COM S 311

Homework 2

Recitation 5, 1-2pm, Marios Tsekitsidis

Christian Shinkle

February 4, 2018

1. (a) For the worst-case run time, r =
$= \sum_{i=1}^{n} \sum_{j=i}^{n} \sum_{k=1}^{j-i} c$
$= \sum_{i=1}^{n} \sum_{j=i}^{n} (j-i)c$
$= \sum_{i=1}^{n} c \sum_{j=i}^{n} (j-i)$ by property of summations
$= \sum_{i=1}^{n} c(\sum_{j=i}^{n} j - \sum_{j=i}^{n} i)$ by property of summations
$= \sum_{i=1}^{n} c(\frac{(n+i)(n-i+1)}{2} - (i(n-i)+1))$
$= c\sum_{i=1}^{n} \frac{n^2+n-i^2+i}{2} - (in - i^2 + 1)$
$= c\sum_{i=1}^{n} \frac{n^2+n-i^2+i}{2} - in + i^2 - 1$
$= c\sum_{i=1}^{n} \frac{n^2+n-i^2+i-2in+2i^2-2}{2}$
$= c/2\sum_{i=1}^{n} n^2 + n + i^2 + i - 2in - 2$
$= c/2(\sum_{i=1}^{n} n^2 + \sum_{i=1}^{n} n + \sum_{i=1}^{n} i^2 + \sum_{i=1}^{n} i - \sum_{i=1}^{n} 2in - \sum_{i=1}^{n} 2)$
$= c/2(n^3 + n^2 + (2n^3 + 3n^2 + n)/6 + (n^2 + n)/2 - n^3 - n^2 - 2n)$
$= c/2(\frac{1}{3}n^3 + n^2 - \frac{4}{3}n)$
Therefore, $c/2(\frac{1}{3}n^3 + n^2 - \frac{4}{3}n) \in O(n^3)$.

   (b) For the worst-case run time,
inner loop $= \sum_{j=1}^{i} c = i * c$
For the outer loop, note after k iterations, $(((i/2)/2)/2.../2) = 1$. So, $i/2^k = 1$ which implies $k = \log_2 i$. So, the outer loop $= \log(ic) \in O(\log i)$.
Because $i = n$ at the start of the outer loop, the worst-case run time is $O(\log n)$.

2. (a) For the worst-case run time, it is $\sum_{i=1}^{n-1} \sum_{j=1}^{i} c$ because the for-loop will always run $n-1$ times and the while-loop will have to run from $j = i$ until $j = 1$ because the maximum number of times it will run is if the array is reversed and when the array is reversed, $a[j-1]$ will always be greater than $a[j]$ so the while-loop will only terminate when $j = 0$. Also, note that swap and decrementing j both run in constant time.
Therefore, $\sum_{i=1}^{n-1} \sum_{j=1}^{i} c =$
$c\sum_{i=1}^{n-1}(i+1)$
$\approx c\sum_{i=1}^{n-1} i$
$= c((n-1)n)$
$= c(n^2 - n) \in O(n^2)$

   For best-case run time, it is $\sum_{i=1}^{n-1} c$ because the for-loop will always run $n-1$ times and the while-loop will never run because $a[j-1]$ will ways be less than $a[j]$ so the algorithm will never enter it. Note, this means $j = 1$ and checking $a[j-1] > a[j]$ will run in constant time. Therefore, $\sum_{i=1}^{n-1} c$ $= c(n-1) \in O(n)$

   (b) The run times of the algorithms are as follows:

Sorted Arrays

	3000	30000	300000
Selection	6	84	8069
Bubble	0	0	2
Insertion	0	1	2

Reverse Sorted Arrays

	3000	30000	300000
Selection	12	1098	108541
Bubble	14	432	455
Insertion	10	455	44862

Randomized Arrays

	3000	30000	300000
Selection	11	1115	109894
Bubble	8	1167	119419
Insertion	2	231	23291

3.

