Please fill in the following information.

THEN PUT YOUR EMAIL ADDRESS ON EVERY PAGE OF THE EXAM !

NAME: **SOLUTION**

Email (please put complete address):

Neatness counts. We will not grade what we cannot read.

Exam is worth 100 points. Please phrase your answer succinctly. Write your answer on the same page, on which the question is given. There are four questions and a total of six pages. Make sure you turn in all these pages.

Do not attempt to look at other students' work. Keep your answers to yourself.
Any sort of cheating will result in a zero grade.

Read and sign the statement below. Wait for instructions to start the examination before continuing to the next page.

"I signify that the work shown in this examination booklet is my own and that I have not received any assistance from other students nor given any assistance to other students."

_____(Signature)

Q1. (15 points)

Given *f(n), g(n)* are two positive non-decreasing functions and c is a positive constant, prove or disprove:

$$g(n)^{cf(n)} = O(g(n)^{f(n)})$$

Answer:
The statement is false. To prove that we need to find two functions *f(n)* and *g(n)* and a constant c such that

$$g(n)^{cf(n)} \neq O(g(n)^{f(n)})$$

or equivalently

$$g(n)^{f(n)} = o(g(n)^{cf(n)})$$

But this is true for any strictly increasing functions *g(n)* and *f(n)* and any constant $c > 1$ since

$$\lim_{n\to\infty} \frac{g(n)^{f(n)}}{g(n)^{cf(n)}} = \lim_{n\to\infty} \frac{1}{g(n)^{(c-1)f(n)}} = 0$$

Q 2. (35 points)

(a) **(15 points)** Solve the following recurrence giving your solution in Θ notation:

$$T(n) = T(\frac{n}{2} + \sqrt{n}) + n$$

Answer: $T(n) = \Theta(n)$.

Clearly $T(n) = \Omega(n)$. Also *T(n)* is a monotone non decreasing function. Note, for $n > 16$, $\frac{n}{2} + \sqrt{(n)} < \frac{3n}{4}$ *so* $T(n) < T(\frac{3n}{4}) + n$. Using the Master Theorem we get $T(n) = O(n)$.

(b) **(20 points)** Suppose ***n*** balls are tossed randomly in ***b*** bins. Assume no ball is dropped on the floor and ***b > n.***

(i) What is the probability that a bin is empty? Must justify your answer.

$$(1 - \frac{1}{b})^n$$

(ii) What is the probability no bin has more than one ball? Must justify your answer.

The problem is similar to the *birthday paradox* problem, where we need to find the:

Probability[n people in a room having distinct birthdays, assuming b days in a year]

See answer on page 132 of the text (Third Edition), (or page 107 of the Second Edition)

Q3 (25 points) (10 points setting up the recurrence, 15 points solving the recurrence)

In the following assume for simplicity that n is a power of 2. Let $A(x)=\sum_{i=0}^{n} a_i x^i$ and $B(x)=\sum_{i=0}^{n} b_i x^i$ be two n-degree polynomials and $C(x)=A(x)B(x)=\sum_{i=0}^{2n} c_i x^i$ their product.

Consider the following way of rewriting the polynomial $A(x)$ as a combination of two $\frac{n}{2}$ -degree polynomials: $A(x)=A_1(x)+x^{\frac{n}{2}}A_2(x)$ where $A_1(x)=\sum_{i=0}^{\frac{n}{2}} a_i x^i$ and $A_2(x)=\sum_{i=0}^{\frac{n}{2}} a_{\frac{n}{2}+i} x^i$. Similarly $B(x)=B_1(x)+x^{\frac{n}{2}}B_2(x)$. Then

$$C(x)=A(x)B(x)=(A_1(x)+x^{\frac{n}{2}}A_2(x))(B_1(x)+x^{\frac{n}{2}}B_2(x))=$$
$$A_1(x)B_1(x)+x^{\frac{n}{2}}(A_1(x)B_2(x)+A_2(x)B_1(x))+x^n A_2(x)B_2(x)$$

What is the running time of the algorithm that computes the products $A_1(x)B_1(x)$, $A_1(x)B_2(x)$, $A_2(x)B_1(x)$, and $A_2(x)B_2(x)$ recursively and then computes $C(x)$ using the above formula?

