Benha University
Faculty of Engineering (at Shoubra)
Electrical Engineering Department
M.Sc. (Computer Systems Engineering**)**

Midterm Exam
Subject: Advanced Algorithms - CES 608
Date: Sat 02/04/2016
Duration: 30 minutes

Attempt the following questions.

№ of Questions: 3 in 3 page(s)
Total Mark: 30

Question 1: **(10** Marks**)**

Consider the problem of adding two n-bit binary integers, stored in two n-element arrays A and B. The sum of the two integers should be stored in binary form in an $(n+1)$-element array C.

State the problem formally and write pseudocode for adding the two integers.

Solution:

This is how the given problem can be formally defined:

Input: Two n-bit binary integers, stored in two n-element arrays A and B.

Output: An $(n+1)$-element array C representing the sum of the two integers stored in A and B.

This is the pseudocode:

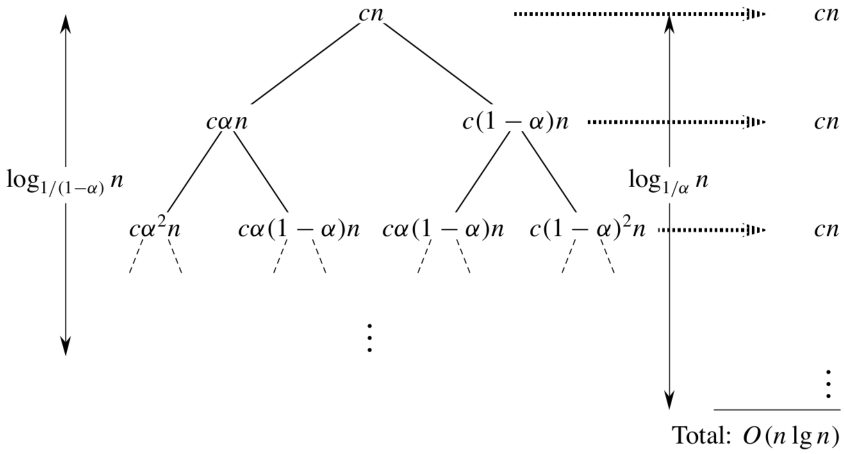
```
ADD-BINARY(A,B)
n = A.length
let C[1..n+1]  be a new array
c = 0
for i = 1 to n
    C[i] = (A[i] + B[i] + c) mod 2
    c = (A[i] + B[i] + c) / 2
C[i] = c
return C
```

Question 2: **(10 Marks)**

Use a recursion tree to give an asymptotically tight solution to the recurrence $T(n) = T(\alpha n) + T((1-\alpha)n) + cn$, where α is a constant in the range $0 < \alpha < 1$ and $c > 0$ is also a constant. Verify your solution by the substitution method.

Solution:

Without loss of generality, let $\alpha \geq 1-\alpha$, so that $0 < 1-\alpha \leq 1/2$ and $1/2 \leq \alpha < 1$.



The recursion tree is full for $\log_{1/(1-\alpha)} n$ levels, each contributing cn, so we guess $\Omega(n \log_{1/(1-\alpha)} n) = \Omega(n \lg n)$. It has $\log_{1/\alpha} n$ levels, each contributing $\leq cn$, so we guess $O(n \log_{1/\alpha} n) = O(n \lg n)$.

Now we show that $T(n) = \Theta(n \lg n)$ by substitution. To prove the upper bound, we need to show that $T(n) \leq dn \lg n$ for a suitable constant $d > 0$.

$$\begin{aligned}
T(n) &= T(\alpha n) + T((1-\alpha)n) + cn \\
&\leq d\alpha n \lg(\alpha n) + d(1-\alpha)n \lg((1-\alpha)n) + cn \\
&= d\alpha n \lg \alpha + d\alpha n \lg n + d(1-\alpha)n \lg(1-\alpha) + d(1-\alpha)n \lg n + cn \\
&= dn \lg n + dn(\alpha \lg \alpha + (1-\alpha)\lg(1-\alpha)) + cn \\
&\leq dn \lg n \ ,
\end{aligned}$$

if $dn(\alpha \lg \alpha + (1-\alpha)\lg(1-\alpha)) + cn \leq 0$. This condition is equivalent to

$$d(\alpha \lg \alpha + (1-\alpha)\lg(1-\alpha)) \leq -c \ .$$

Since $1/2 \leq \alpha < 1$ and $0 < 1-\alpha \leq 1/2$, we have that $\lg \alpha < 0$ and $\lg(1-\alpha) < 0$. Thus, $\alpha \lg \alpha + (1-\alpha)\lg(1-\alpha) < 0$, so that when we multiply both sides of the inequality by this factor, we need to reverse the inequality:

$$d \geq \frac{-c}{\alpha \lg \alpha + (1-\alpha)\lg(1-\alpha)}$$

or

$$d \geq \frac{c}{-\alpha \lg \alpha + -(1-\alpha)\lg(1-\alpha)} \ .$$

The fraction on the right-hand side is a positive constant, and so it suffices to pick any value of d that is greater than or equal to this fraction.

To prove the lower bound, we need to show that $T(n) \geq dn \lg n$ for a suitable constant $d > 0$. We can use the same proof as for the upper bound, substituting $\geq$ for $\leq$, and we get the requirement that

$$0 < d \leq \frac{c}{-\alpha \lg \alpha - (1-\alpha)\lg(1-\alpha)} \ .$$

Therefore, $T(n) = \Theta(n \lg n)$.

Question 3: **(10** Marks**)**

Suppose that instead of swapping element `A[i]` with a random element from the subarray `A[i..n]`, we swapped it with a random element from anywhere in the array:

```
PERMUTE-WITH-ALL(A)
1 n = A.length
2 for i = 1 to n
3     swap A[i] with A[RANDOM(1,n)]
```

Does this code produce a uniform random permutation? Why or why not?

Solution:

In the original and modified algorithm, line 3 is executed n times. For every index $i = 1 \ldots n$ in the original algorithm, there are $n - i + 1$ possible values returned by the random number generator. This means that there are $\prod_{i=1}^{n}(n - i + 1) = n!$ possible sequences each with probability $1/n!$. On the other hand, for every index $i = 1 \ldots n$ in the modified algorithm, there are n possible values returned by the random number generator. This means that there are $\prod_{i=1}^{n}(n) = n^n$ possible sequences each with probability $1/n^n$. Given that there are only $n!$ distinct permutations and n^n is not divisible by $n!$, then it is clear that some permutations will appear more frequently than others as a result of every possible sequence. Consequently, the modified algorithm does **not** produce a uniform random permutation.

Good Luck

Dr. Islam ElShaarawy