- Direct Proof: For $48n^4 - 46n^2 + 25n + 31 \in O(n^4)$,
  $\exists c$ s.t. $\forall n, 48n^4 - 46n^2 + 25n + 31 \leq cn^4$.
  Note, $48n^4 - 46n^2 + 25n + 31$
  $\leq 48n^4 + 46n^4 + 25n^4 + 31n^4$
  $= 150n^4$
  Therefore, $48n^4 - 46n^2 + 25n + 31 \leq cn^4$ for $c = 150$, so
  $48n^4 - 46n^2 + 25n + 31 \in O(n^4)$

- Direct Proof: For $n^{\log n} \in O(2^{\sqrt{n}})$, $\exists c$ s.t. $\forall n, n^{\log n} \leq c2^{\sqrt{n}}$.
  Using the fact, $\forall k, a > 0, \log^k n \in O(n^a)$, this shows
  $\log^2 n \in O(\sqrt{n})$ which means $\exists c_1$ s.t. $\log^2 n \leq c_1\sqrt{n}$.
  Then, $\log^2 n \leq c_1\sqrt{n}$
  $= 2^{\log^2 n} \leq 2^{c_1\sqrt{n}}$
  $= n^{\log n} \leq 2^{c_1\sqrt{n}}$
  Therefore, let $c = 2^{c_1}$. This shows $\exists c$ s.t. $\forall n, n^{\log n} \leq c2^{\sqrt{n}}$. Therefore, $n^{\log n} \in O(2^{\sqrt{n}})$.

- Proof by Contradiction: Assume $2^{2^{n+1}} \in O(2^{2^n})$.
  Then, $2^{2^{n+1}} \leq c2^{2^n}$. Then,
  $2^{2^{n+1}}$
  $= 2^{2(2^n)}$
  $= 2^{2^n+2^n}$
  $= 2^{2^n} * 2^{2^n}$
  So, $2^{2^n} * 2^{2^n} \leq c2^{2^n}$
  $2^{2^n} \leq c$
  This is a contradiction because c cannot outgrow a function of n. Therefore, $2^{2^{n+1}} \notin O(2^{2^n})$.

- Proof by Contradiction: Assume $n^3(5 + \sqrt{n}) \in O(n^3)$. Then,
  $n^3(5 + \sqrt{n}) \leq cn^3$
  $5+\sqrt{n} \leq c$. This is a contradiction because c cannot outgrow a function of n. Therefore, $n^3(5+\sqrt{n}) \notin O(n^3)$.

4. (a) The algorithm for calculating change from base k to 10 is the following:
   $n =$ number of digits in base-k number
   $result = 0$
   for i in the range $[0, n-1]$
       $result = result + pow(k, i) * digits[n-i-1]$
   return $result$

The algorithm for calculating *pow* is the following:
Give a, i,
$x = a$ $m = i$ $y = 1$
while $m > 1$

    if (m is odd)
        $y = x * y$
        $m = (m - 1)/2$
    else if m is even
        $m = m/2$
    $x = x * x$

return $(x * y)$
For the run time analysis, we start with run time for *pow*:
Let k be the number of iterations through the while-loop.
Then, $4k + 3$ is the number of operation. Note $k = \log i$.
Therefore, *pow* runs in O($\log i$) where i is the exponent of the power.
Using this information, the run time of the full algorithm is:
$\sum_{i=0}^{n-1} *\mathrm{O}(\log i)$
$= n * \log i \in O(n \log n)$

(b) The algorithm of calculating base k number from decimal number m:
Inputes: k, m where k is the base of the number being calculated and m is the decimal number being converted from.
Let j = number of digits of the output number and let *output* be an array who's indices represent the digits of the number returned by the algorithm from the most significant bit to the least significant bit.
for i in range $[0, j - 1]$

    $output[j - i - 1] = m \mod k$
    $m = m/k$

return *output*
Note $j = 1 + \log m$. This means the run time of this algorithm is:
$\sum_{i=0}^{\log m} c = \log(m)c \in O(\log m)$