Quiz #3: Analysis of Algorithms

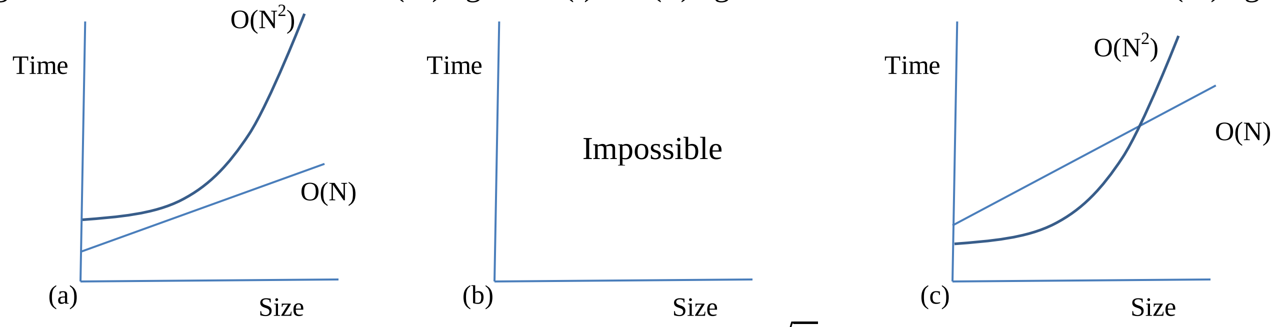
ICS-46 Fall 2016

Name ____________________

Recall the Academic Integrity statement that you signed. Write all answers clearly on these pages, ensuring your final answers are easily recognizable. The number of points for each problem is clearly marked, for a total of 25 points. I will post my solutions on the web on Friday, off the **Solutions** link, after class.

---

1. (3 pts) Sketch approximate Size vs. Time curves for the two algorithmic complexity classes required in each of the pictures below: for one, instead write **Impossible**: (a) an O(N) algorithm that is **always** faster than an $O(N^2)$ algorithm.. (b) an O(N) algorithm that is **never** faster than an $O(N^2)$ algorithm. (c) an O(N) algorithm that is **sometimes** faster than an $O(N^2)$ algorithm.



2. (2 pts) Assume that a sorting function **s** is in the complexity class $\mathbf{O(N\sqrt{N})}$. (a) What is its doubling-signature: how much more time (by what factor) does it take to solve a problem twice as large (compute an actual number)? Show your calculation and simplification. (b) If **s** can sort a 1,000 element array in **.002** seconds; how long will **s** take to sort a 100,000 element array?

(a) Assume **Ts(N) ~ c** $N\sqrt{N}$. Then, **Ts(2N) / Ts(N)** ~ **c** $2N\sqrt{2N}$ / **c** $N\sqrt{N}$ ~ $2\sqrt{2}$ ~ **2.83**

(b) **Ts(100N) / Ts(N)** ~ **c** $100N\sqrt{100N}$ / **c** $N\sqrt{N}$ ~ $100\sqrt{100}$ = **1,000; so .002 x 1,000 = 2 seconds**

3. (2 pts) To work correctly, binary searching requires a precondition that an array is sorted. It seems like this function should check this precondition and throw an exception if it is not met. What is the problem of precondition checking here? Your argument should discuss information about all relevant complexity classes.

Binary search is $\mathbf{O(\log_2 N)}$. Checking the precondition is **O(N)**. If we check the precondition, we have degraded the complexity class to $\mathbf{O(\log_2 N + N) = O(N)}$: we might as well do a linear search, which is **O(N)**.

4. (6 pts) The following two functions each determine whether or not any two values in array **a** (storing **N** values) sum to **v**. As is shown in the notes: (a) Write the complexity class of each statement in the box on its right. (b) Write the full expression that computes the complexity class for the entire function. (c) Simplify what you wrote in (b). Assume a fast **qsort** function and that **search** is therefore searching a sorted array.

```
bool sum_to_v (int a[], int N, int v){

  for (int i=0; i<N; ++i)              O(N)

    for (int j=i; j<N; ++j)            O(N)

      if (a[i]+a[j] == v)              O(1)

        return true;                   O(1)

  return false;                        O(1)
```

