Time Complexity Analysis of Closest Pairs Algorithm Variants

Report for Practical 1 by 180003815

Abstract

The main objective of this practical was to implement various solutions to the Closest Pair of points algorithm with different time complexities. Using python 3, three variants of solution to this problem were implemented in order of decreased time complexity. Through experimentation it was proved the time complexity of the algorithms implemented were $O(n(\log n)^2)$. These reduce time complexities were achieved by using divide and conquer methods. The second variation was proven to be $O(n \log n)$.

Introduction

The aim of the first section of this practical was to implement a solution to the closest pair of points problem using algorithm denoted by CS302 Lectures[1]. The closest pair of points problem is we are given a set of points in 2D space and we need to find the pair of points which are the closest in this set. This problem has a naïve solution by running Euclidean distance formula upon every combination of points in the point space. This has a time complexity of $O(n^2)$. The Euclidean distance formula used for determining the distance between 2 points is denoted below.

$$\text{distance}(p_1, p_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

The algorithms created are denoted `cp1`, `cp3` and `cp4` respectively throughout. Algorithm `cp1` is the algorithm which is implemented using Quick Select and sorting by y index at each stage. Algorithm `cp3` was implemented by sorting the set of points initially and maintaining sorted order throughout all the dividing stage, and using a merge sort like return of points sorted in increasing order of y indexes.

Procedures and Methodology

Implementation of `cp1`

The recurrence relation for the time complexity:

$$T(n) = T(n/2) + O(n \log n)$$

Algorithm

The closest pair of points algorithm for `cp1` was implemented by the algorithm as follows:

1. Find the length of the set of points
2. If the length is less than or equal to 3 run the brute force algorithm for closest distances of points and return distance and the pair points which are the closest.
3. If the length is greater than 3 use quick select to find the median x coordinate of the set of points. (quick select will move this x value to its true position, the middle of the list, all points to the left of it will be of a smaller x value and all to the right will be a greater value.)
4. Split the list of points around the median and run closest points algorithm recursively upon left and right lists respectively.
5. Set the minimum distance to the smallest return be closest points algorithm on left and right sides of lists.
6. Find the smallest distance across the median point
    1. Filter out all the points which are out with the distance of boundary
    2. Sort the filtered points by y coordinates
    3. Compare each point to the 6 most significant points close to it. Storing the shortest distance found.
7. Return the smallest distance found.

Quick Select

Quick Select has amortized time complexity of $O(n)$.

The pivot selection strategy chosen was to always select the middle index. This was chosen as was chosen as it avoid the worst case ($O(n^2)$) scenario of selecting from a sorted array which already has values in the correct index of another benefit was unlike random selecting pivot the results are consistently reproducible as their is no random variation in running time so similar results can be observed when measuring times to run the program.

Measuring Time Complexity

The experiment can be ran using the following commands:

```
./cp2
```

The following experimental procedure was undertaken to show the time complexity of `cp1`

Experimental Procedure Parameters

The following parameters are used to complete the experiment and can be changed from the object initialization.

1. The closest points algorithm which is being tested
    1. This algorithm works for all the algorithms created
2. The number of test points to generate
    1. The experimental results obtained use a length of `1000`
3. The number of times to run a length of dataset to find the fastest time.
    1. This is to avoid random variation in clock time obtained
    2. Since fastest time converges this removes all the random variation in clock time due to events such as garbage collection obtained
    3. This was chosen over the average or median since the lowest value is less vulnerable to large fluctuation in results since it does converge to a value.
    4. The number of times chosen in this experiment was `4`
4. The seed for the random generation of points:
    1. This is available to make sure that results obtained are reproducible.
    2. The experimental results obtained use a seed of `10`
5. The name of the file which the experimental data is being outputted to

Experimental Procedure

1. Instantiate Complexity experiment class with chosen parameters
2. The experiment algorithm loops for the max number of test parameter chosen
3. A data set of `n+100` items with the x coordinate being the same is generated with random why coordinates using the seed set. `n` is the number represent the nth iteration of the loop.
4. The program times how long it takes the give algorithm to run and repeats this process till the chosen repeat parameter number is reached. The minimum is then enter to the data as the chosen value.
5. The minimum from the first iteration of the program is set as the scale of the program.
6. To compare the experimental time obtained the $O(n^2), O(n \log n), O(n(\log n)^2)$.
    1. Functions were created were each of these respective time complexities
    2. They were divide by the `f(100)` to appropriately scale them by the initial scale of the initial minimum found of the experiment. This was to ensure that all the function had the same starting output. The movement of the functions respective to the time observed of experimental results. This scale if appropriate since the `scale/f(100)` is just a constant so is valid of time complexity comparisons.

The experiment has a progress bar to show how close to completion it is.

