Homework – Bucket Sort

Brad Peterson – Weber State University

This homework assignment implements both a single threaded and then a multithreaded bucket sort. Notable features include:

- Creating and destroying child threads with a fork/join model
- Using a mutex to manage a critical region of code to avoid race conditions
- Create a thread pool model to distribute out work more fairly to any thread ready to work

You need to complete the assignment by implementing the *single threaded* `singleThreadedBucketSort()` with these steps:

- **Step 1 – Copying values from the array into the buckets**
- **Step 2 – Sort each bucket**
- **Step 3 – Copy values from all buckets back into the original array**

Once this works, you should then implement a multithreaded version of this by implementing the *multi threaded* `multiThreadedBucketSort()`. Only Step 2's `multiThreadedBucketSort()` and `multiThreadedStep2()` will utilize parallelism.

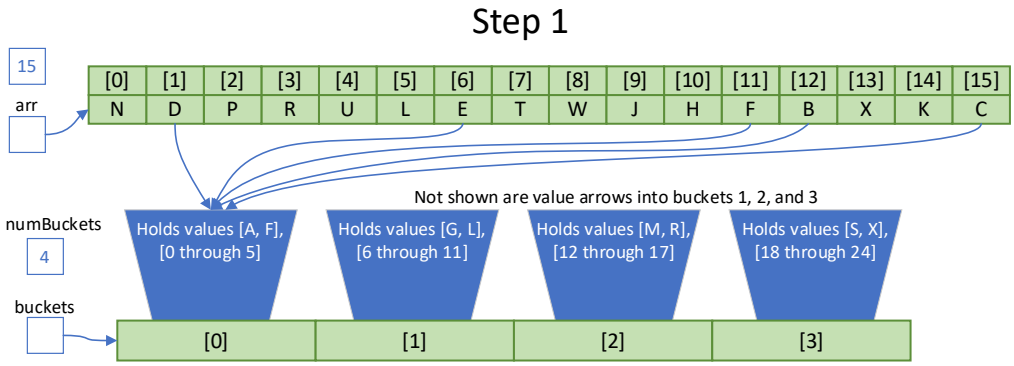
The collection of buckets has been implemented for you as an array of vectors. Thus, `buckets[i][j]` is the *j*th value in the *i*th bucket. To put a value into an *i*th bucket, use `buckets[i].push_back(some_value);`.

Instructions:

Single threaded implementation:

Implement a `Step1()`, `singleThreadedStep2()`, and `Step3()`.

Step 1 – Copying values from the array into buckets. For every item in the input array of random values, use basic computer integer arithmetic to determine into what *j*th bucket index the *i*th value of the array should be copied.



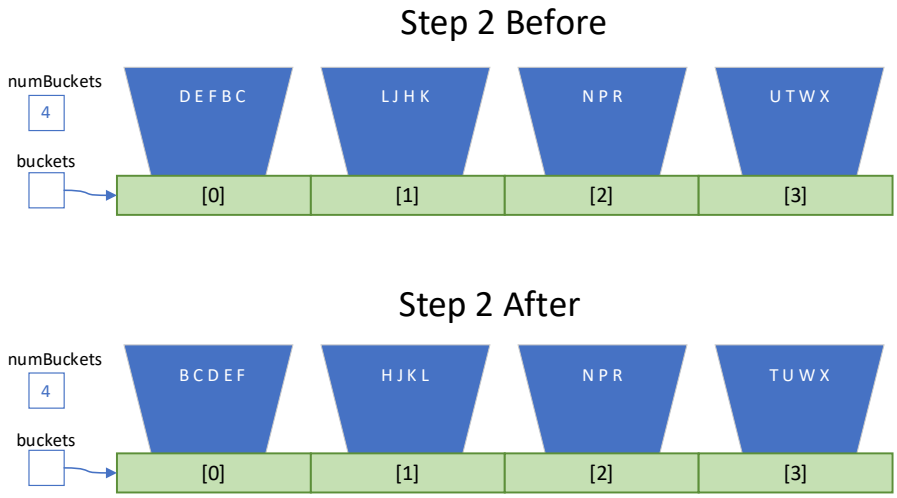
Students often make a mistake with the arithmetic on the first try. For example, suppose the data has a range of 100 numbers, 0 through 99. Suppose also there are 4 buckets, of which each bucket gets 25 numbers. If the data value is 85, then a student would assume to simply compute 85 / 25 = 3, and thus 85 would be placed into bucket 3, which is correct. If the data value is 25, then 25 / 25 = bucket 1, which is correct. If the data value is 24, then 24 / 25 = bucket 0, which is correct. So far so good. Next, the student desires to compute the bucket range value of 25. The student takes the high range, 99, adds 1, divides by the number of buckets (4), and gets

