Logistics: You should form groups of 2-4; no solo projects and no groups with > 4 people. Since there are 7 projects, clearly, not every group will have a unique project. Once you decide on your team, you should e-mail Laraib the names of your team members, along with the student ID and e-mail of each. The title of the e-mail should contain the string [CSE549 Final Project]. Additionally, the e-mail should contain a list of 3 projects in rank order (i.e. the first in the list is the one you most want to work on, the second is the one you wish to work on if the first is unavailable, etc.). More than one group is allowed to work on each project, but we do not want e.g. all of the groups choosing the same project. We will try to assign each group's first choice, but cannot guarantee this. Please send this e-mail by Thurs. Oct 6. Once we have received e-mails from all of the teams, we will assign the projects for each team. Once your team has their assignment, you should set up a short meeting with Laraib and I (during our office hours) so that we can answer your initial questions, provide you with more detailed directions, and, if applicable, tell you where to obtain sample data etc.

1. **Improving statistics for paralog detection in RapClust (Python, statistics+implementation)**

Description: In RNA-sequencing data, two genes can have high sequence similarity, due to gene duplication (paralogs) or transfer (orthologs). It can be difficult to distinguish between transcripts of genes that are homologous (i.e. paralogs or orthologs) and transcripts that represent isoforms of the same genes. Yet, such information is important when we want to pose and ask questions about differential gene expression, since paralogs should be treated as different genes, but isoforms of a single gene should not. Across different conditions, the expression levels of paralogous genes genes can vary. This information can be used to differentiate between the two genes. Our tool, RapClust, clusters transcripts in a *de novo* setting based on the gene from which we think they derive. This task is particularly difficult, because, in the *de novo* setting, we don't know the genome of the organism we're analyzing, and so can't map these transcripts back to the genome.

Ideally, transcripts from the same gene should end up in one cluster and those from paralogs/orthologs should fall in separate clusters, since they're originally separate genes. However, due to sequence similarity, they often end up in a single cluster. One standard way of trying to solve this problems is via hypothesis testing (as in the tool Corset). Given two contigs t_1 and t_2 in conditions c_1 and c_2, we have a total of four counts. We assume that the counts originates from a poisson distribution, $N \sim Poss(\lambda)$. For hypothesis testing, we can assume if the change of λ across conditions is significant, the origin of the two contigs is different. Hence our null hypothesis is that the transcripts are from the same origin, $H_0 = \lambda^{c_1}{}_{t_1} / \lambda^{c_1}{}_{t_2} = \lambda^{c_2}{}_{t_1} / \lambda^{c_2}{}_{t_2}$. We wish to test if we should reject this null hypothesis, leading us to believe that the transcripts are actually from paralogous genes.

The problem with this construct is that the count data is not correctly modelled as a Poisson distribution, because the variance is generally greater than mean of the data. This project work includes testing different statistical models, such as the negative binomial, instead of the Poisson. Additionally, you are encouraged to study the sequence of the transcripts themselves to try and derive a classifier that decides if two sequences are isoforms of the same gene, or paralogs of different genes. For example, build a statistical model (e.g. a pair HMM or another model of your choice) on known sets of isoforms, and known sets of paralogs (training), and now, for a new pair of transcripts (testing), determine which model better describes the pair. RapClust source code is publicly available and easy to understand. Major changes would only be in the section doing paralog separation. Data for testing will be provided.

Code: https://github.com/COMBINE-lab/RapClust

2. **Visualizing equivalence classes and mapping ambiguity graph**

Description: The idea of fragment equivalence classes is briefly explained in section 2.2 of the RapClust paper. Given these classes and the corresponding graph from RapClust, we want to be able to view them using a web interface. Since the graph will be huge, some optimization will need to be applied to allow for easier viewing (for example displaying complete details of only some segments of the graph). The user should be able to use the graph to get details about the contigs and the shared equivalence classes. If provided, sequences for each contig should also be displayed in some way. There should be a way to view basic statistics about the graph, such as edge weight distribution and connectivity. Remaining credit will be granted for useful and interesting visualizations and metrics that you come up with. In particular, users should be able to compare and contrast the equivalence class graphs that come from two different experimental / conditions, and the visualization should help them summarize what is shared and different between the equivalence classes in these conditions. The visualizations here should be interactive, implemented, most likely, using a javascript framework.

