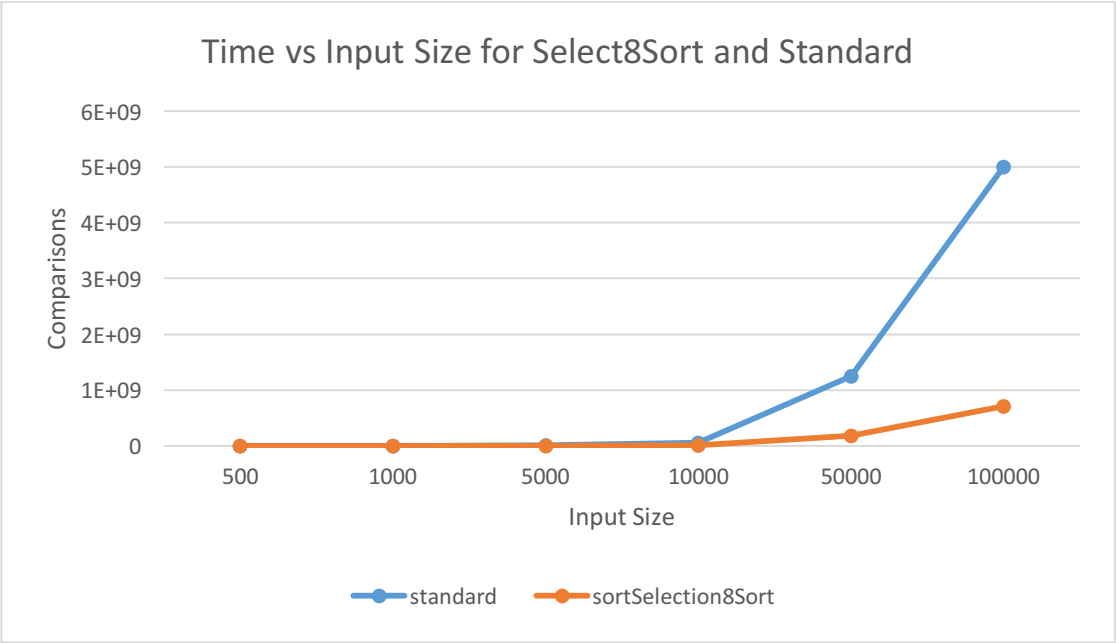
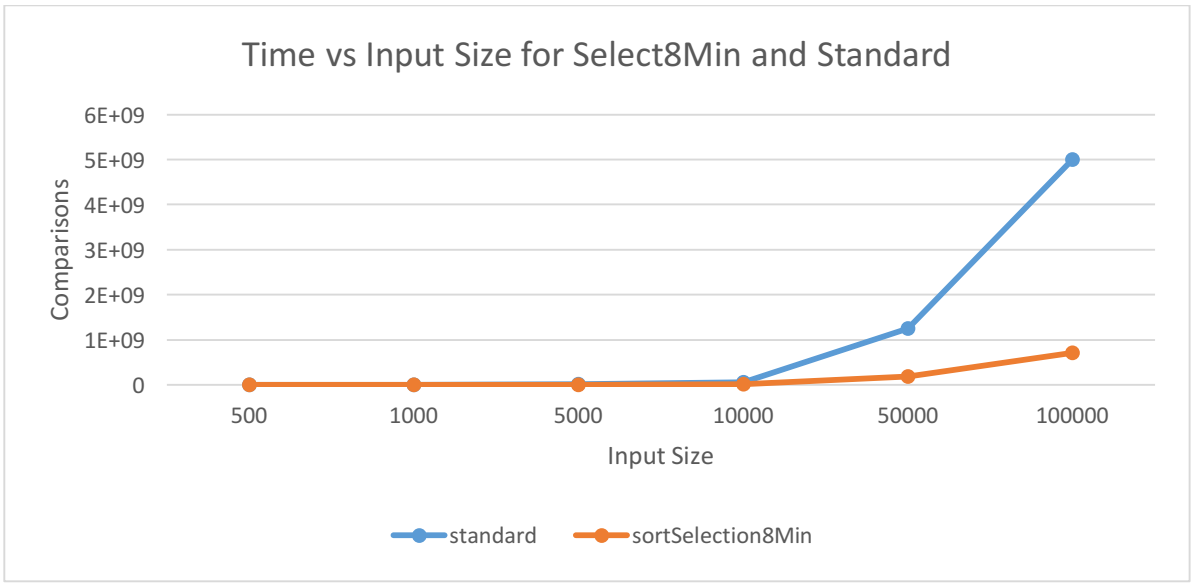
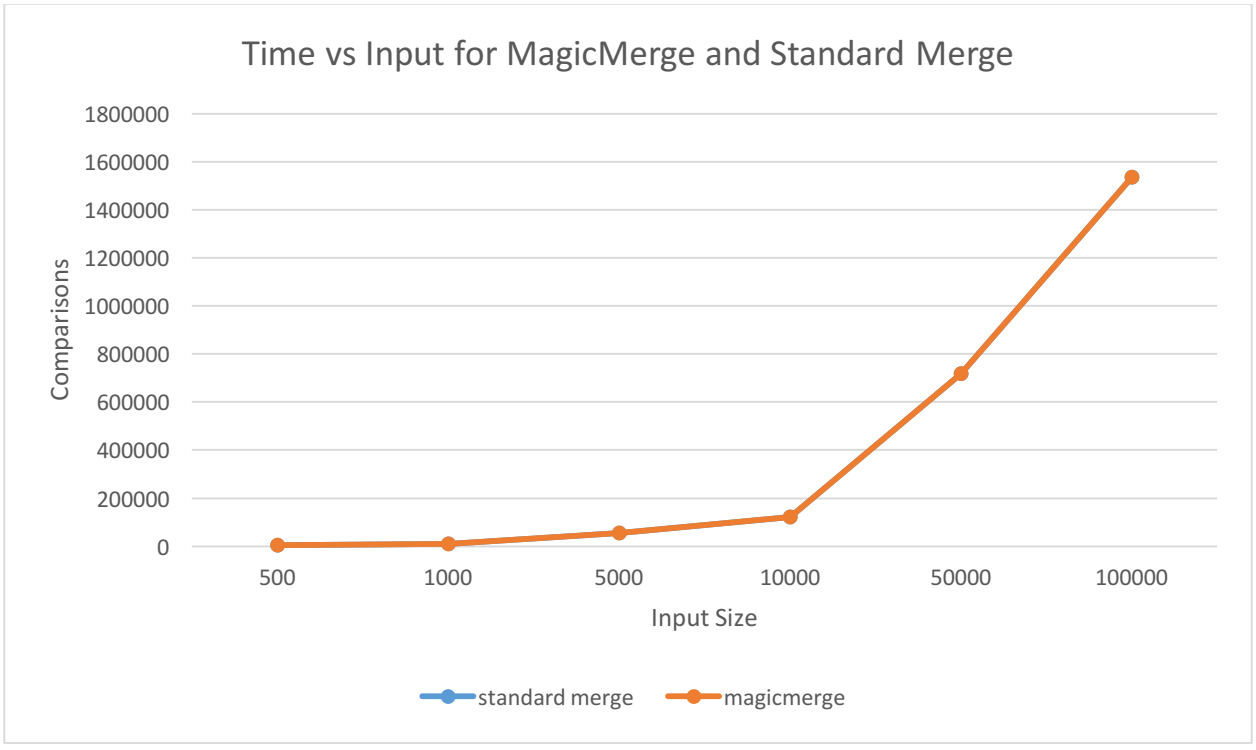
Kazi Rahma
Project 2 Report
Date: 14th October 2016
Collaborators: I discussed the specifications of the analysis with Johanna Collins and Aakriti Mahajan.

