Yuhao Zhuang

Homework 1 analysis

Set number	Number of Rectangles	Number of Areas	Ratio of #of rectangles vs # of areas	Time for Linear search (microseconds)	Time for Binary search (microseconds)
1	100	10	10:1	11.025	79.3
2	100	20	5:1	34.187	48.177
3	100	25	4:1	31.378	29.619
4	100	50	2:1	173.521	65.205
5	1000	10	100:1	162.317	500.473
6	1000	250	4:1	3517	596.838
7	1000	500	2:1	7779.08	647.342
8	10000	10	1000:1	1851.8	5036.31
9	10000	2500	4:1	142701	7900.93
10	50000	5000	10:1	$1.74* 10^6$	19893.9
11	100000	10	10000:1	13362.9	36334.8

For linear search, as the number of rectangle inputs increases, the running time also increases by almost the same ratio. For example, with 10 areas and 100, 1000, 10000, and 100000 rectangles, the running time is approximately 11, 162, 1851, and 13362 microseconds respectively. As the number of rectangle input increases by tenfold, the running time increases slightly more than ten times. This shows that the running time of linear search will increase proportionally with the number of rectangles. The same concept applies to the number of areas. As the number of areas increase, the time it takes to search through increases proportionally. For example, with 1000 rectangles (ignore the 100 ones because size not large enough), and 10, 250, and 500 areas, the running time was approximately 162,3517,7779 microseconds respectively. As the area increases by 25 and 2 times, the running time increases about the same. So both number of areas and number of rectangles increases the running time proportionally. This can be explained by the O(#of areas * #of rectangles) running time of the linear search because it goes through every area in the area vector, and maximally every area in the rectangle vector.

For binary search, as the number of rectangle inputs increases, the running time also increases. With 10 areas and 100, 1000, 10000, and 100000 rectangles, the running time is approximately 79, 500, 5036, 36334 microseconds respectively. Although the difference in running time aren't as exact as tenfold each time, it is still pretty close to ten. This is quite surprising because the inner loop of the binary search divides the search size by half each time and more variations could happen. I was expecting to see a O(log(N)) increase in running time. But it can be explained by that the list of rectangles were sorted from smallest to largest beforehand, and the sort function might be a linear growth function, so the increase in running time is seemingly close to be a linear function. And as the number of areas increases, the running time somehow aren't affected by that much. With 1000 rectangles and 10, 250, and 1000 areas, the running time is approximately 500, 596, 674 microseconds respectively, which doesn't really show any linear correlations with the amount increase in number of areas. This could be explained by that although with each added area, the running time should be multiplied proportionally, the binary search has a sort method beforehand, which might take up a lot running time. And since the number of rectangles stays the same, the same amount will be added to running time in this case. And with more areas being searched, the more number of inner binary search will be performed. And since it is a binary search divides search size by half each time, the inner rectangle search is very efficient at handling large amount of data due to its logarithmic growth, so the increase in running time is not very significant compared to linear search.

When there is a small number of areas to search from, binary search seems to be taking about 3 times as much longer than linear search regardless of the increase in the amount of rectangle inputs. (Set 1,5,8,11) This could be explained by the extra sort function before the binary search, as it seems to be a linear growth with the increase in size of rectangles and taking a lot of time for the binary search method. But when the number of areas increases, binary search takes a lot less time than linear search. (Set 1-4, Set 5-9) The sort function of the binary search is constant because there are same number of rectangles, so it doesn't account the difference in running time. The number of areas affects how many times the inner binary search for the rectangles is being runned. The increase in running time is very minimal because it is performing more logarithmic search in the inner loop as the number of area (outer loop) increases. So this makes the binary search very efficient at searching through a large number of areas because the sort function, which I suppose takes a lot time to run, stays constant, but the actual inner search, which is logarithmic,to match the areas takes a lot less time compared to the linear search. So although binary search should take less time to run than linear search because the running time is logarithmic compared to linear, the sort function before binary search adds a lot of running time to the binary search. This makes the linear search more efficient at handling inputs that are very small, where running the sort function is just not worth it and inner binary search won't reduce running time that much, or at handling inputs that have a very large number of rectangles and small number of areas (large ratio of rectangle to area size). Because the sort function before the

binary search is taking a lot of time. So in short, linear search is more efficient at handling inputs where it is not worth it to sort the rectangle inputs before. And for binary search, it is really efficient at handling inputs with more areas (small ratio of rectangle to area size), because with more areas, more inner search functions have to be performed but the running time of the sort function beforehand is still the same. This makes the binary search worth it compared to linear search because the growth of inner search is a lot less since it is growing logarithmically, and the impact of the increase in running time of the sort function is a lot less with more searches run through.

There isn't a specific rectangle to area ratio to determine whether linear or binary search is more efficient. For example, comparing set 3 and 6, which has the same ratio, the running time in set 3 seems to be equal, but in set 6, binary search is almost 7 times faster than linear search. Although generally the smaller the ratio, the more chance binary search is more efficient than linear search, it really depends on the number of areas and more specific - each data sets. We can however, find the ratio for each individual data set. For example with 100 rectangles and 25 areas (Set 3), running time between linear and binary search is equivalent. But overall, there is no specific ratio to determine which is is more efficient, but generally the smaller the ratio, which means greater areas, binary search is more likely to be more efficient than linear search.

References:

Homework prompt
Lecture Slide