ما در این پروژه برای پیاده‌سازی درخت هافمن از ساختار مین‌هیپ استفاده کردیم. تابع‌های اصلی در مین‌هیپ ما، درج، استخراج و هیپیفای است. این تابع‌ها به شرح زیر هستند:

تابع درج: ما لیستی از نودها داریم. هنگام درج کردن ابتدا به آخر این لیست نود جدید را اضافه می‌کنیم. سپس این نود را با والدش مقایسه می‌کنیم و اگر از آن بزرگ‌تر بود، جای آن‌ها را عوض می‌کنیم. این کار را تا جایی که به ریشه‌ی درخت یا همان اندیس ۰ در لیست برسیم، و یا مقدار نود جدید از والدش کمتر شود ادامه می‌دهیم. در نهایت جای درست این نود را در لیست‌مان پیدا می‌شود. مرتبه‌ی زمانی این تابع log n است.

تابع استخراج: در این تابع ما ریشه‌ی درخت مین‌هیپ‌مان را که همان عنصر ۰ام در لیست است بر می‌گردانیم و آخرین عضو لیست‌مان را به جای آن می‌گذاریم و سپس درخت را با توجه به این تغییر، هیپیفای می‌کنیم. مرتبه‌ی زمانی این تابع، همان مرتبه‌ی تابع هیپیفای که log n است، می‌باشد.

تابع هیپیفای: در تابع هیپیفای ما ابتدا ریشه‌ی درخت را که به آن داده‌ایم، بررسی می‌کنیم و مقدار آن را با مقدار فرزندان چپ و راست‌اش مقایسه می‌کنیم و کوچک‌ترین بین آن‌ها را پیدا می‌کنیم. سپس مقدار ریشه را با کوچک‌ترین، در صورت وجود عوض می‌کنیم و این کار را برای آن با صدا کردن دوباره‌ی تابع ادامه می‌دهیم. مرتبه‌ی زمانی این تابع log n است.

حال به ساختار درخت هافمن بپردازیم. ما برای این کار ابتدا جدولی به نام freqMap درست می‌کنیم و فرکانس کاراکترهای ورودی‌مان را پیدا می‌کنیم. این کار را با استفاده از تابعی به نام findFreq که از مرتبه‌ی n^2 است انجام می‌دهیم. سپس به ازای هر مدخل در جدول‌مان یک نود می‌سازیم و آن را در مین‌هیپ‌مان درج می‌کنیم. این کار از مرتبه‌ی nlog n است. سپس دوتا دوتا از این درخت هیپ استخراج می‌کنیم و با کمک آن‌ها یک نود جدید می‌سازیم و آن را در درخت هیپ درج می‌کنیم. این کار را تا زمانی که سایز درخت هیپ‌مان به ۱ برسد ادامه می‌دهیم. مرتبه‌ی زمانی این کار nlog n است. پس از این مراحل ما از تابعی برای محاسبه و ذخیره‌ی کد هافمن هر

کاراکتر استفاده می‌کنیم. این تابع storeCodes نام دارد و نحوه‌ی کار آن چنین است. این تابع درخت را پیمایش می‌کند و اگر به فرزند چپ رفت یک ۰ به رشته‌ای که برای ذخیره‌سازی کد، استفاده می‌شود اضافه می‌کند و اگر به راست رفت یک ۱ به آن رشته اضافه می‌کند. این کار را تا زمانی که به یک برگ برسد ادامه می‌دهد و سپس اطلاعات بدست آمده را در یک جدول به نام codeMap ذخیره می‌کند. با توجه به ماهیت بازگشتی این تابع، پس از اجرای آن کد هافمن تمام کاراکترها پیدا می‌شود. مرتبه‌ی زمانی این تابع 2^n است. در نهایت به معرفی تابع دیکد می‌پردازیم. این تابع decodeFile نام دارد و نحوه‌ی کار آن چنین است. درخت هافمن و رشته‌ی انکود شده را به عنوان ورودی می‌گیرد و رشته را پیمایش می‌کند و با توجه به آن ۰ و ۱ ها، درخت را پیمایش می‌کند، تا به برگ برسد و مقدار آن برگ را به رشته‌ی پاسخ اضافه کند. سپس دوباره به ریشه بر می‌گردد و این کار را تا رشته تمام شود ادامه می‌دهد. در نهایت رشته‌ی پاسخ را بر می‌گرداند. مرتبه‌ی زمانی این تابع n است.