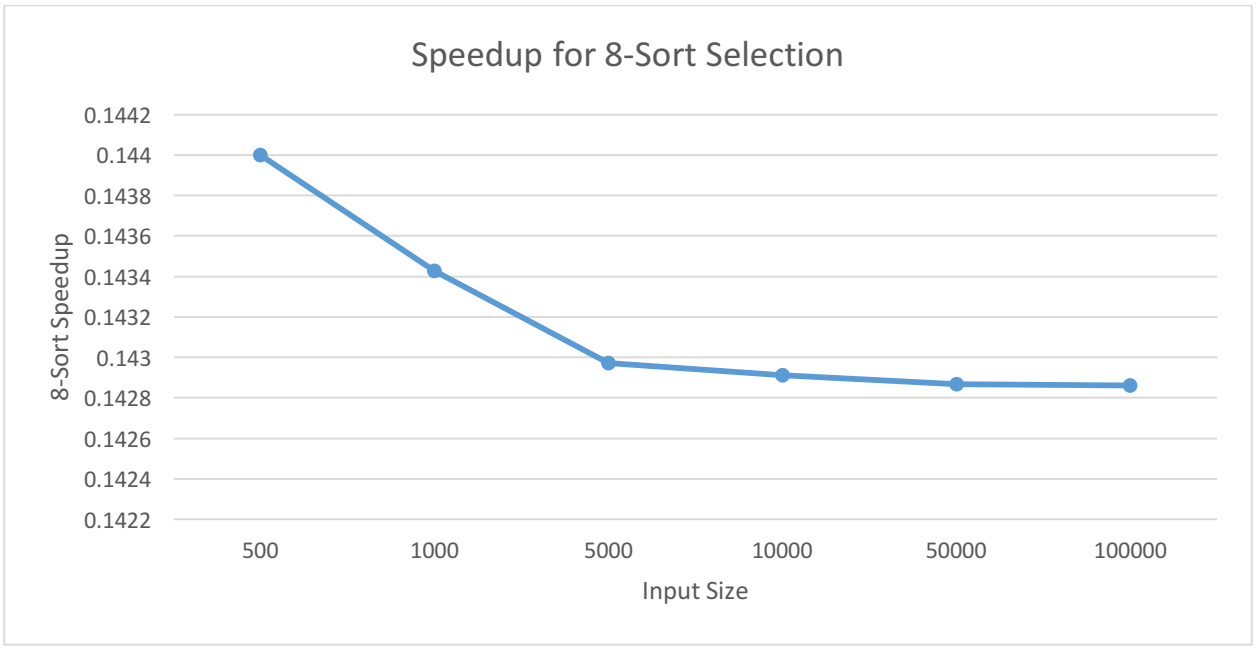
Experimental Analysis

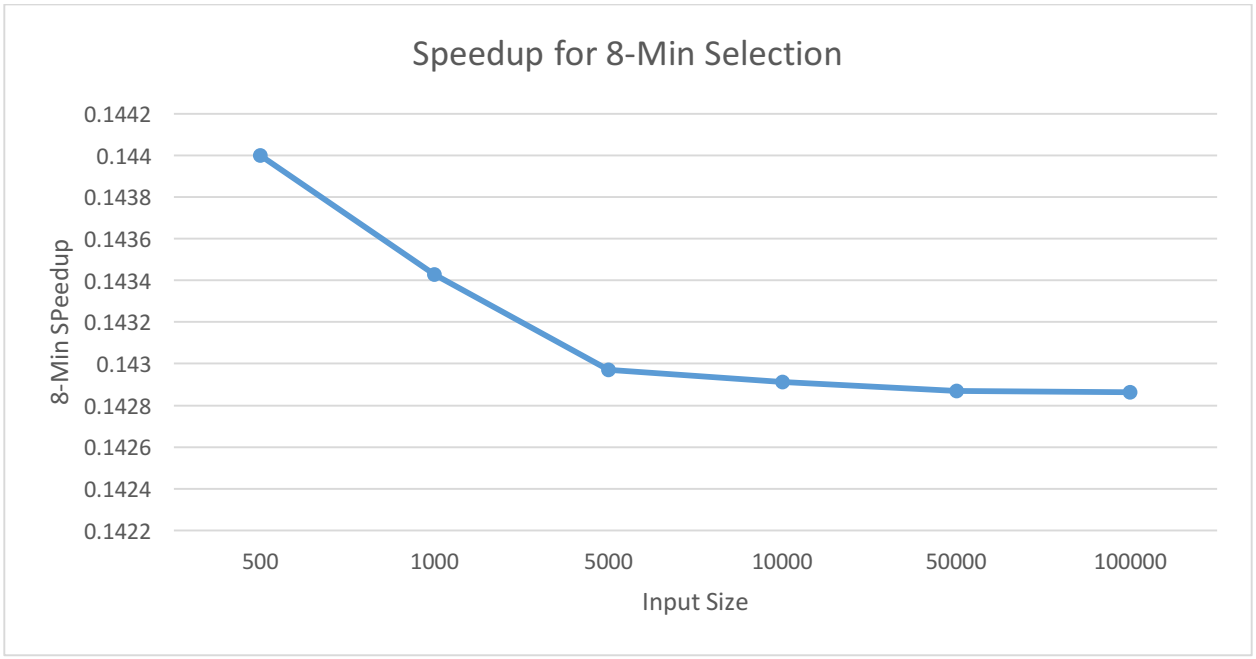
1)