25. Again, so far so good. The issue happens when the data range is 0 through the maximum unsigned integer. Here, the maximum unsigned integer is 2^32 – 1 = 4,294,967,295. This number also has the property that all 32 bits are set to 1. The student takes the high range, 4,294,967,295, adds 1, and divides by the numbers of buckets, and wrongly gets the value 0. This issue occurs because 4,294,967,295 + 1 = 0 in 32-bit unsigned integer arithmetic. When all 32 bits are set to 1, adding another 1 to that value flips all 32 bits back to 0, and the 33rd bit is a lost carry bit.

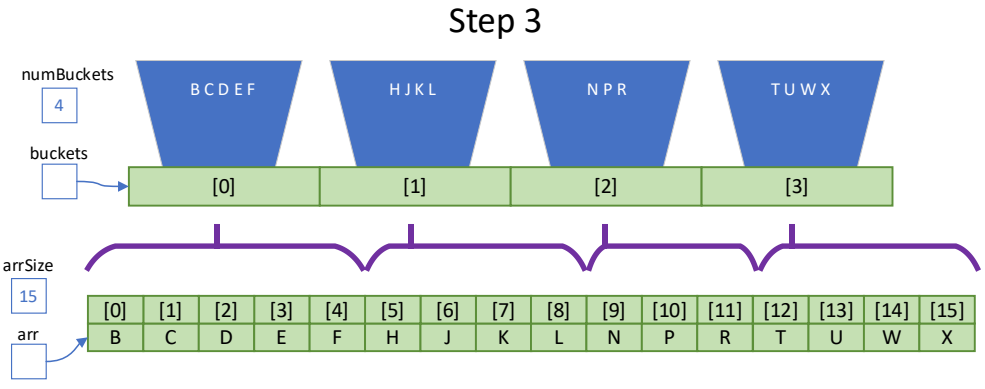
A workaround here is to use the arithmetic: rangePerBucket = largestPossibleValue / numBuckets + 1 when numBuckets is greater than 1, or rangePerBucket = largestPossibleValue when numBuckets is 1. Then divide an array value by the rangePerBucket to determine which bucket this array value should go into. Then put that array value into that bucket. Note that this arithmetic can fail if numBuckets is 1, as 4,294,967,295 can once again wrap around to 0. You will need an if statement to check for a numBuckets = 1 scenario.

Once you have a bucket index for value, simply go to that bucket (use [] indexing), then use .push_back(value);

Step 2 – Sort each bucket. For each bucket, sort it. A function has been provided for you to sort a bucket.



Step 3 – Copying values from the array into buckets. For every *i*th bucket, copy every *j*ith item from that bucket into the original input array `arr` into successive *k*th elements of that array. This logic is a bit tricky to write, as it requires two loops and three indexes. Plan your code carefully.



Multithreaded implementation:

Keep `Step1()`, and `Step3()` as they are. Implement the `multiThreadedThreadedStep2()`, and modify

