Homework 1 Solutions
Com S 311

1. **Big O** (50 points) This problem is on determining whether one function is Big-O of the other. Provide a proof for your answers. If you are showing that if $f \in O(g)$, then you must show that there is a constant $c \geq 1$ such that for every $n \geq 1$, $f(n) \leq cg(n)$. Your proof must state the value of the constant c, and, if your inequality only holds for values of n that are above some threshold N, then you must provide the value of N as well.

   If you are showing that $f \notin O(g)$, then you must show that for every constant $c \geq 1$ and every threshold $N > 0$, there is always an $n > N$ for which $f(n) \geq cg(n)$. (Alternatively, you may provide a proof by contradiction.)

   In your proofs, you may use the fact that for every $k > 0$ and $a > 0$, $n^k \in O(n^{k+a})$, and $\log^k n \in O(n^a)$.

   (a) Is $8n^2 + 35n + 46 \in O(n^2)$?

   Ans. We will prove that there is a constant c such that $8n^2 + 35n + 46 \leq cn^2$

   $$\begin{aligned} 8n^2 + 35n + 46 &\leq 8n^2 + 35n^2 + 46n^2 \\ &= 89n^2 \end{aligned}$$

   Thus $8n^2 + 35n + 46 \leq 89n^2$ for every natural number. This proves that $8n^2 + 35n + 46 \in O(89n^2)$.

   (b) Is $2^{2^{n+1}} \in O(2^{2^n})$?

   Ans. No. We will prove by contradiction. Assume that $2^{2^{n+1}} \in O(2^{2^n})$. Thus there is a constant c such that for every natural number,

   $$2^{2^{n+1}} \leq c \times 2^{2^n}$$

   By taking $\log_2$ on both the sides we obtain that

   $$\begin{aligned} 2^{n+1} &\leq \log c + 2^n \\ \Rightarrow \quad 2^n &\leq \log c \end{aligned}$$

   However, 2^n is a growing function and $\log c$ is a constant. Thus 2^n can not be less than $\log c$ for every natural number. This is a contradiction. Thus $2^{2^{n+1}} \notin O(2^{2^n})$

   (c) Is $n^3(5 + \sqrt{n}) \in O(n^3)$?

Ans. No. We will prove by contradiction. Assume that $n^3(5+\sqrt{n}) \in O(n^3)$ Thus, there is a constant $c > 0$ such that

$$\begin{aligned} \forall n, n^3(5+\sqrt{n}) &\leq cn^3 \\ &\Rightarrow \forall n, 5+\sqrt{n} \leq c \text{by dividing both sides by } n^3 \end{aligned}$$

However, $5+\sqrt{n}$ is a growing function and c is a constant. Thus $5+\sqrt{n}$ can not be less than c for every natural number. This is a contradiction.

(d) Prove that for any number $a \geq 0$, $2^{n+a} \in O(2^n)$
Ans. Take $c = 2^a$. Since $2^{n+a} = 2^a \times 2^n$, we have that $2^{n+a} \in O(2^n)$ for every $a \geq 0$.

(e) Prove that if $f \in O(g)$ and h is any positive-valued function, then $fh \in O(gh)$.
Ans. Note that fh stands for $f \times h$. Since $f \in O(g)$, there is a constant c such that for every natural number

$$f(n) \leq c \times g(n)$$

Multiply both sides by $h(n)$ to obtain

$$f(n)h(n) \leq c \times g(n)h(n)$$

Thus $fh \in O(gh)$

2. **Analyze Runtime** (40 points) Formally derive the run time of the each algorithm below as a function of n and determine its Big-O upper bound. You must show the derivation of the end result; if all you do is state the Big-O bound, you'll receive zero points.

(a)
```
for i in the range [1, n]
   for j in the range [i, n]       // note this starts at i, not at 1
     Constant Number of Operations
```
Ans. Solution not provided. Please see TA/instructor.

(b)
```
for i in the range [1, n]
   for j in the range [1, n*i]
     Constant Number of Operations
```
Ans. Each iteration of the inner loop takes constant time, and the inner loop runs while j ranges over $[1, ni]$. Thus the time for inner loop is

$$\sum_{j=1}^{ni} c = cni$$

During each iteration of the outer loop, the inner loop is run once. Thus the time take by each iteration of the outer loop is

$$d + cni$$

where d is a constant. Thus the total time os

$$\sum_{i=1}^{n} [d + cni]$$

$$\begin{aligned}\sum_{i=1}^{n}[d+cni] &= \sum_{i=1}^{n} d + \sum_{i=1}^{n} cni \\ &= dn + cn\sum_{i=1}^{n} i \\ &= dn + cn^2(n+1)/2 \\ &= O(n^3)\end{aligned}$$

(c)
```
x = pow(2, n)
i = 1
while i <= x
  for j in range [1, i]
     Constant Number of Operations
  i = i * 2
```

Ans. Each iteration of the inner loop take time c which is a constant. Let c_1 be the constant associated with the inner loop and c_2 be the constant associated with the outer loop, and let $c = \max\{c_1, c_2\}$. Time for inner loop is atmost

$$\sum_{j=1}^{i} c$$

During each iteration of the outer loop, the inner loop is run once. Thus the time take by each iteration of the outer loop is

$$d + \sum_{j=1}^{i} c$$

where d is a constant. For Outer loop, the index variable ranges over $[1, 2, 4, 8, \cdots, 2^n]$. Thus the time taken by the algorithm is at most

$$\sum_{i\in\{1,2,4,\cdots,2^n\}} [d + \sum_{j=1}^{i} c]$$

which equals

$$\sum_{i\in\{1,2,4,\cdots,2^n\}} d + c \sum_{i\in\{1,2,4,\cdots,2^n\}} i$$

Note that the number of items in the set $\{1, 2, 4, \cdots, 2^n\}$ is $n+1$. Thus

$$\sum_{i\in\{1,2,4,\cdots,2^n\}} d = d(n+1)$$

Note that

$$c \sum_{i\in\{1,2,4,\cdots,2^n\}} i = c(1+2+4+8+16+\cdots+2^n)$$

This is a geometric progression with common ratio 2. The above sum equals $c(2^{n+1}-1)$. Thus the total time taken by the algorithm is $d(n+1) + c(2^{n+1}-1)$ which is $O(2^n)$.

