or her position (on both sides). Design an algorithm for finding a placement of guards that uses the minimum number of guards to guard all the paintings with positions in X. No pseudo-code required.

Answer:

We can apply a greedy method to cover all of the selected locations on L with the fewest number of length-2 intervals (for such an interval is the distance that one guard can protect). This greedy method begins with x0 and traverses all points within distance 2 of x0. If xi is the next uncovered point, we repeat the covering phase beginning with xi. This technique is then repeated until we have covered all of the points in X.

Problem 12

Show that we can solve the telescope scheduling problem in O(n) time even if the list of n observation requests is not given to us in sorted order, provided that start and finish times are given as integer indices in the range from 1 to n^2.

Answer:

According to Theorem 12.3, the Telescope issue can be solved in O(n) time if the list of observations is ordered by start and finish timings. So, all we must do is sort them O(n) times. This course has only covered the radix-sort algorithm, which runs in O(n) time.

To achieve O(n) sorting, we store start and finish times in a Bucket B such that B[1] contains the sequence of all observations with s=1 and B[i] contains the sequence of all observations with (i-1)2 s= i2, for i=1 to n. Because the array Bucket now has a size of n, the second loop in the Bucket sort only runs from 1 to n.