`multiThreadedBucketSort()` to call it via fork/join model

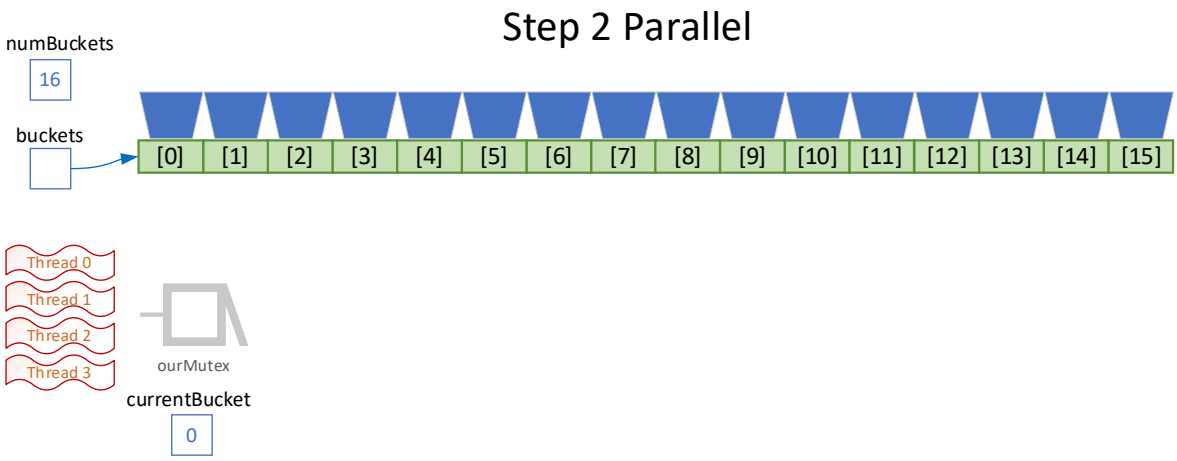
Within `multiThreadedBucketSort()`, after `Step1()` reset the global variable `currentBucket` to `0`.

Create an array of thread tracking objects (see lecture). This array should be sized to the number of threads.

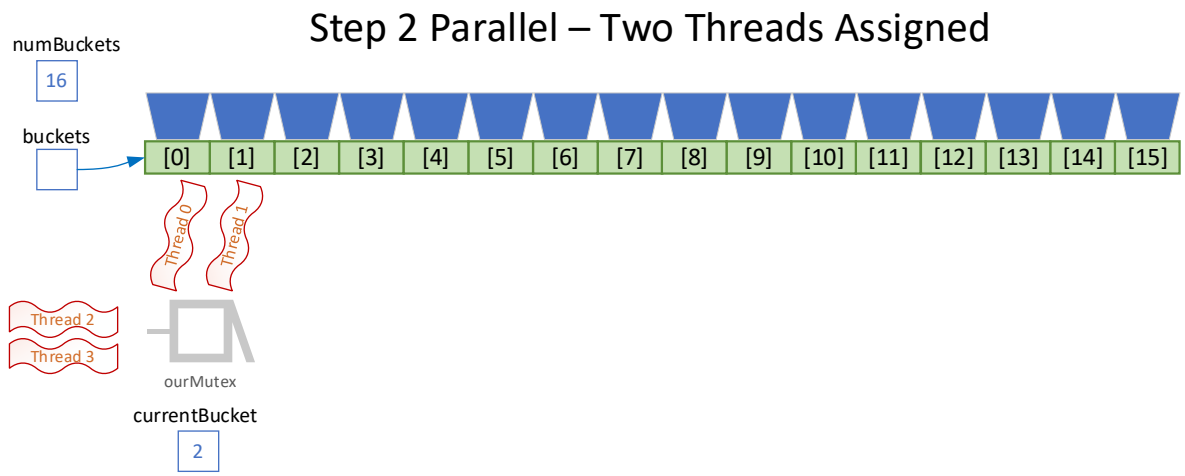
Fork and create child threads that start their lifetime in `multiThreadedThreadedStep2()`. This requires a for loop, then invoking `thread()`, and for the first argument, passing in the function name in which the threads start (make sure you just give the function name `multiThreadedThreadedStep2`, you don't need to give it the parentheses). A second argument into `thread()` is optional, and is mainly useful if you want for debugging purposes to know which thread is doing what work. If you want the this debugging feature, pass in the loop iteration variable as the second argument of `thread()`, and likewise modify `Step2()` so it has a parameter to accept that argument. The returned value of each `thread()` call is stored in the appropriate index of the thread tracking object array created from the prior paragraph.

After you fork the threads, create another for loop to join the threads. For each object in the array of thread tracking objects, call its `join()` method.

