Analysis of Algorithms - Homework 1

Kylie Gorman

September 16, 2014

Problem 1 *1-5*

a) Put the elements of S in the knapsack in left to right order if they fit, i.e. the first fit algorithm.
Counterexample: S = $\{1, 2, 5, 9, 10\}$; T = 22
1 + 2 + 5 + 9 = 17 and if 10 is added 27 >22.

b) Put the elements of S in the knapsack from smallest to largest, i.e. the best-fit algorithm.
Counterexample: S = $\{1, 2, 3, 4, 5\}$; T = 5
1 + 2 = 3 and if 3 is added 6 >5.

c) Put the elements of S in the knapsack from largest to smallest.
Counterexample: S = $\{5, 4, 3, 2, 1\}$; T = 8
5 + 4 = 9 >8

Problem 2 *1-6*

U = $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$
S1 = $\{1, 3, 5, 7, 9\}$
S2 = $\{1, 2, 3, 4\}$
S3 = $\{5, 6, 7\}$
S4 = $\{8, 9, 10\}$

If the algorithm is followed correctly, S1 should be the first subset selected because it has the greatest number of uncovered elements. S2 or S4 would be selected after, but S3 would still be needed to cover the universal set. This algorithm is incorrect because only S2, S3, and S4 are needed to cover all of the elements.

Problem 3 *1-16*

Prove by induction that n^3 + 2n is divisible by 3 for all n $\geq$ 0.
Base Case:
n = 0
$(0)^3$ + 2(0) = 0 = 3(0)
n = 1
$(1)^3$ + 2(1) = 3 = 3(1)
n = 2
$(2)^3$ + 2(2) = 6 = 3(2)

Induction Hypothesis:
Assume P(n), therefore n^3 + 2n = 3k, k $\in$ N

Induction Step:
P(n+1) = $(n+1)^3$ + 2(n + 1) = 3(k + 1), k $\in$ N
= n^3 + $3n^2$ + 3n + 1 + 2n + 2
= (n^3 + 2n) + 3 (n^2 + n + 1)
Use induction hypothesis:
According to the induction hypothesis, n^3 + 2n is divisible by 3 and so is the other half of the equation because 3 (n^2 + n + 1)/3 = (n^2 + n + 1). Therefore the entire equation is divisible by 3.

Problem 4 *1-25*

a) Algorithm takes time proportional to n^2

Since it takes 1 second to sort 1,000 items, it will take 10 times as long to sort 10,000 items. Therefore, it will take 10^2 or 100 seconds.

b) Algorithm takes time proportional to nlogn

The same logic can be used from part a. The algorithm will take 10 times as long, so with 10,000 items the time will be 10log10 or about 33.22 seconds.

Problem 5 *2-8*

a) f(n) = $logn^2$; g(n) = logn + 5: f(n) = Θ(g(n))
logn^2 can be simplified to 2logn, making both equations have an equal most significant term and thus, grow at the same rate.

b) f(n) = $\sqrt{n}$; g(n) = logn^2: f(n) = Ω(g(n))
$\lim_{n\to\infty} \frac{\sqrt{n}}{logn^2} = 2 * \lim_{n\to\infty} \frac{\sqrt{n}}{logn} = \infty$
Although logn^2 can be simplified to 2logn, g(n) still grows significantly slower than

f(n), making it a lower bound.

c) f(n) = $log^2 n$; g(n) = logn: f(n) = Ω(g(n))
$\lim_{n\to\infty} \frac{log^2 n}{logn}$ = $\lim_{n\to\infty}$ logn = ∞
$log^2 n$ will always grow at a faster rate than logn, making logn the lower bound.

d)f(n) = n; g(n) = $log^2 n$: f(n) = Ω(g(n))
$\lim_{n\to\infty} \frac{n}{log^2 n} = \infty$
n will always grow faster than $log^2 n$.

e) f(n) = nlogn + n; g(n) = logn: f(n) = Ω(g(n))
$\lim_{n\to\infty} \frac{nlogn+n}{logn} = \infty$
Since nlogn is multiplied by n, it will grow faster than logn alone.

f) f(n) = 10; g(n) = log10: f(n) = Θ(g(n))
10 and log10 are both constants and therefore grow at the same rate.

g) f(n) = 2^n; g(n) = $10n^2$: f(n) = Ω(g(n))
$\lim_{n\to\infty} \frac{2^n}{10n^2} = \infty$
When looking at both of the functions asymptotic growth, 2^n will eventually dominate $10n^2$.

h) f(n) = 2^n; g(n) = 3^n: f(n) = O(g(n))
$\lim_{n\to\infty} \frac{2^n}{3^n} = 0$
Therefore, 3^n dominates 2^n and makes g(n) an upper bound of f(n).