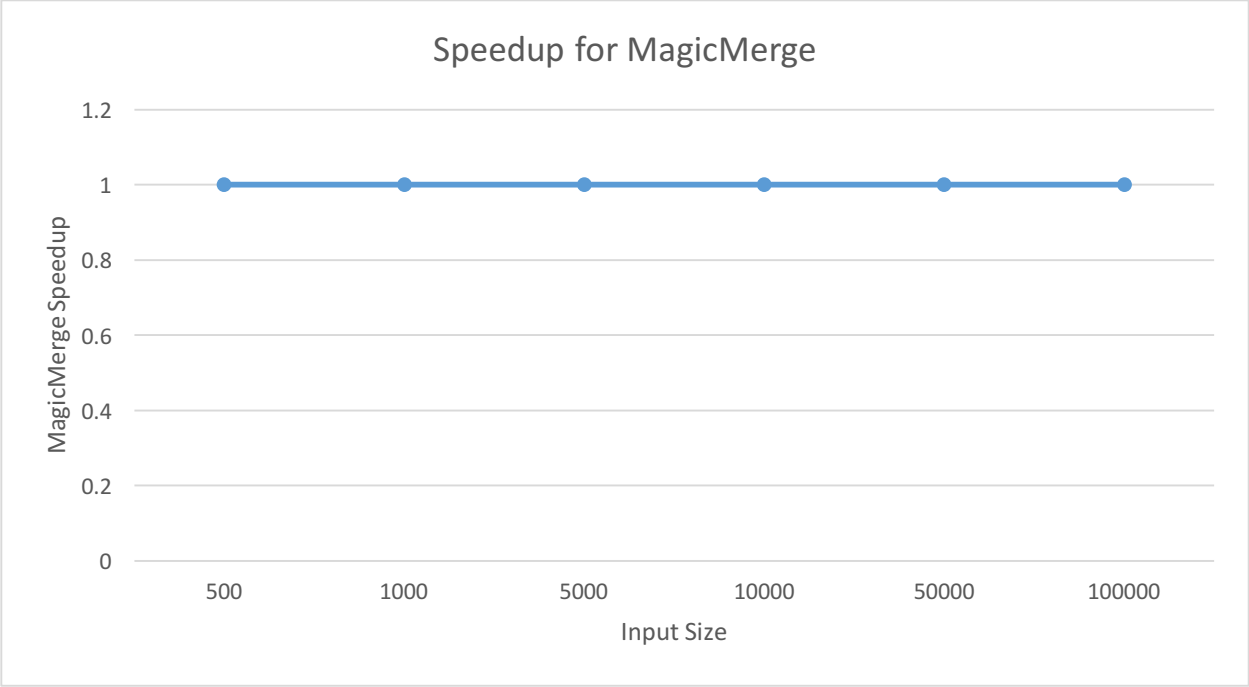



Time vs Input Size for Select8Min and Standard
6E+09
5E+09
4E+09
3E+09
2E+09
1E+09
0
Comparisons
500
1000
5000
10000
50000
100000
Input Size
standard
sortSelection8Min


Time vs Input for MagicMerge and Standard Merge
1800000
1600000
1400000
1200000
1000000
800000
600000
400000
200000
0
Comparisons
500
1000
5000
10000
50000
100000
Input Size
standard merge
magicmerge


Speedup for 8-Sort Selection
8-Sort Speedup
0.1442
0.144
0.1438
0.1436
0.1434
0.1432
0.143
0.1428
0.1426
0.1424
0.1422
500
1000
5000
10000
50000
100000
Input Size