Implementation of `cp3`

$$T(n) = T(n/2) + O(n)$$

Algorithm

The closest pair of points algorithm for `cp3` was implemented by the algorithm as follows:

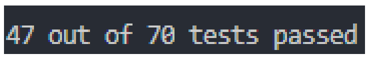
1. Sort the set of points
    1. $O(n \log n)$ operation which occurs once before the closest points function is called.

2. Call the Closest points function
3. If the length is less than or equal to 3 run the brute force algorithm for closest distances of points and return distance and the pair points which are the closest. Also return the space of points sorted in order of increasing y.
4. If the length is greater than 3, to find the median x coordinate of the set of points just use the length/2 floored since the array of points is sorted in order of x.
5. Split the list of points around the median and run closest points algorithm recursively upon left and right lists respectively.
6. Set the minimum distance to the smallest return be closest points algorithm on left and right sides of lists.
7. Insert elements of the set points retrieved from the right side into the left side return set of points in sorted order using a merge sort like comparison. This set eliminates the need to sort by y.
8. Find the smallest distance across the median point
    1. Filter out all the points which are out with the distance of boundary
    2. Compare each point to the 6 most significant points close to it. Storing the shortest distance found. Using the y sorted set of points
9. Return the smallest distance found.

Implementation of `cp4`

There was not enough time nor materials found online to begin the implementation of this. Given more time this function would have been implemented.

`stacscheck` output



```
47 out of 70 tests passed
```

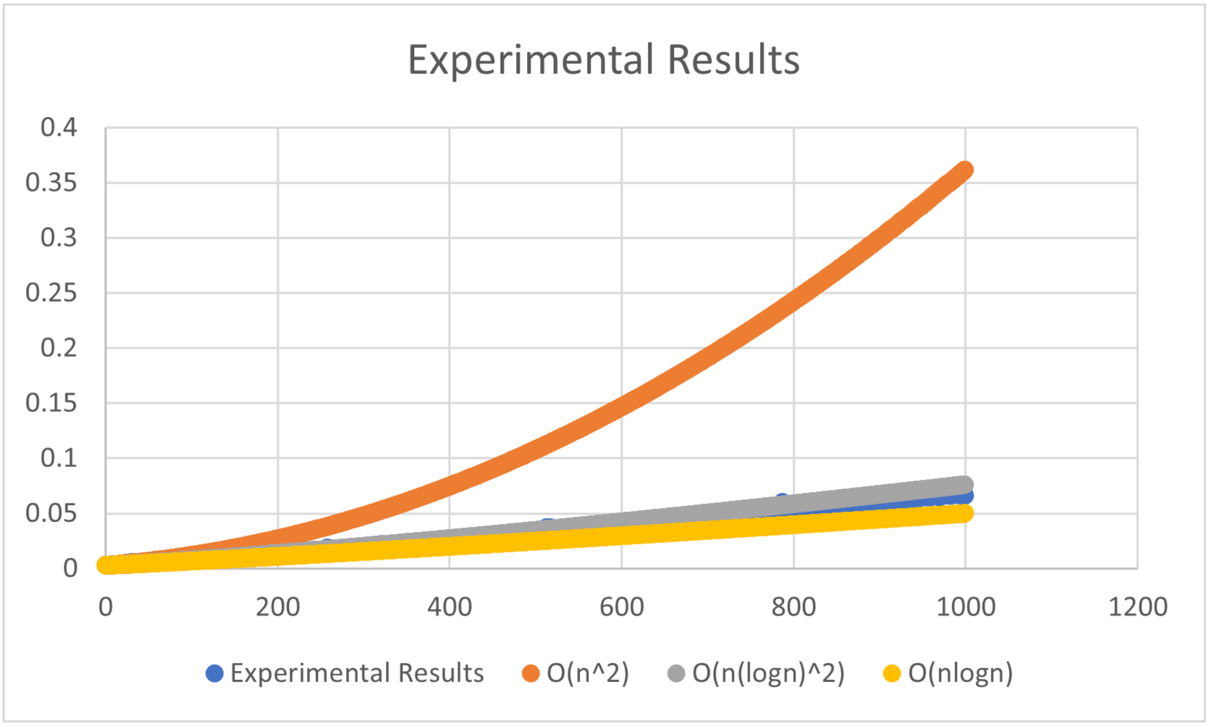
Example of `cp2` Execution

This is the progress bar seen when running the experiments.