(d)

```
i = n
while i >= 1
  for j in range [1, i]
     Constant Number of Operations
  i = i / 2
```

Ans. Let c_1 be the constant associated with the inner loop and c_2 be the constant associated with the outerloop, and let $c = \max\{c_1, c_2\}$. Each iteration of the inner loop takes time $\leq c$. Thus the time taken for the inner loop is at most

$$\sum_{j=1}^{i} c = ci$$

For the outerloop, the index variable i ranges over $\{n, n/2, n/4, \cdots 1\}$. Thus the time take by the algorithm is at most

$$\sum_{i \in \{n, n/2, n/4 \cdots, 1} ic$$

which equals

$$c \sum_{i \in \{n, n/2, n/4 \cdots, 1} i$$

Thus summation is a geometric progression with a common ratio of $1/2$. Thus thus summation is atmost $2n$. Thus the total time taken is at most $2cn$ which is $O(n)$.

3. **Think about correctness** (20 points) In class we discussed the *Interval Scheduling* problem: given a set I of jobs $[s, e]$, where s is the start time and $e > s$ is the end time, select a subset S of I containing a maximal number of non-overlapping jobs. (If you missed class that day, you can find examples and more discussion of this problem in Section 1.2 of the text by Skeina, available online through the ISU library.) Here is a proposed algorithm to solve this problem:

```
start with an empty set S
while I is nonempty
  find the job j such that the midpoint $(s + e) / 2$ is earliest
  remove j from I
  remove all jobs from I that overlap j
  add I to the solution S
```

Decide whether the algorithm is correct. If you believe it is correct, give an informal explanation for why it works. If you believe it is incorrect, provide a test case on which it fails (counterexample).

Ans. Algorithm is incorrect. Consider the following three intervals. $I_1 = [1, 100]$. $I_2 = [50, 60]$ and $I_3 = [70, 90]$. The algorithm picks$[1, 100]$ whereas I_2 and I_3 is the optimal solution.

4. **Implementation** (40 points) Consider the following methods that compute the greatest common divisor of two integers.

```
gcd(a, b):
   n = min(a, b)
   for (i = n; i >=1; i--)
      if  both a%i and b%i are zero
       return i
```

```
fastgcd (a, b):
   if b equals 0
       return a
   else
       return fastgcd(b, a % b)
```

Write Java methods that implement each of the above algorithms. (Use type `long` rather than `int` for the arguments and return values. Play with the program by giving very large numbers as inputs to both the numbers.

(a) Pick two 9-digit primes and run both methods. Report the run time.

(b) Pick two 10 digit primes and run both methods. Report the run time.

Please do not submit your code; just report your run times. You can use `System.currentTimeMillis()` to calculate run time of your methods.

You can look at `https://primes.utm.edu/lists/small/millions/` for 9 digit primes. You can look at `https://primes.utm.edu/lists/small/small.html` for 10 digit primes.

5. **Design an algorithm and derive runtime** (30 points) Given an array $A = [a_0, a_1, a_2, \cdots, a_{n-1}]$ of integers, let p_A be the following polynomial:

$$a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_2x^2 + a_1x + a_0$$

Give an algorithm that gets an array A and an integer t as inputs and outputs $p_A(t)$ (value of the polynomial at t). Derive the worst-case run-time of your algorithm (as a function of the size of the input array). Your grade depends partly on the runtime of the algorithm. Remember that $Math.pow(\cdot, \cdot)$ is not a primitive operation.

Write your algorithm in clear pseudocode. (Do not submit Java/C code.)

Ans. Consider the following algorithm.

$pow = 1$; $val = 0$.
for i in the range $[1, \cdots, n]$ {
$val = val + pow$;
$pow = t * pow$;
}
Output val

Each iteration of the loop takes $O(1)$ time, thus the the time taken is $O(n)$.

GUIDE LINES:

- Please write your recitation number, time and TA name.
- It is important to know whether your really know! For each problem, if you write the statement "I do not know how to solve this problem" (and nothing else), you will receive 20% credit for that problem. If you do write a solution, then your grade could be anywhere between 0% to 100%. To receive this 20% credit, you must explicitly state that you do not know how to solve the problem and you must not submit any attempted solution.
- You must work on the homework problems on your own. You should write the final solutions alone, without consulting any one. Your writing should demonstrate that you understand the proofs completely.
- When proofs are required, you should make them clear and rigorous.
- Any concerns about grading should be made within one week of returning the homework.
- **Please submit your HW via Canvas. If you type your solutions, then please submit pdf version. If you hand-write your solutions, then please scan your solutions and submit a pdf version. Please make sure that the quality of the scan is good, and your hand writing is legible. Name your file** $YourNetID$**-**$HW1.pdf$**. For example, if your net id is** `bondj`**, then the file name should be** $bondj$**-**$HW1.pdf$**. HW's submitted in incorrect format (non pdf, incorrect file name etc) will incur a penalty of 20%**
- If you hand writing is not legible or the quality of the scan is too poor to read, your homework will not be graded.