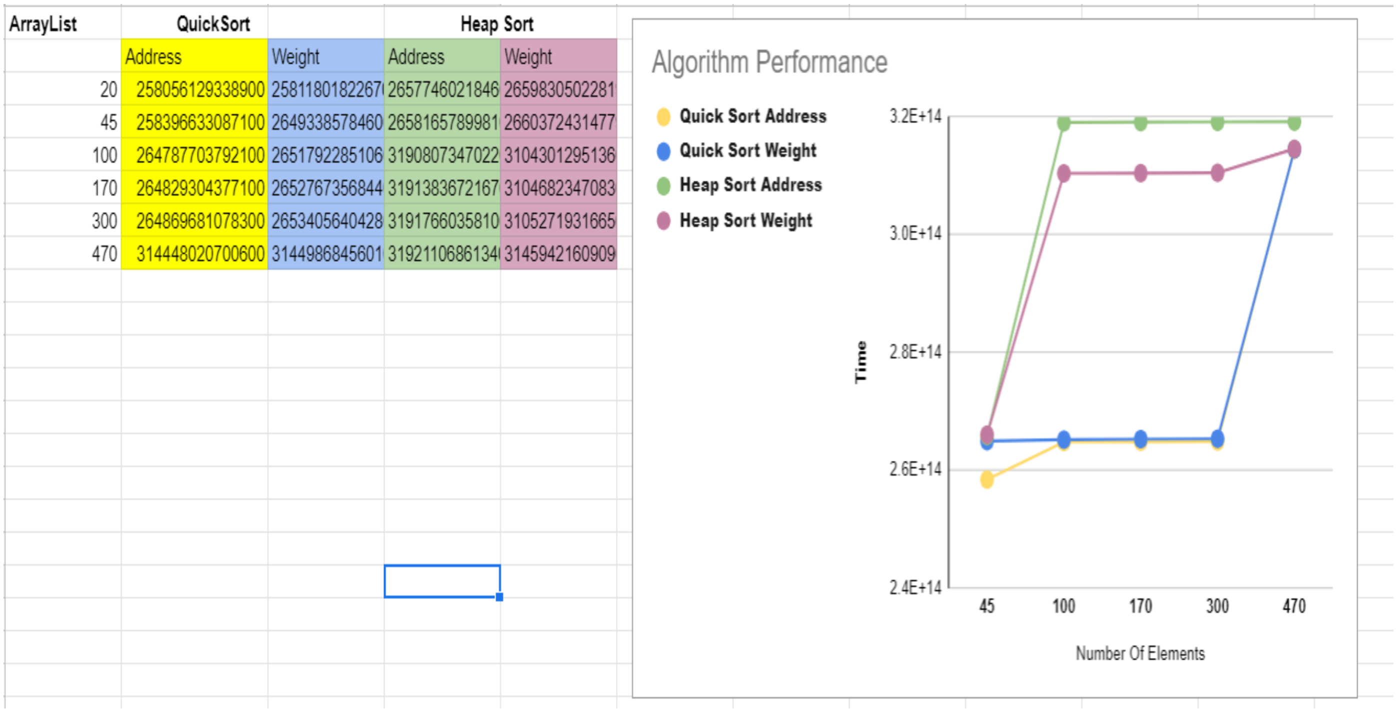

ArrayList	QuickSort		Heap Sort	
	Address	Weight	Address	Weight
20	258056129338900	2581180182267	2657746021846	2659830502281
45	258396633087100	2649338578460	2658165789981	2660372431477
100	264787703792100	2651792285106	3190807347022	3104301295136
170	264829304377100	2652767356844	3191383672167	3104682347083
300	264869681078300	2653405640428	3191766035810	3105271931665
470	314448020700600	3144986845601	3192110686134	3145942160909

For this project I had to sort the address,weight by using Quick Sort, Heap Sort, and Radix Sort. I had trouble parsing my file using the buffer reader that I started late with implementing my algorithms, adding all the errors I was getting from Quick and Heap sort I didn't get to radix sort. I made a bitcoin object to store all attributes a bitcoin had. An arraylist helped with storing all the data. I added switch cases so I didn't have to implement another separate method for weight.