Speedup for 8-Min Selection
8-Min SPeedup
0.1442
0.144
0.1438
0.1436
0.1434
0.1432
0.143
0.1428
0.1426
0.1424
0.1422
500
1000
5000
10000
50000
100000
Input Size



Examining the graphs for speedups for sortSelect8Sort and sortSelect8Min we see that the ratio of $\frac{number\ of\ comparisons\ done\ in\ magic\ box}{number\ of\ comparisons\ done\ in\ standard\ sort}$ is greater for smaller input sizes and lesser for bigger input sizes. The average of the speedups for sortSelect8Sort and sortSelect8Min in my case is the same which is 0.143174225 . The increase in the number of boxes used is not constant in this case. The increase is not constant in the cases for my sortSelect8Sort and sortSelect8Min. The rate of increase actually decreases with the growing size of n.

For the speedup of my magicmerge and standard merge, if I examine the graph then I see that the ratio of $\frac{number\ of\ comparisons\ done\ in\ magic\ box}{number\ of\ comparisons\ done\ in\ standard\ sort}$ = 1 for all input sizes. This is because I am using the same number of comparisons as the standard merge sort.

2) In the case of my algorithms, both the 8-sort and 8-min magic boxes perform with the same level of efficiency. Even though the number of comparisons done by standard selection sort is greater than the number of boxes used by implementing the sortSelect8Sort and sortSelect8Min, both the 8-sort and 8-min magic boxes have the same level of efficiency in my algorithms. However, I believe that the 8-sort magic box would have been able to give more efficient result in terms of comparisons if it was implemented differently in the algorithms. Since the 8-sort magic box is giving the indices in the sorted order, it could have been used more efficiently in the algorithm to sort more than 2 elements at a time.

Analytical Answers

1) For an array of size n, n>=8:

   For n = 8, I am using $\left\lceil \frac{8}{7} \right\rceil = 1$ box.

   For n = 10, I am using $\left\lceil \frac{10}{7} \right\rceil$ = 2 boxes.

   For n = 16, I am using $\left\lceil \frac{16}{7} \right\rceil$ = 3 boxes.

   For n = 64, I am using $\left\lceil \frac{64}{7} \right\rceil$ = 9 boxes.

   For n = 1000, I am using $\left\lceil \frac{1000}{7} \right\rceil$ = 143 boxes.

   For n = 100000, I am using $\left\lceil \frac{100000}{7} \right\rceil$ = 14286 boxes.

   These are some different examples of boxes used for n >= 8.

   If I follow the pattern it tells me that I am using $\left\lceil \frac{n}{7} \right\rceil$ boxes. Furthermore, if I examine my code I see that I am incrementing i by 7 after each iteration, and using the magic box in the next 8 elements starting from i to find the minimum. Hence, I am iterating $\left\lceil \frac{n}{7} \right\rceil$ times and using $\left\lceil \frac{n}{7} \right\rceil$ magic boxes in my mim8min function.

2) For an array of size n, n>= 8, the 8-min box will be used once if the first chunk of 8 elements in unsorted. According to my code, if the first chunk of 8 elements is unsorted then the 8-min box will be called once to check the minimum index and if it is not in index 0 of the chunk of 8 elements, then I return false. So for my code it will depend on the input array. For example, if the input array is 1 2 3 4 5 6 7 8 -10 -9 -8 -7 -6 -5 -4 -3 -2 -1, then according to my code, I will use 2 boxes since the first chunk of 8 elements is sorted. It will take the element in minimum index, which is 1 in this case, and place it in index 7 and start iterating from index 7 to find the minimum index of the next set of 8 elements by calling the magic box again. Hence, I a using 2 boxes in this case. So according to my code, the number of boxes used to find is-sorted will depend on the input array. However, the worst case will be when the input array is already sorted. In this case I will be using n-1 boxes.

   For my isSorted8Sort, I am using the same approach that I am using for my isSorted8Sort. I am going to check my first chunk of 8 elements using 1 eightSort magic box, and if it is not sorted I return false. But if my first chunk is sorted, I use another eightSort magic box to check the next set of elements including the minimum from the first one. This goes on till I reach a chunk which is not sorted. So in this case also, the number of boxes I use will depend on the input array. The worst case will be when the input array is sorted and I use n-1 magic boxes.