Problem 6 *2-23*

a) Yes, O(n^2) refers to a worst case time and upper bound, so the lower bound is still unknown. Therefore, some inputs can be O(n) since n < n^2.

b) Yes, O(n^2) refers to a worst case time and it is possible, but not probable, that all inputs are O(n).

c) Yes, Θ(n^2) means that the algorithm takes n^2 for the worst case time only, so some inputs can take O(n).

d) No, Θ(n^2) means an algorithms worst case time has to follow n^2 so all inputs cannot take O(n).

e) Yes, the function will always take Θ(n^2) because $100n^2$ and $20n^2$ $nlog_2 n$ both have a highest power of 2.

Problem 7 *2-42*

Algorithm in existence: O(nlog($\sqrt{n}$)) Lower bound for sorting: Ω(nlogn)

This is possible because:
nlog($\sqrt{n}$) = nlog($n^{1/2}$) = 1/2nlogn which grows at the same rate of nlogn.

Problem 8 *2-46*

a) If you are provided with an infinite supply of marbles, go halfway up the building and drop a marble. If the marble breaks, rule out the top half of the building and if it doesnt break, rule out the bottom half. Continue the process similar to a binary search which will take logn time (O(logn)) and will require 7 marbles.

b) If you make the assumption that you can retrieve your marbles and if you are only given 2 marbles, we will require a process more similar to a linear search. Start from the bottom floor and drop the marble continuing up one floor at a time until the marble breaks.

Problem 9 *2-47*

Put one coin in the first bag, 2 coins in the second bag, 3 coins in the third bag, etc. until you end up with 10 coins in the tenth bag. Weigh all of the bags at once. If every bag was real gold, it should weigh 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 = 55 * 10 = 550g. The weight will be altered depending on which bag was fake. For example, if the first bag was fake, the weight would be off by 1 g and 4 g if it was the fourth bag.

Problem 10 *2-48*

Put 3 balls on each side of the balance. If the scales are even, this means that all 6 balls weigh the same amount and the heavier ball is one of the remaining two balls. Put each ball on one side of the scale and the heavier side contains the heavier ball. If the original 6 balls were tilted to one side that means that the heavier ball is one of the three balls on the heavier side. Pick 2 of the three remaining balls at random and put one on each side of the balance. If the scale is tilted to one side, that side contains the heavier ball. If the balance is even, the ball that was not weighed was the heavier ball. You never need more than 2 weighings for any scenario.

Problem 11 *2-49*

All of the companies can be thought of as equal entities, meaning that order does not matter and we are dealing with a combination and not a permutation. A company

cannot be chosen twice to merge with two different companies in one stage, so we also do not have to worry about repetition. Therefore, we are dealing with the equation $\frac{n!(n-r)!}{r!}$. The n represents the number of elements and the r represents how many elements are chosen.
Each stage will reduce by 1 element. For example, if you began with 4 elements, once 2 merge, you will be left with 3 elements. This process will repeat unit you reach one single company. Since you are only merging 2 companies at a time, the value of r will always be 2. Therefore, to determine the number of possible merges of n companies, you must multiply the combination equation from n, n-1, n-2, , 2.

Problem 12 *2-51*

Work from the final 2 people backward. When 2 people are left, the fifth person to speak will get all of the money because he wont need any votes. Therefore, when 3 people are left, the 6th person to speak would be happy with just a dollar. So the 4th person to speak can give the 5th person no money because their vote is not needed and give the 6th person a dollar and keep $299 for himself. The 3rd person to speak knows that the 5th person does not want the situation to go down to 3 people, because he will not get any money. Therefore, the 3rd person will give the 5th person $1 and nothing to anyone else because their votes are not needed. The second person to speak knows that the 4th and 6th people wont get any money if the situation reduces to 4 people, so the 2nd person will offer them each a dollar and offer the 3rd and 5th people nothing, leaving $298 for the second person. Finally, the 1st person to speak follows the same pattern, offering the 3rd and 5th pirates $1 and the 6th, 4th, and 2nd pirates no money since their votes do not matter, leaving $298 for the first pirate and leaving all of the pirates alive.