Looking at the graph, we have time(ms) as our y value and number of elements as our x value. Quick sort average case run time is 0(n log n). It takes about the same time to sort the address and weight.
When sorting address with heap sort we can see it takes longer than weight. The time complexity with heap sort isO(n log n). Build max heap takes O(n)
Heapify runs O(log n) but we end up calling it n-1 times.

```java
import java.io.BufferedReader;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.io.FileReader;
import java.io.*;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.Scanner;

public class CVSReader {


    public static void main(String[] args) throws IOException {
        // First way
        String path;
        String line;
        Bitcoin currentBitcoin = null;
        int numberOfElements = 470;

        // ArrayList<Bitcoin> bitcoinsarray = new ArrayList<Bitcoin>();
        List<Bitcoin> bitcoinsarray = new ArrayList<Bitcoin>();
        List<Bitcoin> bitcoinarray2 = new ArrayList<>();
        List<Bitcoin> sortedbitcoinadress = new ArrayList<>();

        List<String> bitcoinAdress = new ArrayList<>();


        path = "/Users/Mariana/Desktop/CSC365/BitcoinHeistData.txt";
        line = "";
        FileWriter fw = new FileWriter("/Users/Mariana/Desktop/CSC365/output.txt", true);
        PrintWriter out = new PrintWriter(fw);

        try {

            BufferedReader br = new BufferedReader(new FileReader(path));
            line = br.readLine();

            while ((line = br.readLine()) != null) {


                String adress = "";
                int year = 0;
                int day = 0;
                int length = 0;
                double weight = 0;
                int count = 0;
                int looped = 0;
                int neighbors = 0;
                double income = 0;
                String label = "";
                String[] values = line.split(",");
                adress = values[0]; //prints adress
                year = Integer.parseInt(values[1]);
                day = Integer.parseInt(values[2]);
                length = Integer.parseInt(values[3]);
                weight = Double.parseDouble(values[4]);
                count = Integer.parseInt(values[5]);
                looped = Integer.parseInt(values[6]);
                neighbors = Integer.parseInt(values[7]);
                income = Double.parseDouble(values[8]);
                label = values[9];


                currentBitcoin = new Bitcoin(adress, year, day, length, weight, count, looped, neighbors, income, label);
                bitcoinsarray.add(currentBitcoin);
                line = br.readLine();


            }
            // PRINTS ALL THE DATA INTO OUTPUT FILE
            /*for(Bitcoin b: bitcoinsarray){
                out.println(b.toString());
            }*/
            //out.close();


        } catch (FileNotFoundException e) {
            e.printStackTrace();

        } catch (IOException e) {
            e.printStackTrace();

        }
        bitcoinarray2 = getRandomBitcoin(bitcoinsarray, numberOfElements);
        int low = 0;
        int high = bitcoinarray2.size() - 1;
        Scanner scanner=new Scanner(System.in);
        System.out.println("Quicksort:"+"\n" +
                "Address=1"+"\n"+
                "Weight=2"+"\n"+"HeapSort:"+"\n"+"Adress=3"+"\n"+"Weight=4"+":");
        int choice;
        choice=scanner.nextInt();
        switch (choice){
            case 1:
                out.println("Unsorted");
                for (Bitcoin a : bitcoinsarray) {
                    out.println( a.toString());
                   }
                   long startTime=System.nanoTime();
                   QuickSort(bitcoinarray2, low, high,1);
                   long k=System.nanoTime();

                   out.println("sorted Arraylist");
                   for (Bitcoin b : bitcoinarray2) {
                       out.println(b.toString());

                }
                out.println("Program runtime:"+k+"ms");
                out.close();
                break;
