Algorithms and Complexity: Homework #2

Due on February 8, 2019 at 3:10pm

Professor Bradford

Patrik Sokolowski

Problem 1

Page 67, Exercise 1.
Suppose you have algorithms with the five running times listed below. (Assume these are the exact running times). How much slower do each of these algorithms get when you (a) Double the input, or (b) Increase the input size by 1?

n^2
$(a)\ DOUBLING\ INPUT\ SIZE$
$given\ running\ time = n^2$
$doubling\ input = (2*n)^2 = 4*n^2$
$gets\ 4\ times\ slower$

$(b)\ INCREASE\ INPUT\ SIZE\ BY\ 1$
$adding\ 1\ to\ input = (n+1)^2 = (n+1)(n+1) = n^2 + 2n + 1$
$gets\ slower\ by\ an\ additional\ 2n + 1$
$---$

n^3
$(a)\ DOUBLING\ INPUT\ SIZE$
$given\ running\ time = n^3$
$doubling\ input = (2*n)^3 = 8*n^3$
$gets\ 8\ times\ slower$

$(b)\ INCREASE\ INPUT\ SIZE\ BY\ 1$
$adding\ 1\ to\ input = (n+1)^3 = (n^2+2n+1)(n+1) = n^3 + 3n^2 + 3n + 1$
$gets\ slower\ by\ an\ additional\ 3n^2 + 3n + 1$
$---$

100n^2
$(a)\ DOUBLING\ INPUT\ SIZE$
$given\ running\ time = 100n^2$
$doubling\ input\ = 100(2*n)^2 = 4*100n^2$
$gets\ 4\ times\ slower$

$(b)\ INCREASE\ INPUT\ SIZE\ BY\ 1$
$adding\ 1\ to\ input = 100(n+1)^2 = 100(n^2+2n+1) = 100n^2 + 200n + 100$
$gets\ slower\ by\ an\ additional\ 200n + 100$
$---$

nlogn
(a) DOUBLING INPUT SIZE
given running time = 100n^2
$doubling\ input\ = (2*n)log(2*n)$
$gets\ 2\ times\ slower$

Problem 1 continued on next page. . .

(b) $INCREASE\ INPUT\ SIZE\ BY\ 1$
$doubling\ input = (n+1)log(n+1) = log(n+1)^{n+1} = log(n+1)^n * (n+1) = log(n+1)^n * (n+1)*n^n \frac{}{n^n}$ $=$
after some more work = nlogn + log(n+1) + n[log(n+1) - logn]
gets slower by an additional log(n+1) + n[log(n+1) - logn]

2^n
(a) $DOUBLING\ INPUT\ SIZE$
$given\ running\ time = 2^n$
$doubling\ input = 2^{2*n} = (2n)^2$
$gets\ 2\ times\ slower,\ (doubles)$

(b) $INCREASE\ INPUT\ SIZE\ BY\ 1$
$adding\ 1\ to\ input = 2^{n+1} = (2^n) * 2$
$gets\ slower\ by\ an\ additional\ itself\ (doubles)$
$---$

Problem 2

Page 67, Exercise 2.

Your computer can perform 10^{10} *operations per second. For each of the algorithm's what is the largest input size n f*

$Total\ number\ of\ operations\ in\ an\ hour:\ 10^{10}\ operations/sec\ *\ 60sec/min\ *\ 60min/hour = 3.6 * 10^{13}$

$a)n^2\ largest\ input\ size\ n^2 = 3.6 * 10^{13}\ ,\ n = 6,000,000\ operations\ (\sqrt{3.5 * 10^{13}})$
$b)n^3\ largest\ input\ size\ n^3 = 3.6 * 10^{13}\ ,\ n = 33,019\ operations\ (\sqrt[3]{3.5 * 10^{13}})$
$c)100n^2\ largest\ input\ size\ 100n^2 = 3.6 * 10^{13}\ ,\ n = 600,000\ operations$
$d)nlogn\ largest\ input\ size\ nlogn = 3.6 * 10^{13}\ ,\ n = 1.29 * 10^{12}\ operations$
$e)2^n\ largest\ input\ size\ 2^n = 3.6 * 10^{13}\ ,\ n = 45\ operations$
$f)2^{2^n}\ largest\ input\ size\ 2^{2^n} = 3.6 * 10^{13}\ ,\ n = 5\ operations$

Problem 3

Page 67, Exercise 3.

Arrange these algorithms in ascending order of growth rate.

$\mathrm{f}_1(n) = n^{2.5}$ $\quad exponential,\ higher\ than\ f_2\ and\ f_3$
$f_2(n) = \sqrt{2n}$ $\quad lowest\ degree$
$f_3(n) = n + 10$ $\quad higher\ degree\ of\ 1\ than\ f_2$
$f_4(n) = 10^n$ $\quad highest\ degree/most\ exponential$
$f_5(n) = 100^n$ $\quad highest\ degree/most\ exponential$
$f_6(n) = n^2 logn$ $\quad exponential,\ higher\ than\ f_2\ and\ f_3$

$f_2 < f_3 < f_6 < f_1 < f_4 < f_5$

Problem 4

Page 68, Exercise 6.
Given an Array A consisting of n intergers A[1], A[2], . . . , A[n]. You'd like to output a 2 dimensional n by n array B in which B[i,j](for i¡j) continues the sum of array entries A[i] through A[j].

Given Algorithm:

```
.    For i = 1, 2, ... n
.         For j = i+1, i+2, ..., n
.              Add up array entries A[i] through A[j]
.              Store the result in B[i,j]
.         End for
.    End for
```

A) For some function f that you should choose, give a bound of the form O(f(n)) on the running time of this algorithm on an input of size n (i,e, a bound on the number of operations performed by the algorithm).

O(n^3)
$Outer for - loop\ goes\ on\ n\ times\ O(n)$
$Inner\ for - loop\ goes\ n\ times\ for\ the\ outer\ loop\ (another O(n))$
$adding\ up\ the\ array\ entries\ is\ O(n)\ times,\ inside\ the\ loops$
$resulting\ in\ O(n^3)$

B) For the same function f, show that the running time of the algorithm an on input size n is also asymptotically tight bound of O(f(n)) on the running time.

Outer loop goes through n times
Iterations od inner loop: (n-1) + (n+1) + ... 1 + 0 = (1/2)(n-1)((n+1)-1) = (1/2)$n^2 - (1/2)n$
$Number\ of\ operations\ on\ adding\ for\ n\ iterations\ of\ inner\ loop:$
$For i = 1 : 1 + 2 + ... + n - 1 = (1/2)(n - 1)(n) = 0.5n^2 - (1/2)n$
$For i = 2 : 1 + 2 + ... + n - 2 = (1/2)(n - 2)(n - 1) = 0.5n^2 - (1/2)n$
$For i = k : 1 + 2 + ... + n - k = (1/2)(n - k)(n(k + 1))$
$so for i = n - 1 : 1 + (n - (n - 1)) = 1 + 2 = n^2/8$
$There\ are\ at\ least\ n^3/16\ addition\ operations\ for\ all\ n = 2$
$Algorithm\ is\ lower\ bounded\ by\ n^3$

C) Give a different algorithm to solve the problem, with an assymptotically better running time.
Algorithm with (O(n^2))$instead\ of\ O(n^3)) :$
$indexedSum = 0;$

. $For i = 1, 2, ...n$
. $indexedSum = A[i];$
. $For j = i + 1, i + 2, ...n$
. $indexedSum+ = A[j]$
. $Store\ indexedSum\ in\ B[i, j]$
. $End\ for$
. $End\ for$