Homework 1
Com S 311
Due: Sep 3, 11:59PM
Total Points: 100

Late submission policy. If you submit by Sept 4, 11:59PM, there will be 20% penalty. That is, if your score is x points, then your final score for this homework after the penalty will be $0.8 \times x$. Submission after Sept 4, 11:59PM will not be graded without explicit permission from the instructors.

Submission format. Your submission should be in pdf format. Name your submission file: `<Your-net-id>-311-hw1.pdf`. For instance, if your netid is `asterix`, then your submission file will be named `asterix-311-hw1.pdf`.

If you are planning to wri n is readable (sometimes sca

If you plan to snap pict readable - in most cases, such submissions are difficult to read and mor up submitting parts of your work.

If you would like to type your work, then one of the options you can ex

Organization of the assignment. This homework has two parts. Part 1 of the assignment will be graded explicitly and strictly. That is, for every solution you provide, if it is not correct or partially correct, you will be provided with basic comments outlining the problem with your solution. If you correctly answer all questions in Part 1 and submit within the specified deadline (not late), you will secure 100% points. As noted in the syllabus, any re-grade request must be submitted within one week of the release of the grades.

Part 2 of the assignment will not be graded explicitly. If you write some reasonable answers (as assessed by the teaching staff members) for all the questions (Part 2 has two questions), you can get at most 20 points (10 points/question; there are no partial credits for the answering only parts of any question). This can be counted as extra credits for this assignment. We will not consider requests to re-grade Part 2 of the assignment.

Learning outcomes.

1. Determine whether or not a function is Big-O of another function
2. Analyze asymptotic worst-case time complexity of algorithms

1 Part 1

1. Prove or disprove the following statements. Provide a proof for your answers.

   If you are showing that if $f(n) \in O(g(n))$, then you must show that there are some $c > 1$ and $N > 0$ such that $\forall n \geq N$, $f(n) \leq cg(n)$. You will have to show the range of values of c and N for which the claim is true.

   On the other hand, if you are showing that if $f(n) \notin O(g(n))$, then you must prove why there exists no $c > 1$ or $N > 0$ such that $\forall n \geq N$, $f(n) \leq cg(n)$.

   (30 Points)

   (a) $2n^3 + 35n + 46 \in O(n^3)$

   Proof. The following inequalities hold for every $n \geq 1$

   $$\begin{aligned} 2n^3 + 35n + 46 &\leq 2n^3 + 35n^3 + 46n^3 \\ &= 83n^3 \end{aligned}$$

   We take $c = 83$ and $N = 1$, Thus we have

   $$\forall n \geq 1(2n^3 + 35n + 46) \leq 83n^3$$

   So $2n^3 + 35n + 46 \in O(n^3)$.

   (b) $6n^2 - 41n$

   Ans. The fo

   $$6n^2 - 41n + 2 \leq 6n^2 + 42n^2 + 2n^2$$

   We take $c = 50$ and $N = 1$.

   $$\forall n \geq 1, (6n^2 - 41n + 2) \leq 50n^2$$

   Thus $6n^2 - 41n + 2 \in O(n^2)$

   (c) $\forall a \geq 1 : 2^n \in O(2^{n-a})$

   Ans. Note that $2^n \leq \frac{2^n}{2^a}$.

   We take $c = 2^a$ and $N = 1$, thus

   $$\forall n \geq 1, 2^n \leq 2^a \times 2^{n-a}$$

   thus $2^n \in O(2^{n-a})$ for every $a \geq 1$.

   (d) $\forall a \geq 1 : O(\texttt{log}_2 n) \in O(\texttt{log}_a n))$

   Ans. Note that $\log_2 n = \log_a n \times \log_2 a$ for every a. Thus for a given a, we take $c = \log_2 a$ and $N = 1$, Now

   $$\forall n \geq 1, \texttt{log}_2 n \leq \log_2 a \times \texttt{log}_a n$$

   Thus $\texttt{log}_2 n) \in O(\texttt{log}_a n))$ for every $a \geq 1$.