            case 2:
                out.println("Unsorted:");
                for(Bitcoin c:bitcoinarray2){
                    out.println(c.toString());
                }
                long statTime2=System.nanoTime();
                QuickSort(bitcoinarray2,low,high,2);
                long l=System.nanoTime();
                out.println("Sorted:");
                for(Bitcoin d:bitcoinarray2){
                    out.println(d.toString());
                }
                out.println("Program runtime:"+l+"ms");
                out.close();
                break;
            case 3:
                out.println("Unsorted:");
                for (Bitcoin d:bitcoinarray2){
                    out.println(d.toString());
                }
                long startTime3=System.nanoTime();
                HeapSort(bitcoinarray2,3);
                long h=System.nanoTime();
                out.println("Sorted:");
                for(Bitcoin e:bitcoinarray2){
                    out.println(e.toString());
                }
                out.println("Program runtime:"+h+"ms");
                out.close();
                break;
            case 4:
                out.println("Unsorted:");
                for(Bitcoin f:bitcoinarray2){
                    out.println(f.toString());
                }
                long starTime4=System.nanoTime();
                HeapSort(bitcoinarray2,4);
                long u=System.nanoTime();
                out.println("Sorted:");
                for(Bitcoin g:bitcoinarray2){
                    out.println(g.toString());
                }
                out.println("Program runtime:"+u+"ms");
                out.close();
                break;


        }
        scanner.close();

```

```java


    }


    public static List<Bitcoin> getRandomBitcoin(List<Bitcoin> bitcoinsarray, int totalItems) {
        Random rand = new Random();
        List<Bitcoin> newBitcoinList = new ArrayList<>();
        for (int i = 0; i < totalItems; i++) {
            // take random index between 0 to size of given list
            int randomIndex = rand.nextInt(bitcoinsarray.size());
            newBitcoinList.add(bitcoinsarray.get(randomIndex));
        }
        return newBitcoinList;

    }
    //QUICKSORT


    private static void QuickSort(List<Bitcoin> inputBitcoin, int low, int high,int choice) {
        if (choice==1) {
            if (low < high) {
                int partitionIndex = Partition(inputBitcoin, low, high, 1);
                //sort each partition
                QuickSort(inputBitcoin, low, partitionIndex - 1, 1);//left
                QuickSort(inputBitcoin, partitionIndex + 1, high, 1);//right

            }
        }
        if(choice==2){
            if (low < high+1) {
                int partitionIndex = Partition(inputBitcoin, low, high, 2);
                //sort each partition
                QuickSort(inputBitcoin, low, partitionIndex - 1, 2);//left
                QuickSort(inputBitcoin, partitionIndex + 1, high, 2);//right

            }

        }
    }

    public static void SwapBitcoin(List<Bitcoin> inputBitcoin, int index1, int index2) {
        Bitcoin temp = inputBitcoin.get(index1);
        inputBitcoin.set(index1, inputBitcoin.get(index2));
        inputBitcoin.set(index2, temp);


    }

    public static int Partition(List<Bitcoin> inputBitcoin, int low, int high,int choice) {
        //pick the pivot
        if(choice==1) {

            int middleIndex = low + (high - low) / 2;
            Bitcoin pivot = inputBitcoin.get(middleIndex);
            //Divide them
            SwapBitcoin(inputBitcoin, low, middleIndex);// swap pivot with the far left
            int pindex = low + 1;//left pointer
            for (int i = pindex; i <= high; i++) {
                if (inputBitcoin.get(i).getAddress().compareTo(pivot.getAddress()) < 0) {
                    SwapBitcoin(inputBitcoin, i, pindex);// swap if element is less than the pivot
                    pindex++;
                }
            }

            SwapBitcoin(inputBitcoin, low, pindex - 1);