(b) `O(N)*O(N)*O(1) + O(1)`

(c) $O(N^2)$

```
bool alt_sum_to_v (int a[], int N, int v){

 qsort(a,N, ...)                       O(NLogN)

 for (int i=0; i<N; ++i)               O(N)

   if (search(a,N,v-a[i]))             O(Log N)

     return true;                      O(1)

 return false;                         O(1)
```

(b) `O(NLogN) + O(N)*O(LogN) + O(1) = O(2NLogN + 1)`

(c) `O(N Log N)`

5. (4 pts) Assume that functions **f1** and **f2** compute the same result by processing the same argument. Empirically we find that $\mathbf{T_{f1}(N) = 100\ N}$ and $\mathbf{T_{f2}(N) = 10\ N \log_2 N}$ where the times are in seconds. (a) Solve algebraically for what size **N** these two functions take the same amount of time. Show how you **calculated** your answer. (b) When is it faster to use **f1**? When **f2**? (c) Briefly describe how we can write a simple function **f** that runs as fast as the fastest of **f1** and **f2** for all size inputs.

(a) We solve for when these two functions are equal: i.e., for what **N** does $\mathbf{100\ N = 10\ N \log_2 N}$; dividing each side by **10N** we have $\mathbf{10 = \log_2 N}$ or $\mathbf{2^{10} = N}$ (or **1,024** ~ **N**). If the complexity classes were more complicated, we could have plotted these equations, or plugged-in various values of **N** to find when they cross.

(b) It is faster to use f1 when **N > 1,024**

It is faster to use f2 when **N < 1,024**

(c) Write a function **f** that tests its argument size; if it is > **1,024** it calls **f1** to solve the problem; if it is < **1,024** it calls **f2** to solve the problem.

6. (5 pts) Assume that function **f** is in the complexity class $\mathbf{O}(\sqrt{N} \times Log_2 N)$, and that for $\mathbf{N = 10^6}$ **(1,000,000)** the function runs in **2 seconds**.

(a) Write a formula, **T(N)** that computes the approximate time that it takes to run (also in seconds) **f** for any input of size **N**. Show your work/calculations by hand, approximating logarithms, finish/simplify all the arithmetic.

From the formula, $\mathbf{T(N) = c\ (\sqrt{N} \times Log_2 N)}$ we have $\mathbf{2 = c\ (\sqrt{10^6} \times Log_2 10^6)}$; $\mathbf{2 = c\ (10^3 x20)}$; therefore, $\mathbf{c = 10^{-4}}$. So, $\mathbf{T(N) = 10^{-4} \times (\sqrt{N} \times Log_2 N)}$

(b) Compute how long it will take to run when $\mathbf{N = 10^{12}}$ (one million times as big as $\mathbf{10^6}$). Show your work/calculations by hand, approximating logarithms, finish/simplify all the arithmetic.

$\mathbf{T(N) = 10^{-4} \times (\sqrt{10^{12}} \times Log_2 10^{12}) = 10^{-4} \times (10^6 \times 40) = 10^2 \times 40 = 4{,}000}$ **seconds**

(c) If another function `g` solved the same problem with complexity class **O(N),** would you prefer `f` (analyzed above) or `g` for solving large problems, based on running time only? Explain/justify your reasoning.

(1) $Log_2 N < \sqrt{N}$ so their product is < **N**, or (2) From the above analysis, solving a problem a million times bigger increases the time by a factor of only 2,000; for `g`, an **O(N)** algorithm, would require a factor of a million more time, meaning even with a small constant, eventually it would run more slowly.

7. (3 pts) Fill in the last line of the three empty rows, which shows how much bigger of a problem can be solved for each complexity class on a machine that runs twice as fast. Solve algebraically when you can, with Excel/calculator when you must. Hint: I used a calculator only for O(N Log_2 N) and solved it to 3 significant digits.

N = Problem Size	Complexity Class	Time to Solve on Current Machine (secs)	N Solvable in the same Time on Machine 2x as Fast
10^6	O(Log_2 N)	1	10^{12}
10^6	O(N)	20	2 x 10^6
10^6	O(N Log_2 N)	20	1.916 x 10^6
10^5	O(N^2)	1000	1.414 x 10^5