3. **Comparing Bootstraps vs. Gibbs on RNA-seq data (data analysis)**

Description: Salmon is a state-of-the-art tool for measuring gene expression (quantifying the abundance of different RNA transcripts in an experiment). One of the primary benefits of this method is that it is orders of magnitude faster than the competition, while producing results of the same or better accuracy. Salmon determines expression levels by solving a maximum likelihood problem. A result of this formulation is that one often gets accurate estimates of the transcript abundances, but has no notion of confidence in these predictions. That is, predictions can be highly specific and highly accurate, or highly specific and highly inaccurate. Thus, it is very valuable to provide some measure of confidence in the estimates that are inferred. There are a number of ways to estimate such confidence. One way is “bootstrapping” and another is “Gibbs sampling”. The former approach samples from the original data a number of times, and reruns the maximum likelihood estimator independently on all of these samples. The latter approach produces a “chain” of samples by resampling each observation (sequencing read) in turn.

The goal here is to determine which of these methods for quantifying the certainty of an estimate is most accurate and which is most efficient. Accuracy can be assessed on simulated data by looking at how often the true expression value resides within the bound of the computed credible interval, and by comparing the relative sizes of the corresponding credible intervals. For example, we wish to find the smallest 95% credible interval where the true estimate is within this interval >= 95% of the time. This project will involve extensive testing of accuracy. Finally, you will also explore the relative accuracy between the normal maximum likelihood estimates, and the "summary statistics" produced by the bootstrap estimates and Gibbs sampler. The goals will be to determine if the maximum likelihood estimates are more, less, or equally as accurate as the mean, median and mode of the posterior samples. When comparing the posterior estimates themselves, you will analyze a number of datasets to determine if bootstraping and Gibbs estimate posterior variance accurately (or if one method estimates this better than the other), which cases one or the other method fails, and what characteristics are shared among the failure cases (e.g. are they transcripts that are from genes with many isoforms, are they transcripts from paralogous genes, etc.?).

4. **Packed suffix array (C++, implementation)**
   Description: Our recent paper, RapMap, uses a suffix array (among other data structures) to quickly look through the transcriptome for matching substrings from the reads. For this purpose, the suffix array is initially built on the transcriptome, in an indexing phase, before the mapping process begins. Each character in the suffix array occupies one byte in memory. However, the DNA/RNA alphabet contains only 4 characters [A,T,G,C] which can be encoded using 2 bits of data. Binary encoding the sequence could reduce the space used by the suffix array by 75%.

   This project entails programming a suffix array using binary encoding, along with associated functions, such as construction (for construction, you're allowed to build on existing code, but keep in mind that you're not using a simple byte encoding of the alphabet) and search (these should be written from scratch), and timing results for these. For the search functionality, you will be provided with an API that describes the particular queries you must be able to perform. Initial testing can be done using simulated sequencing data. However, real sequencing data (when the suffix array will be used in practice) contains a delimiter between separate segments of the sequence (called transcripts). That is, we are building a *generalized* suffix array, and so, this delimiter must be encoded in some way. Thus, as part of this project, you will have to devise an efficient way to include an encoding of the delimiter in your suffix array. One idea is to somehow use blank (checkpoint) bytes, but this will require extra bookkeeping. However, other ideas on this are welcome.

5. **Alternative memory layouts for the suffix array**
   Description: The suffix array is a classic data structure for fast exact search in a corpus of text. It is notable, in large part, for its simplicity. However, it has recently seen a resurgence of use in tools like STAR and RapMap. This is, in part, because the simplicity of the typical suffix array operations make them very efficient on modern architectures, and, in practice, an uncompressed suffix array can often outperform more complex (and even asymptotically faster) data structures like the BWT + FM-index on real queries.

Extending on the idea of adopting data structures and access patterns that work well on modern machine architectures, the goal of this project is to implement the typical suffix array search operations (find a pattern, enumerate all occurrences of a pattern, find the maximum mappable prefix of a pattern, and find the longest common extension of two suffixes in the original text) using a more efficient memory layout. The suffix array is based on the idea of performing a binary search over suffixes. Recently, there has been some interesting work discussing different memory layouts for comparison-based search in sorted data. While binary search is asymptotically optimal, the layout of data in memory can make a (sometimes large) practical difference. In this project, you will explore the practical efficiency of the memory layouts described in this work as applied to the suffix array, and you will benchmark the standard suffix array operations (mentioned above) in the *vanilla* suffix array, versus the suffix array stored in this alternative memory layout. The goal here will be to create a practically fast implementation that outperforms the *vanilla* suffix array as much as possible under a broad variety of (genomic) data and different operations.