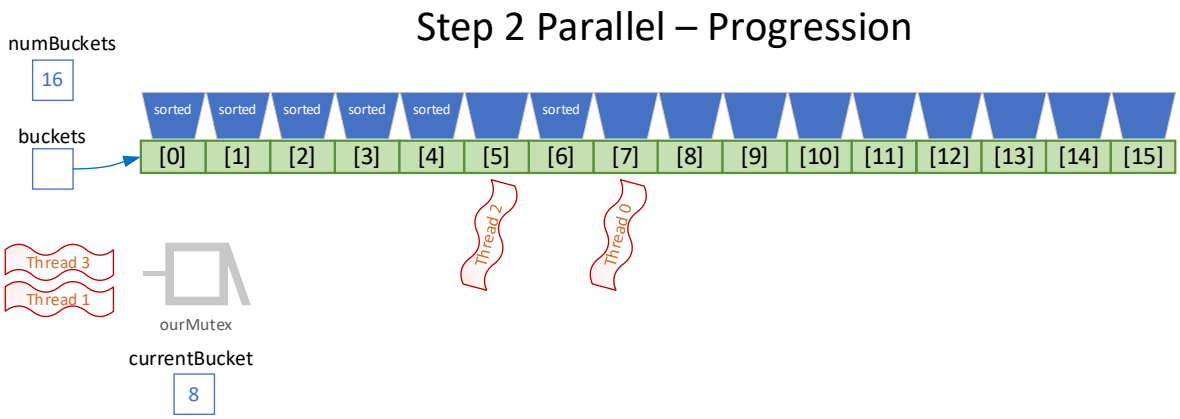
Thread Pool Model in `multiThreadedBucketSort()`:



Each thread will be running independently in `multiThreadedThreadedStep2()`. The goal here is for each thread to obtain a valid bucket on which to work, sort that bucket, then loop around again to find a next valid bucket on which to work, etc.



Create a simple int variable for a thread's bucket index (each thread gets its own variable independent from the others). Initialize that index value to zero. Create an infinite loop. Inside the loop lock the mutex called `ourMutex`. Only one thread at a time can exist in this mutex code region. Copy the global variable's current working bucket index (`currentBucket`) into the thread's bucket index. Increment the global variable bucket index (`currentBucket`). Unlock the mutex. Check if this thread's bucket index is a valid bucket index. If it's not, return out of the function, this thread is done as there is no work left to be found. If it is valid bucket index, have this thread sort that bucket.



In the above figure, six buckets are sorted. Two are currently being sorted. Eight are not sorted. Thread 2 is still working on sorting bucket 5 while another thread has completed bucket 6 (that thread just did its work faster than thread 2). Two threads have reached the mutex and will attempt to lock the mutex to get a unique bucket, and the first thread to unlock that mutex will get to sort bucket 8.

Debugging:

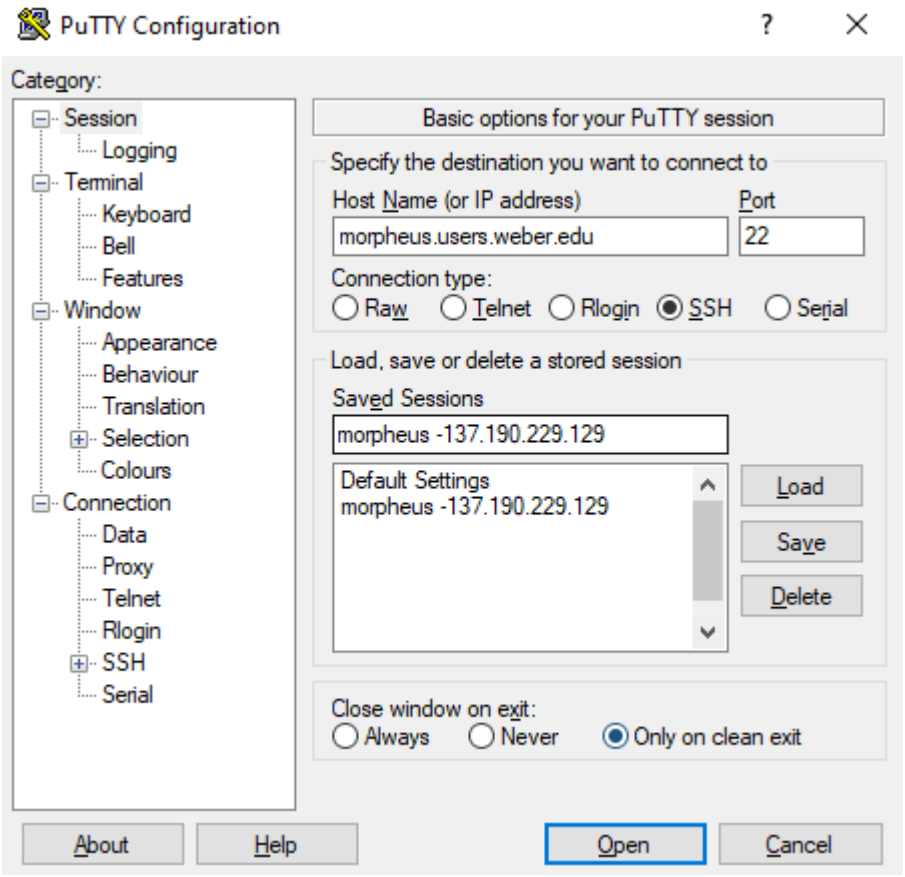
Debugging is trickier for multithreaded programs. I strongly recommend using `printf()` and printing messages to screen to indicate the status of what your code is doing at that moment (don't forget the `\n`). The `cout` tool is awful in a multithreaded context, as its output tends to have race conditions and smear with other threads' `cout` output.