Answer:

We are solving recursively 4 subproblems of size $\frac{n}{2}$ each. Furthermore the time to combine the solutions is the time needed to add polynomials of degree n which is easily seen to be $O(n)$. So the running time of the algorithm satisfies the following recurrence

$$T(n)=4T(\frac{n}{2})+O(n)$$

for which the solution according to the Master Theorem is: $T(n)=\Theta(n^2)$.

Q4 (25 points)

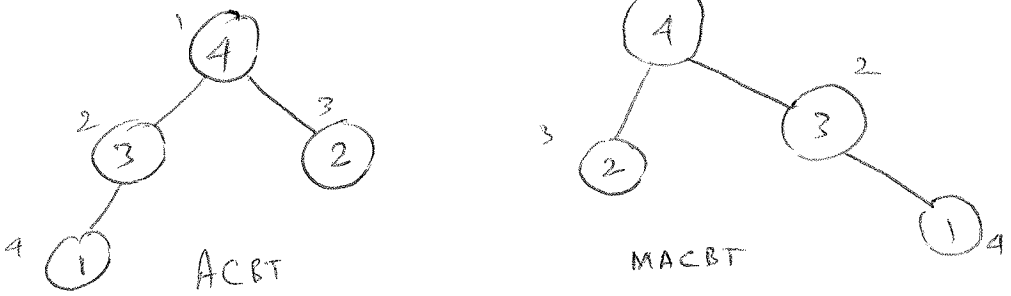
(a) (10 points) Professor Oliver uses the following algorithm for merging k sorted lists, each having n/k elements. He takes the first list and merges it with the second list using a linear time algorithm for merging two sorted lists, such as the merging algorithm used in merge sort. Then, he merges the resulting list of $2n/k$ elements with the third list, merges the list of $3n/k$ elements that results with the fourth list, and so forth, until he ends up with a single sorted list of all n elements.

Analyze the worst-case running time of the professor's algorithm in terms of n and k.

Answer:
Merging the first two lists, each of n/k elements, takes $2n/k$ time. Merging the resulting $2n/k$ elements with the third list of n/k elements takes $3n/k$ time, and so on. Thus for a total of k list, we have:

$$\begin{aligned} T(n) &= \frac{2n}{k} + \frac{3n}{k} + \cdots + \frac{kn}{k} \\ &= \sum_{i=2}^{k} \frac{in}{k} \\ &= \frac{n}{k} \sum_{i=2}^{k} i \\ &= \frac{n}{k} \frac{(k+2)(k-1)}{2} \\ &= \theta(nk) \end{aligned}$$

(b) (15 points) A "mirror" almost complete binary tree (MACBT) is an almost complete binary tree but skewed towards right. In an MACBT the right child of a node (with index *i*) has the index value 2**i* and the left child has the index value 2**i*+1. As example of an ACBT and its associated MACBT is given below. Accordingly, a heap structure can be built having the MACBT property.



Suppose an array A contains a heap data structure with *n* elements. In other words, A has an embedded ACBT structure. We would like to rearrange elements of A in manner that A remains a valid heap, but with an MACBT structure.

(i) Give a minimum cost algorithm for this change **(10 points)**

(ii) For what values of *n* your algorithm encounters the best case scenarios? **(5 points)**

Answer:

Since, no restriction is given on retaining the association of data value with the identity of a child (right ~ left), a simple answer is that *no re-arrangement or re-shuffling* of data in array *A* is needed. The new indexing relation for the MACBT embedded in the array *A* remains valid as the required heap can just be a mirror image of the original heap. To convince yourself, work out an example (also see the Figure shown above). There is no best or worst case scenario for this answer.

Alternatively, you can swap nodes in an ACBT structure across a level and correspondingly exchange elements of the array *A* using the parent-children relation of indices of ACBT to deduce indices for such exchange. Note, the maximum number of exchanges per level is half the number of nodes at each level, including the leaves, in case the last level is full. You may require Heap-Insert procedure at the last level, if it is not full. For this answer, the best case is when the last level of the tree is full.