            return pindex;// return the index of pivot value
        }
        else {
            int middleIndex = low + (high - low) / 2;
            Bitcoin pivot2 = inputBitcoin.get(middleIndex);
            //Divide them
           SwapBitcoin(inputBitcoin, low, middleIndex);// swap pivot with the far left
            int pindex = low + 1;
            for (int i = pindex; i <= high; i++) {
                if (Double.compare(inputBitcoin.get(i).getWeight(),pivot2.getWeight())<=0){
                    SwapBitcoin(inputBitcoin, i, pindex);// swap if element is less than the pivot
                    pindex++;
                }
            }

            SwapBitcoin(inputBitcoin, low, pindex - 1);

            return pindex-1;

        }

    }

    // HEAPSORT
    public static void HeapSort(List<Bitcoin>inputBitcoin,int choice){
        if(choice==3) {

            BuildMaxHeap(inputBitcoin, 3);
            int sizeHeap = inputBitcoin.size() - 1;
            for (int i = sizeHeap; i > 0; i--) {
                SwapBitcoin(inputBitcoin, 0, i);
                sizeHeap = sizeHeap - 1;
                heapify(inputBitcoin, 0, sizeHeap, 3);
```

```java

                }
            }
            else{
                BuildMaxHeap(inputBitcoin, 4);
                int sizeHeap = inputBitcoin.size() - 1;
                for (int i = sizeHeap; i > 0; i--) {
                    SwapBitcoin(inputBitcoin, 0, i);
                    sizeHeap = sizeHeap - 1;
                    heapify(inputBitcoin, 0, sizeHeap, 4);

                }

            }

        }
        public static void BuildMaxHeap(List<Bitcoin>inputBitcoin,int choice){
            if(choice==3) {
                for (int i = (inputBitcoin.size() - 1) / 2; i >= 0; i--) {
                    heapify(inputBitcoin, i, inputBitcoin.size() - 1, 3);
                }
            }
            if(choice==4) {
                for (int i = (inputBitcoin.size() - 1) / 2; i >= 0; i--) {
                    heapify(inputBitcoin, i, inputBitcoin.size() - 1, 4);
                }
            }



        }


        public static void heapify(List<Bitcoin>inputbitcoin,int i, int sizeHeap,int choice){
            if(choice==3) {
                int left = 2 * i;
                int right = 2 * i + 1;
                int largest;
                if (left <= sizeHeap && inputbitcoin.get(left).getAddress().compareTo(inputbitcoin.get(i).getAddress()) > 0) {
                    largest = left;

                } else {
                    largest = i;
                }
                if (right <= sizeHeap && inputbitcoin.get(right).getAddress().compareTo(inputbitcoin.get(largest).getAddress()) > 0) {
                    largest = right;
                }
                if (largest != i) {
                    SwapBitcoin(inputbitcoin, i, largest);
                    heapify(inputbitcoin, largest, sizeHeap,3);
                }
            }
            if(choice==4){
                int left = 2 * i;
                int right = 2 * i + 1;
                int largest;
                if (left <= sizeHeap && inputbitcoin.get(left).getWeight()>=inputbitcoin.get(i).getWeight()) {
                    largest = left;

                } else {
                    largest = i;
                }
                if (right <= sizeHeap && inputbitcoin.get(right).getWeight()>=inputbitcoin.get(largest).getWeight()) {
                    largest = right;
                }
                if (largest != i) {
                    SwapBitcoin(inputbitcoin, i, largest);
                    heapify(inputbitcoin, largest, sizeHeap,4);
                }

            }
        }
      /*  // RADIX SORT
       static void radixsort(List<Bitcoin>inputBitcoin,int n) {
            int max = 256;

        }


        public static void bucketsort(List<Bitcoin>inputBitcoin,int stringlenth){
            int max=256;
            ArrayList<String> buckets=new ArrayList[max];
            for(int i=0;i<max;i++){
                buckets.get(i)=new ArrayList<>()
            }
        }
    */
    }
```