You can use the visual step-by-step debugger, but depending on the IDE, trickier things can happen as you step over or step into.

Run the Program:

Run the assignment locally. Take a screenshot of the result of your best program times. (Your program will run slower as it's in debug mode. If you want to see faster times, you must configure another build for release mode, which CMAKE supports.)

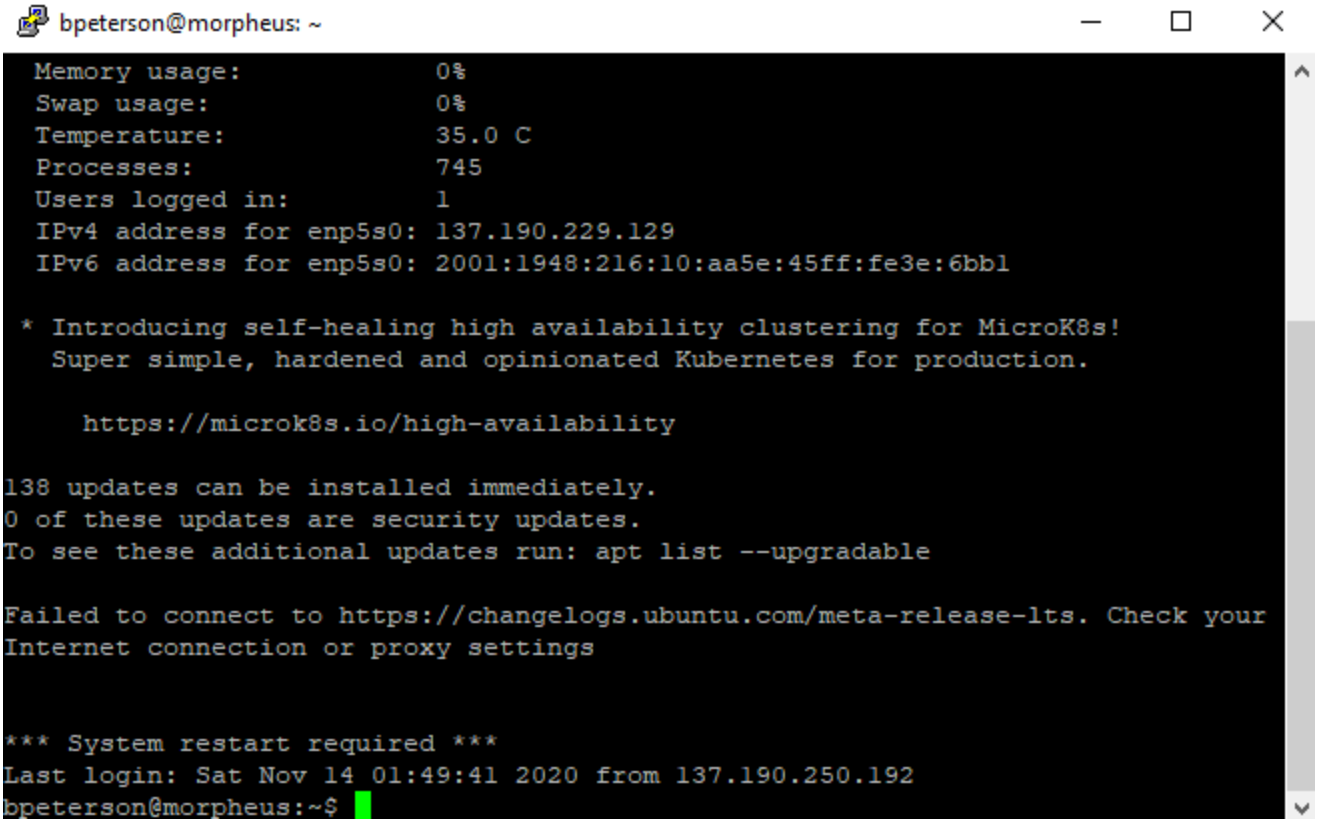
Next, you must run your program on a high-performance server. The server is behind Weber's firewall at the address is morpheus.users.weber.edu. I assume most of us are not inside Weber's firewall, but outside. Thus, the easiest solution to connect to the server behind the firewall is to install the Weber State VPN tool (https://www.weber.edu/help/kb/VPN_Install.html), and then log into the VPN with your normal Weber State credentials. Once the VPN connection is running, I recommend using Putty (https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html) or another ssh tool to verify you can connect to morpheus.users.weber.edu. A screenshot of Putty is found below:



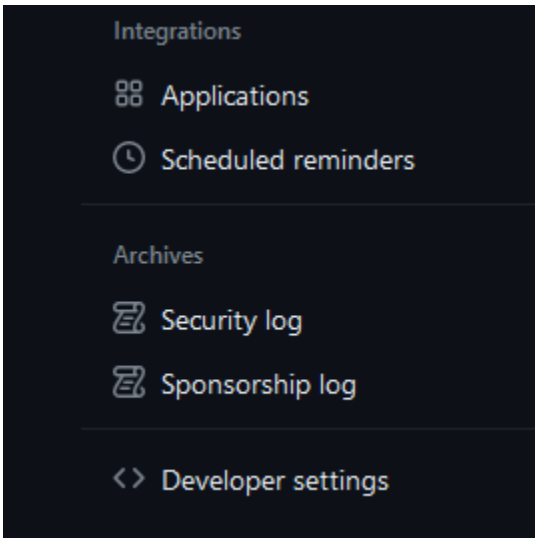
After entering the IP address and clicking Open, the first time your ssh tool should confirm to save a set of keys, just accept that. (Note that ssh does not display any typed password characters, even as asterisks.)

Your username is your initials (lower case) followed by the last 5 digits of your W#. For example, if your name is John Smith and your w# is w00012345 then your username would be js12345. Your password will be your first name (first letter capitalized) followed by "cs!". For example, if your name is John Smith and your w# is w00012345 your password would be Johncs!. One important thing to take note of here is that the usernames and passwords are created from registration data. Try your full first name instead of short versions, i.e. Kimberly instead of Kim.

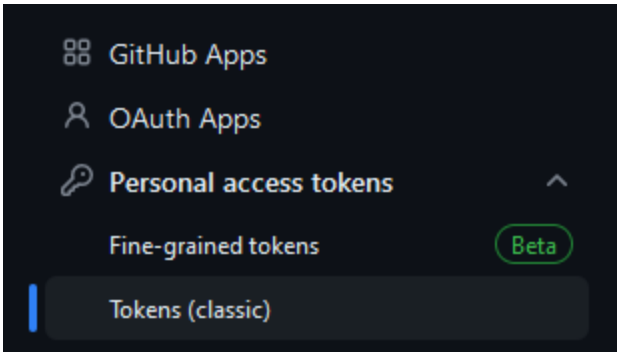
Once you verify you can log into the server via a command line prompt, you should see a screen like this.