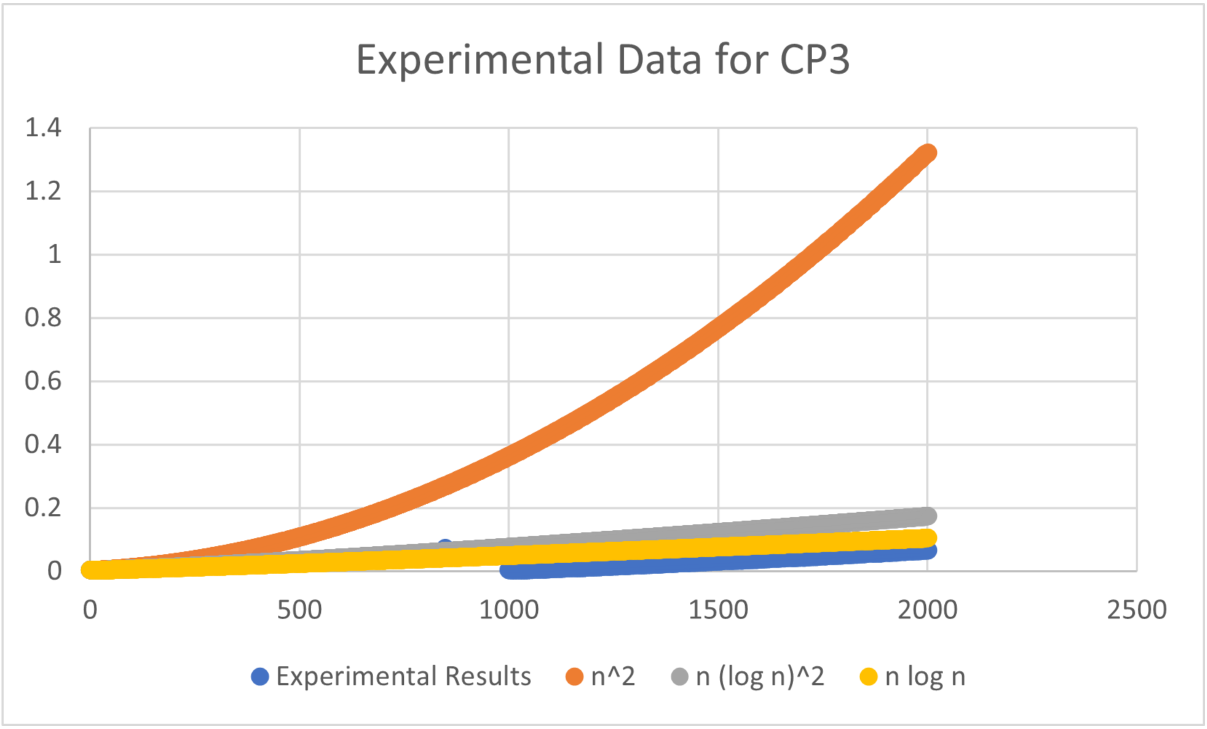
```
Progress: [---------------------------------------------------------->                  ] 60 %
```

Results

The graph of the results obtained can be obtained from the data from `data.csv` contained within project. Excel was used to create the graph presented below. This graphs is a scatter plot. The Experimental data never exceeds n^2. There are 3 noticeable locations where the experimental data exceeds $n(\log n)^2$. The experiment data is always greater than $n \log n$.



The graph of the results obtained can be obtained from the data from `data2.csv` contained within project. Excel was used to create this graph presented below. This graph is a scatter plot. The experimental data never exceeds any of the functions except the is 1 location where a point is noted



Discussion

As can be seen from the experimental data:

$$cp1 \in O(n^2) \qquad cp1 \in O(n(\log n)^2)$$

The 3 points where the experimental data exceeds the $n(\log n)^2$ function are likely to to random variation in runtime and would most likely not appear if: a greater repeat parameter is chosen (e.g. 10).

As can be seen from the experimental data:

$$cp3 \in O(n^2) \qquad cp3 \in O(n(\log n)^2) \qquad cp3 \in O(n \log n)$$

The 1 point of variation can be explained by the fluctuation in clock time.

Conclusion

In this report a 2 variations of the closest points algorithm were implemented using `python3` . Using a class developed the time complexity was tested for `cp1` and `cp3` . The experimental results obtained show that:

$$cp1 \in O(n^2) \qquad cp1 \in O(n(\log n)^2)$$

$$cp3 \in O(n^2) \qquad cp3 \in O(n(\log n)^2) \qquad cp3 \in O(n \log n)$$

References

[1] https://studres.cs.st-andrews.ac.uk/CS3052/Lectures/Week 2/2-3-slides.pdf (slides 11-14)

[2] https://www.geeksforgeeks.org/closest-pair-of-points-using-divide-and-conquer-algorithm/

[3] https://www.geeksforgeeks.org/quickselect-algorithm/

[4] https://ecommons.cornell.edu/bitstream/handle/1813/7460/78-340.pdf?sequence=1&isAllowed=y

[5] https://www.cs.umd.edu/~samir/grant/cp.pdf

[6] https://stackoverflow.com/questions/6169217/replace-console-output-in-python

[7] https://machinelearningmastery.com/how-to-generate-random-numbers-in-python/