3) For an input of size n, n >= 8, my sortSelect8Sort will have a worst case asymptotic performance of O(n^2). I have two for loops for my code. The outer most for loop helps me keep the sorted and unsorted parts of the array separate. My outer for-loop runs n-1 times. The inner for-loop then scans through the unsorted part of my array picking up eight elements and placing them in the 8-sort magic box to get the index of the minimum element. It then takes the minimum with it to the next set of eight elements and finds the minimum of that set. Hence, my inner for-loop runs for $\left\lceil \frac{n}{7} \right\rceil$ times since I am incrementing the loop counter by 7 each time, and my outer loop runs for n-1 times. So in total, in the worst case it will run ($\left\lceil \frac{n}{7} \right\rceil$(n-1)) times. Hence, O($n^2$).

   For my sortSelection8Min, I use the same approach as my sortSelect8Sort. I have an outer for-loop for keeping my sorted and unsorted part of the array separate and it runs for n-1 times. My inner for-loop uses the the eightMin magic box to find the minimum index of the eight elements each time, then carries this minimum to the next set of elements and the finds the minimum in that set. It continues doing it till it has found the minimum in the unsorted part of the array. Then this minimum is placed in the next position of the sorted part of the array. Like my sortSelect8Sort, in the worst case my sortSelect8Min will run ($\left\lceil \frac{n}{7} \right\rceil$(n-1)) times. Hence, O($n^2$).

4) For an array of size n, n>=8, the worst case asymptotic performance of my sortMerge8sort is O(n logn). The base case in my sort function will break the input array to a size of 1 each. Then pass two arrays to my merge function in which I put 1 element from each array into an array of size eight and use the eightSort magic box to find the minimum element and place it in the original array. I could have made my algorithm more efficient if I found a way to sort 8 elements at a time instead of 2 elements at a time. This would have reduced the number of boxes I use for my mergesort.
   In terms of comparison, my mergesort with eightsort magic box uses the same number of comparisons as standard mergesort since I am finding the minimum of two elements using the magic box each time to merge the arrays. If I could find a way to sort more than 2 elements at a time then using magic boxes I could reduce the number of comparisons compared to the standard merge sort.

5) Under the assumption that the 8-sort box is a stable sort box, the sortMerge8Sort is stable. This is because the elements with equal values never move past each other during the sorting process if they were together in the input array. This is because we split the array down to 1 element and merge them comparing each element from the two arrays which we pass to the merge function. Hence, in this process of merging, the elements with equal keys do not move past each other and remain together. Therefore, sortMerge8Sort is stable.

   Under the assumption that the 8-sort box is a stable sort box, the sortSelect8Sort is unstable. In my algorithm, I am finding the minimum of the unsorted part of the array and then moving it to the end of the array until I find the ultimate minimum of the unsorted part of the array. In this process, if two elements with equal values were together in the input array then they move past each other in the sortSelect8Sort before they are put together again in the output array. Hence, sortSelection8Sort is not stable.

6) Based on my implementation of the algorithm using magic boxes, the $\sqrt{n}$ – sort box would require $\left\lceil \frac{n}{\sqrt{n}-1} \right\rceil$ boxes to find the minimum of an input array of size n. However, the worst case asymptotic bound would still be O(n^2) for my selection sort using the $\sqrt{n}$ – sort box because my loops will run for $\left\lceil \frac{n}{\sqrt{n}-1} \right\rceil$(n-1) times For merge sort based on my algorithm, the worst case asymptotic performance would be O(n log n). If we were given a magic box which would take $\sqrt{n}$ elements for an input size of n, it would be more efficient than 8-sort box for inputs of size n>64. For any input less than 64 would result in a root which is less than 8. For example, if we were sorting an array of size 9 then the $\sqrt{n}$ – sort box would only be able to take$\lfloor \sqrt{9} \rfloor$ = 3 elements from the array to find the minimum. Hence, it would require the use of more boxes than the algorithm using the 8-sort box. However, for an input array of size 5000, the $\sqrt{n}$ – sort box would only be able to take $\lfloor \sqrt{5000} \rfloor$ = 71 which would allow us to sort more elements with the use of less number of boxes. Hence, the $\sqrt{n}$ – sort box will be more efficient for inputs of size >64 since it will use lesser number of boxes compared to the 8-sort box. To find the minimum of an input