Required: Student Setup

As the first part of your homework, follow the instructions below to get your environment properly set up for this class. This section is not graded but you will need to do it in order to submit your programming assignments.

- Email fsareshwala@berkeley.edu with a request to be invited to the Campuswire class if you don't already have access
- Complete the instructions on https://github.com/fsareshwala/cs404.1/blob/master/student-setup.md to set up your programming and development environment

Optional: Student Survey

Once you have set up your environment, you will find a file called `survey.md` in the top-level root of the repository. This file contains a survey that asks various questions about your experience with computer science, software engineering, and data structures and algorithms. Answering these questions is completely optional and doesn't affect your grade on this or any future assignment. However, your responses do help me understand your current experience and better tailor the course to your individual needs.

If you are comfortable with giving me a bit more information, please complete the survey within the `survey.md` file in the top-level root of your repository. Your responses will be kept confidential and will never be shared. Once you are done, commit your responses and push them to your upstream repository.

Version Control Systems: `git`

Accountability: who wrote the code?
Branching: support different versions and features
Merging: easily incorporate others' changes

1. What are some of the reasons to use version control systems? Recording keeping: commit logs

   Low efficacy of emailing around latest version of some file.

   You can make changes separately and the merge after and undo bad changes easily.

   Makes sure pieces of code stady together (as some wouldn't makes sense on their own).

   Tracks changes in source file. Also, tracks timing of changes.

2. What advantages does `git` have over previous generations of version control systems? popular

   Distributed version control system: Anyone can clone public repositories; All changes are local until pushed;
   Anyone can make changes without affecting the main project; If project own accepts change, then they can
   pull them; No repository is more important than another; Fast and reliable merges; Atomic commits;

   Networks of trust are more secure. Vs older VCS's; not centralized; no single point of failure; don't need
   permission from project owner to make changes; no non-atomic commits; no confusing branching and merging

3. You get the following error when working with git:

```
Pushing to https://github.com/fsareshwala/dotfiles
To https://github.com/fsareshwala/dotfiles
 ! [rejected]        master -> master (fetch first)
```

   What went wrong?

   As it says, didn't fetch first.

4. Files in the directory `program1` are currently untracked. How can you get git to track them? You may write either the conceptual ideas or the commands necessary that will have git track them.

Push them to staging area with git add.

Complexity

5. Suppose that $T_1(n) = O(f(n))$ and $T_2(n) = O(f(n))$. Which of the following are true:

   A. $T_1(n) + T_2(n) = O(f(n))$

   B. $T_1(n) - T_2(n) = O(f(n))$

   C. $\frac{T_1(n)}{T_2(n)} = O(1)$

   D. $T_1(n) = O(T_2(n))$

   A, B, C, D

6. Group the following into equivalent Big-O functions: n^2, n, $n^2 + n$, $n^2 - n$, and $\frac{n^3}{n-1}$.

   group A: n^2, n^2 + n, n^2 - n, n^3/(n-1)

   group B: n

7. Programs A and B are analyzed and are found to have worst case running times no greater than $150n \log n$ and n^2, respectively. Answer the following questions.

   (a) Which program has the better guarantee on the running time for large values of n ($N > 10,000$)?

   A(10^4) = 19931569 (log base 2) < B(10^4) = 10^8, so A

   (b) Which program has the better guarantee on the running time for small values of N ($N < 100$)?

   A(100) = 99657 > B(100) = 10^4, so B

   (c) Which program will run faster on average for $N = 1,000$?

   A(10^3) = 1494867 > B(10^3) = 10^6, so B

   (d) Can program B run faster than program A on all possible inputs?

   no. (a) is a counterexample. But, in general these functions of n intersect once after n=1 and with different derivatives (slopes) meaning they cross rather than touch and go back to the same order

8. In terms of n, what is the running time of the following algorithm to compute x^n?

```
double power(double x, int n) {
  double result = 1.0;
  for (int i = 0; i < n; i++) {
    result *= x;
  }
  return result;
}
```

   O(n) since *=x is not dependent on n and x isn't changing, and i is up until n.

   I don't think we went over how to get exact time. I guess 2n + 3: 1 for function call, 1 for initializing result, 1 for each check i < n, 1 for each *= x, one for return.

9. An algorithm takes 0.5 ms for an input size of 100 elements. How long will this algorithm take for an input size of 500 elements (assuming that lower order terms are negligible) if the running time is:

   (a) Linear: $O(n)$

   f(n) = c*n --> f(100) = 100c = 0.5 --> c = 0.005 --> f(500) = 0.005*500 = 2.5ms

(b) Linearithmic: $O(n \log n)$ f(n) = c*n*log(n) --> f(100) = 100*log2(100)c = 100*c*log10(100)*log2(10) = 100*d*log10(100) = 0.5 --> c = 0.005 --> f(500) = 0.005*500 = 2.5m

(c) Quadratic: $O(n^2)$ f(n) = c*n ^2 --> f(10^2) = (10^4)c = 0.5 --> c = 5*10^(-5) --> f(500) = 5*10^(-5)*500 = 0.025ms

(d) Cubic: $O(n^3)$ f(n) = c*n^3 --> f(10^2) = (10^6)c = 0.5 --> c = 5*10^(-7) --> f(500) = 5*10^(-7)*500 = 2.5*10^(-4)ms

10. For each of the following program fragments, analyze the running time and provide a time complexity in Big-O notation.

(a)
```
for (int i = 0; i < n; i++) {
  sum++;
}
```

O(n)

(b)
```
for (int i = 0; i < n; i += 2) {
  sum++;
}
```

O(n)

(c)
```
for (int i = 0; i < n; i++) {
  for (int j = 0; j < n; j++) {
    sum++;
  }
}
```

O(n^2)

(d)
```
for (int i = 0; i < n; i++) {
  sum++;
}

for (int j = 0; j < n; j++) {
  sum++;
}
```

O(n)

(e)
```
for (int i = 0; i < n; i++) {
  for (int j = 0; j < n * n; j++) {
    sum++;
  }
}
```

O(n^3)

(f)
```
for (int i = 0; i < n; i++) {
  for (int j = 0; j < i; j++) {
    sum++;
  }
}
```

O(n)

(g)

```
for( int i = 0; i < n; i++) {
  for(int j = 0; j < n * n; j++) {
    for(int k = 0; k < j; k++) {
      sum++;
    }
  }
}
```

O(n^3)