(e) $\forall a \geq 1 : a^{a^{n+1}} \in O(a^{a^n})$.

Ans. Suppose that there exist c and N such that

$$\forall n \geq N, a^{a^{n+1}} \leq c \times a^{a^n}$$

Divide both sides of the inequality by a^{a^n}, to obtain

$$\forall n \geq N, a^{a^n} \leq c$$

This is a contradictions as a^{a^n} is a growing function and c is a constant. Thus $a^{a^{n+1}} \notin O(a^{a^n})$

(f) If $f(n) \in O(g(n))$ and $h(n)$ is any positive-valued function, then $f(n) \times h(n) \in O(g(n) \times h(n))$.

Ans. Since $f(n) \in O(g(n)$, there exist c and N such that

$$\forall n \geq N f(n) \leq c \times g(n)$$

Since $h(n)$ is a positive valued function we can multiply both sides of the inequality by $h(n)$

$$\forall n \geq N f(n) \times \leq c \times g(n) \times h(n)$$

Thus $f(n)$

2. Formally derive the ru n and determine its Big-O upper bou

final answer without any derivation steps will result in 0 point (30 Points)

(a)
```
for (i = 1; i < n-8; i++) {
    for (j = i; j < i+8; j = j++) {
        <some-constant number of atomic/elementary operations>
}}
```

Ans. Each iteration of the inner loop take c_1 time which is a constant. Time taken by the inner loop is

$$\sum_{j=i}^{i+7} c_1 = 8c_1$$

During every iteration of outerloop, the inner loop is run once and a constant c_2 number of operations are performed. Thus one iteration of outer loop takes time

$$c_2 + 8c_1$$

This the total time is

$$\sum_{i=1}^{n-7} [c_2 + 8c_1]$$

$$\begin{aligned}\sum_{i=1}^{n-7}[c_2+8c_1] &= \sum_{i=1}^{n-7}c_2+\sum_{i=1}^{n-7}8c_1\\ &= (n-7)c_2+8(n-7)c_1\\ &= (8c_1+c_2)n-7c_2-56c_1\\ &\in O(n)\end{aligned}$$

(b)
```
for i in the range [1, n] {
    for j in the range [i, n] {
        for k in the range [1, j-i] {
         <some-constant number of atomic/elementary operations>
}}}
```

Each iteration of inner most loop has constant many operations. Inner loop is performed while k ranges over $[1, j-i]$.

Each iteration of the middle loop performs innermost loop. Middle loop is performed while j ranges over $[i, n]$. Each iteration of the outer loop performs the middle loop. Outer loop is done while i ranges over $[1, n]$. Thus runtime is bounded by

$$\sum_{i=1}^{n}\sum_{j=i}^{n}\sum_{k=1}^{j-i}c$$

$$\begin{aligned}\sum_{i=1}^{n}\sum_{j=i}^{n}\sum_{k=1}^{j-i}c &= \sum_{i=1}^{n}\sum_{j=i}^{n}c(j-i)\\ &= \sum_{i=1}^{n}[\sum_{j=i}^{n}cj-\sum_{j=i}^{n}ci\\ &= \sum_{i=1}^{n}c[\frac{n(n+1)}{2}-\frac{i(i-1)}{2}-ci(n-i+1)]\\ &= \sum_{i=1}^{n}c[\frac{n^2}{2}+\frac{n}{2}-\frac{i^2}{2}+\frac{i}{2}-ni+i^2-i]\\ &= \sum_{i=1}^{n}c[\frac{n^2}{2}+\frac{n}{2}+\frac{i^2}{2}-ni-\frac{i}{2}]\\ &= c[\frac{n^3}{2}+\frac{n^2}{2}+\frac{n(n+1)(2n+1)}{12}-\frac{n^2(n+1)}{2}-\frac{n(n+1)}{4}]\\ &\in O(n^3)\end{aligned}$$

(c)
```
x = pow(2, n);
i = 1;
while i <= x {
  for j in range [1, i] {
```

```
        <some-constant number of atomic/elementary operations>
  }
  i = i * 2
}
```

Assume that `pow(2, n)` is computed *magically* in constant time.