6. **Statistical modelling of RNA-seq bias using context trees (C++, implementation)**

Description: Sequence specific bias is a known confounding factor in the estimation of gene expression. Many of the models used for quantification make the assumption that reads are observed independent of their sequence content. However, this is often not the case. Thus, determining the preference with which particular sequences are sampled during the sequencing process, and accounting for this bias during quantification, can improve estimates, sometimes substantially.

In this project, you will implement and compare different probabilistic sequence models to determine which is most accurate. Specifically, you will compare variable-length markov models, interpolated markov models, and context trees (context tree weighting). You will compare the ability of these different techniques to learn a probabilistic model of the data. The testing will include an assessment of how efficient they are to train in terms of time and memory, as well as how well they can predict the biases observed in the underlying data. On synthetic data, the method should (1) not harm quantification estimates when there is no bias introduced (i.e. it should not learn bias where none exists) and (2) improve quantification estimates when there is bias. We will provide testing data on which to assess the different models, and will discuss methods by which accuracy can be assessed. References:
http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3315719/pdf/bts055.pdf and references therein.

7. **Improved Consensus Calling for Pacbio data**

Description: Long reads sequencing technologies like Sequel and RSII by Pacific Bio-Science, Nanopore by Oxford have a significant advantage over short reads in terms of whole genome sequencing and many more biological analyses. However, due to their higher rate of error in the sequencing, typical existing tools do not work, and downstream analysis can be significantly affected. To generate better and high quality reads these technologies have to sequence a single molecule multiple times and generate a *consensus* at the end, which is a significant computational burden. Depending on the number of passes of a single molecule, one can approach 100% accuracy. However, to obtain this accuracy, one may need to read the underlying molecule 60 or more times (and generate all of the resulting “raw” reads, which is costly).

Before moving further let’s first introduce some pacbio based terminologies for long read sequencing. We sequence a single molecule multiple time inside a mechanical box which is called ZMW. Each ZMW returns a **polymerase read** which is one single sequence containing the sequence of the molecule multiple times in both directions. As a preprocessing step, a polymerase read is divided into **subreads;** where each subread represents single pass over the molecule. Note: In the preprocessing stage of chopping the polymerase read to subreads we remove a lot of low-quality region (barcodes and adapter sequences, you can ignore if you don’t know) which can result in different length subreads. Our task is to generate a consensus out of these subreads which represent the *most probable sequence* given these subreads.

The most intuitive way to approach this problem is to perform MSA (Multiple sequence Alignment). The roots of MSA goes back quite a while, but it’s use in long read sequencing technology is most related to a powerful and useful representation for progressive MSA, called the POA (Partial Order Alignment) introduced in the early 2000s . POA is simply a DAG(directed acyclic graph) which concisely represents all the query sequence in the form of a graph and reduce the problem of progressively adding sequences to an MSA to that of finding the most probable / highest scoring (based on the selected framework) path in the graph. There are many intricacies in the process. First of all, the order of the sequences being processed for generating POA can result in different graphs and the weighting function can also play a major role in generating a different consensus at the end (since, as we discussed in class, MSA is NP-HARD).

The consensus can be improved even more using error modeling techniques as shown in the recent work by Pacbio in the form of Quiver/Arrow and by Simpson et al. in Nanopolish. Quiver uses different error scores for each match, mismatch, insertion and deletion while a similar tool for Nanopore data uses profile HMM technique to model error and polish the initial consensus been generated by POA. In the same view, Pacbio has also moved towards developing HMM for PacBio data in the form of Arrow (not published yet). Pacbio’s whole project is open source and their polishing algorithm CCS2(circular consensus sequencing) can be found here (Salute for this).

This project will focus mainly on the first part of the pipeline i.e. generating one consensus sequence using the POA approach. One of the main goals will be to design a better scoring function for extracting a consensus sequence from the graph. Likely, some reading on how pacbio long read sequencing works and the characteristics of the data it produces will be required to understand how a score function should be designed. Currently, PacBio uses the POA graph to get one consensus string using maximum weighted score where scoring is defined in terms of the out-degree of the vertices of the graph. Some slides explaining this can be found here. It is very clear that the current weighting scheme will result in a preference for insertions, which, unsurprisingly are the most dominant form of error in PacBio data. In addition to changing the weighting function, you are encouraged to explore e.g. alternative orderings for building the POA based on ideas like those present in the STAR MSA algorithm. Initial datasets can be found here, to give you an idea of what the data looks like.