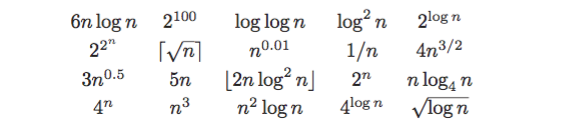
# CS-600-A Homework 1

Zijing Huang 10414952

**R－1.7 Order the following list of functions by the big-Oh notation. Group together (for example, by underlining) those functions that are big-Theta of one another.**



***Hint:* When in doubt about two functions f (n) and g(n), consider log f (n) and log g(n) or 2f(n) and 2g(n).**

The order from lower to higher is as follow:

1/n < 2100 < loglogn < (logn)1/2 < log2 n < n0.01 < ⌈√n⌉ < 3n0:5 < 2logn < 5n < nlog4 n < 6nlogn < ⌊2nlog2 n⌋ < 4n3/2 < 4logn < n2logn < n3 < 2n < 4n < 22^n

**R-1.9 Bill has an algorithm, find2D, to find an element x in an n × n array A. The algorithm find2D iterates over the rows of A and calls the algorithm arrayFind, of Algorithm 1.12, on each one, until x is found or it has searched all rows of A. What is the worst-case running time of find2D in terms of n? Is this a linear-time algorithm? Why or why not?**

The worst case is O(n2) while the target is in last element. It is quadratic instead of a linear-time algorithm. Because arrayFind is called n times, and each time it would traverse n elements until target is found.

**R-1.22 Show that n is o(nlogn).**

We say that n is o(nlogn) if for any constant c > 0 there is any constant n0 >= 0, such that n < c\*nlogn for n >= n0. So 1/c < logn, we choose n0 = 21/c + 1 (when log is the base of 2).

**R-1.23 Show that n2 is w(n)**

To show thatn2logn is w(n), let c > 0 be any constant, there is a constant n0 >= 0 such that n2 > cn. So n > c. We can choose n0 = c + 1.

**R-1.24 Show that n3logn is Ω(n3)**

To prove the expression above, we need to find a constant c > 0 and constant n0 >= 1, such that n3logn >= cn3. We can choose c = 1 and n0 = 2(suppose log is the base of 2).

**R-1.32 Suppose we have a set of n balls and we choose each one independently with probability 1/n1/2 to go into a basket. Derive an upper bound on the probability that there are more than 3n1/2 balls in the basket.**

Base on Chernoff Bounds,

µ = E(X) = n \* (1/n1/2)= n1/2. Then for δ =2, the upper bound is

Pr[X>(1+ δ)u] < (eδ/(1+δ)(1+δ))u => Pr(X > 3µ) <

**C-1.4 What is the total running time of counting from 1 to n in binary if the time needed to add 1 to the current number i is proportional to the number of bits in the binary expansion of i that must change in going from i to i + 1?**

Let t be the time to change every single bit and let k be the total bits. The total work is:

t \* (n/20 + n/21 + n/22… + n/2k) < t \* n \* 2 => O(n)

**C-1.7 Consider the following recurrence equation, defining a function T(n):**

**T(n) = 1 if n = 0**

**T(n) = 2T (n − 1) otherwise,**

**Show, by induction, that T(n) = 2n.**

For T(n):

T(n) = 2 \* T(n-1) = 2 \* 2 \* T(n-2) = … = 2 \* 2n-1 \* T(0) = 2n

**C-1.22 Show that the summation  is O(n). You may assume that n is a power of 2.**

***Hint:* Use induction to reduce the problem to that for n/2.**

⌈log(n/i)⌉ <=  1+log(n/i) < ∫(1+log n -logx) dx = n+ nlogn – n(logn-1) = 2n => O(n).

**C-1.30 Consider an implementation of the extendable table, but instead of copying the elements of the table into an array of double the size (that is, from N to 2N) when its capacity is reached, we copy the elements into an array with additional cells, going from capacity N to N + . Show that performing a sequence of n add operations (that is, insertions at the end) runs in Θ(n3/2) time in this case.**

The size of the array is expanded from N to N + ⌈N1/2⌉

Base on the amortization, each insertion will cost (N+ N1/2)/ N1/2 = 1+ N1/2.

So, total insertion cost is as follow:

∑ 1+ 1+SQRT(N) = ∑ 2+SQRT(N)= 2n + ∑ SQRT(N) from N=1 to N=n

that we can get it is no more than:

(2/3)n3/2+ (1/2)n1/2- 1/6 but no less than

(2/3)n3/2+ (1/2)n1/2+ 1/3 - (1/2)21/2.

So the total cost of the array operation is θ(n3/2).

**A-1.8 Given an array, A, describe an efficient algorithm for reversing A. For example, If A=[3,4,1,5],then its reversal is A=[5,1,4,3]. You can only use O(1) memory in addition to that used by A itself. What is the running time of your algorithm?**

1. We should have two pointers, one points to the beginning of the array, the other points to the end of it.
2. As long as the pointers strictly not pass, swap the values the two pointers point to. And then move lower pointer one higher as well as move the higher pointer one lower.
3. Repeat 2 step.
4. Finish reversing.

The running time would be O(n) because it access every element in the array the one time. And no extra space needed, it only needs two variables to record the pointers. So, the space complexity is O(1).

**A-1.15 Given an integer k > 0 and an array, A, of n bits, describe an efficient algorithm for finding the shortest subarray of A that contains k 1’s. What is the running time of your method?**

1. I set up two points, pointer i point to the first place, pointer j point to the second place of the array.
2. Move j higher until in the range j – i, there are k 1’s in it. If j reach the end of array, then return -1. Otherwise, record j – i as the current shortest length L’.
3. Move j higher, if the current element equals one, move i higher until i point to a 1.

Compare j – i and current L’, record the smaller value to L’.

1. If j reach the end of the length of array, return.

The running time is O(n) because i traverse once as well as j traverse the array once.