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

$$\sum_{i} d$$

which equal

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

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 >= 2 {
  for j in range [1, i] {
        <some-constant number of atomic/elementary 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 $\{n, n/2, n/4, \cdots 2\}$. Thus the time take by the algorithm is at most

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

which equals

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

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

3. Consider the following two methods that compute the greatest common divisor of two positive integers. (40 Points)


```
GCD_1(a, b) {
   n = min(a, b);  // min
   for (int i = n; i >=1; i--)
      if  both (a % i) and (b % i) are zero
          return i;
}
```



```
GCD_2(a, b) {
     x = max{a, b}
     y = min{a, b}
     while (y != 0) {
       z = x % y;
       x = y;
       y = z;
   }
   return x;
}
```

Write a Java program that implements both of the above methods. Play with the program by giving very large numbers as inputs to both the numbers. You do not need to submit your code. Submit the answers to the following questions.

(a) Pick two 9-digit primes and run both methods. Report the actual execution time. Use `https://primes.utm.edu/lists/small/millions/` for 9 digit primes.

(b) Pick two 10 digit primes and run both methods. Report the actual execution time. Use `https://primes.utm.edu/lists/small/small.html` for 10 digit primes.

You can use `System.currentTimeMillis()` to calculate the execution times.

(c) Derive the runtime of `GCD_1` when inputs a and b are n-bit integers.

Ans. An n bit integer can have a maximum possible value of 2^n. Thus the loop is performed in the worst-case 2^n times. each iteration of the loop takes constant time. Min operation also takes constant time. Thus the time taken is $O(2^n)$.

(d) Show that the run-time of `GCD_2` is $O(n)$ when the inputs a and b are n-bit integers.

Ans. We use the following crucial observation if $u \leq v$, then $u\%v \leq u/2$. This is because is $v > u/2$, then dividing $u\%v = u - v$ and $u - v \leq u/2$ when $v > u/2$. If $v = u/2$, then $u\%v = 0$. If $v < u/2$, then $u\%v < v$ and thus $u\%v \leq u/2$.

Let x_0 and y_0 be value of x and y at the start of an iteration. Let x_1 and y_1 be values of x and y after that iteration and x_2 and y_2 be value of x and y after the iteration after.

Note that $y \leq x$ at the start of the iteration, we have that $z = x_0\%y_0$ and by the above observation, we have $z \leq x_0/2$ and z must be less than y_0. Now x is assigned to y and y is assigned to z. Thus after one iteration $y_1 \leq x_0/2$, $y_1 \leq y_0$ and $x_1 = y_0$. Thus at the start of the next iteration we have $y_1 \leq x_1$. Thus after this iteration we have $y_2 \leq x_1/2 = y_0/2$ and $x_2 = Y_1 \quad x_0/2$. Thus after two iterations, the value of y is halved. Sinc eration the value of y becomes hal $O(n)$.

2 Part 2

These problems will not be graded explicitly. If you write reasonably c by the teaching staff members), you will secure extra 20 points—That is 10 points per problem. To receive 10 points, you must answer all parts of the question, otherwise you will receive 0 points.

1. Prove or disprove the following.

   (a) $\forall a \geq 1 : a^n \in O(n!)$.
   Ans. True

   (b) $n^3(5 + \sqrt{n}) \in O(25n^3)$.
   Ans false

   (c) $2^{2n} \in O(2^n)$.
   Ans. False

   (d) If $f(n) \in O(g(n))$, then $[f(n)]^2 \in O([g(n)]^2)$.
   Ans. True

2. 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.

(a)
```
for i in the range [1, n] {
   for j in the range [1, n/i]
     <some-constant number of atomic/elementary operations>
}
```
Ans. $O(n \log n)$

(b)
```
x = n;
while (x >= 2) {
  x= sqrt{x};
}
```
Assume that `sqrt(x)` takes constant time.

Ans. $O(\log \log n)$.