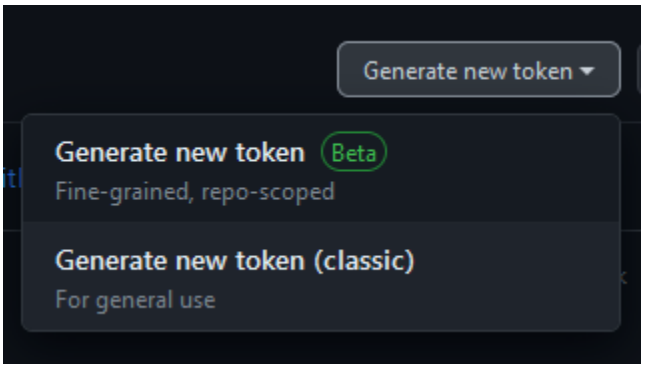
Your next goal is to clone your repository here. To do this, you must first obtain a security token. Log into github.com. Select your profile (or organization if you have several). You can select this in the top right. In the bottom left, choose <> Developer Settings:



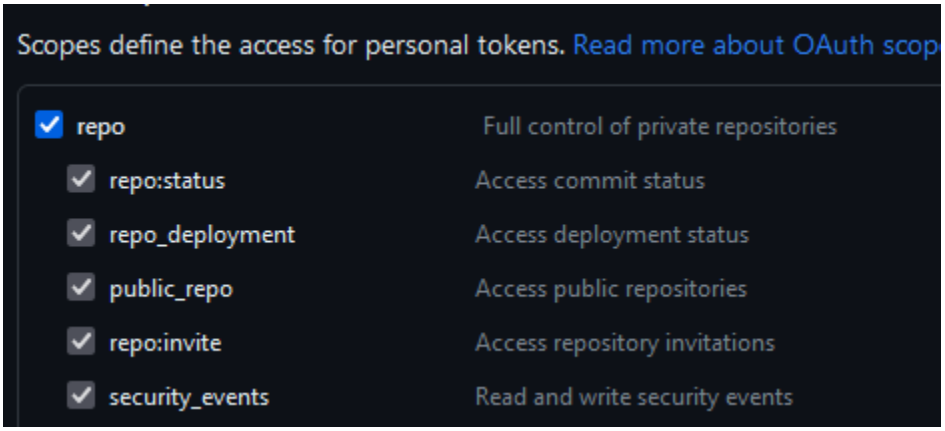
Then choose Personal access tokens, then Tokens (classic).



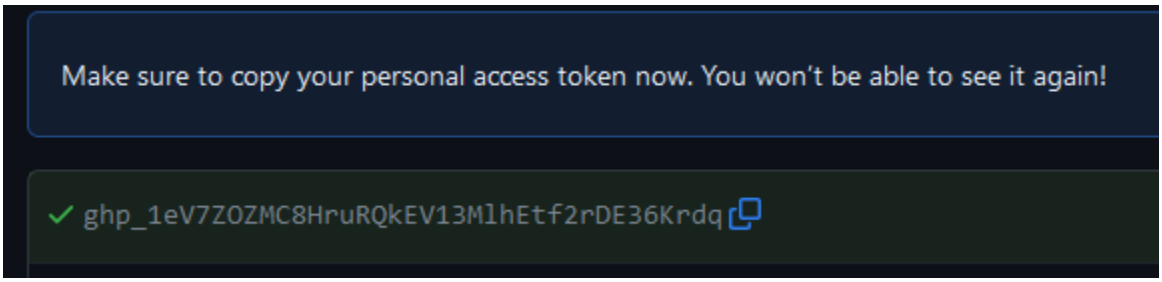
Choose Generate New Token, and then Generate new token (classic)



Give it a name. Choose repo:



Click submit. You should get a long token like this:



Copy that token and put it somewhere safe where you can access it again.

Back in your Linux console, run:

```
git clone https://github.com/your_full_repo_address.git
```

For your username, put in your github username.
For the password, put in the security token you previously generated. In most Linux terminals, a single right mouse click will paste. You won't see the pasted token, but it will paste. Press enter to submit it. If it works, git will clone your repo.

Next, change the directory to your GitHub repo. This will be different for each student. To see your directory, run:

```
ls
```

Then type

```
cd name-of-directory
```

Create a release build:

```
cmake -S . -B release -DCMAKE_BUILD_TYPE=Release
```

Go into the release folder

```
cd release
```

Run make to build:

```
make
```

Run the program:

```
./BucketSortTest
```

Reporting:

Create a document that has two screenshots. One from the output of running your homework on your local machine. A second screenshot of the output from this CS Linux server.