**Task A**

1. Bubble sort

Bubble sort compares the two adjacent elements till the upper limit value of j. In every comparison, if element j is larger than element (j+1), then swap them. After one loop finishes, the loop starts again from the leftmost element and compares adjacent elements. The upper limit value decreases by 1. Loops end until the upper limit value of j becomes 1. As we can see in the following figure, after each version, the largest value in the unsorted array is moved to the upper limit, just like a bubble floating up.

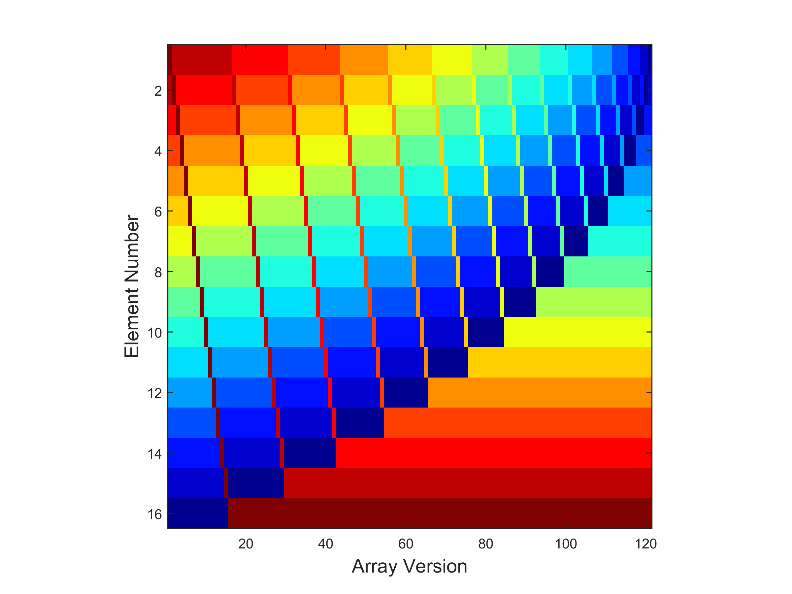


Figure 1: Visualization of Bubble Sort Algorithms for N=16 Elements using Jet Colourmap (values are represented as colour temperature, where blue = low, red = high).

2. Selection sort

Selection sort considers the array as sorted(left) and unsorted(right) parts. First, we create one temporary variable to store the leftmost element V(j) in the unsorted parts. Then, selection sort compares the temporary variable with other unsorted elements in sequence. If the unsorted element is smaller than the temporary variable, we then update the temporary variable. After each loop, substitute V(j) with the temporary variable. Loops end until unsorted array does not exist. As we can see in the following figure, after each version, the sorted array increases.

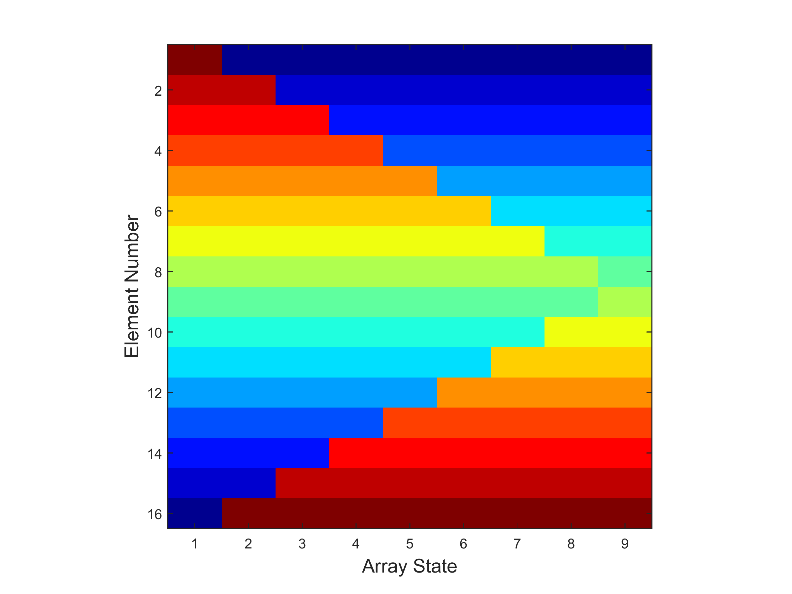


Figure 2: Visualization of Selection Sort Algorithms for N=16 Elements using Jet Colourmap (values are represented as colour temperature, where blue = low, red = high).

3. Shaker sort

Shaker sort is also well known as cocktail sort or bidirectional bubble sort. It consists of forward and backward bubble sort. In the left to right sort loop, Shaker sort compares two adjacent elements in sequence and results in popping the largest element to the upper limit j. In the next loop, Shaker sort starts from (j-1) and compares the two adjacent elements from right to left, resulting in popping the smallest element to the lower limit i. Then the loop starts from (i+1). Loops end until unsorted array does not exist. As we can see in the following figure, after each version, the largest or the smallest value is sorted.

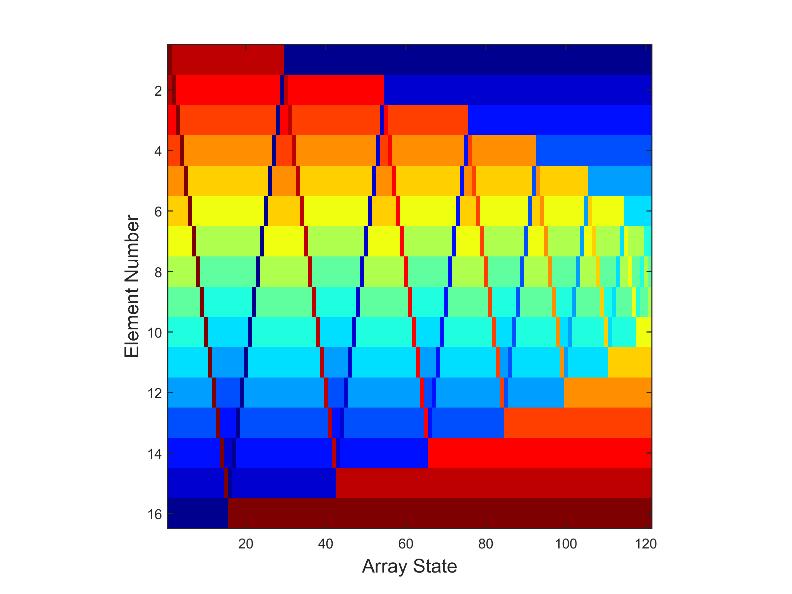


Figure 3: Visualization of Shaker Sort Algorithms for N=16 Elements using Jet Colourmap (values are represented as colour temperature, where blue = low, red = high).

4. Insertion sort

Insertion sort considers the array as sorted(left) and unsorted(right) parts. In the unsorted parts, we select the leftmost element V(j) as a pivot. Then, we compare V(j) with its adjacent left element V(j-1) in the sorted array. If V(j) is smaller than V(j-1), swap them. Continue this process until the left array is sorted. After each loop, we start with the new leftmost element V(j+1). Loops end until unsorted array does not exist. As we can see in the following figure, insertion sort starts locally and gradually sorts the array.

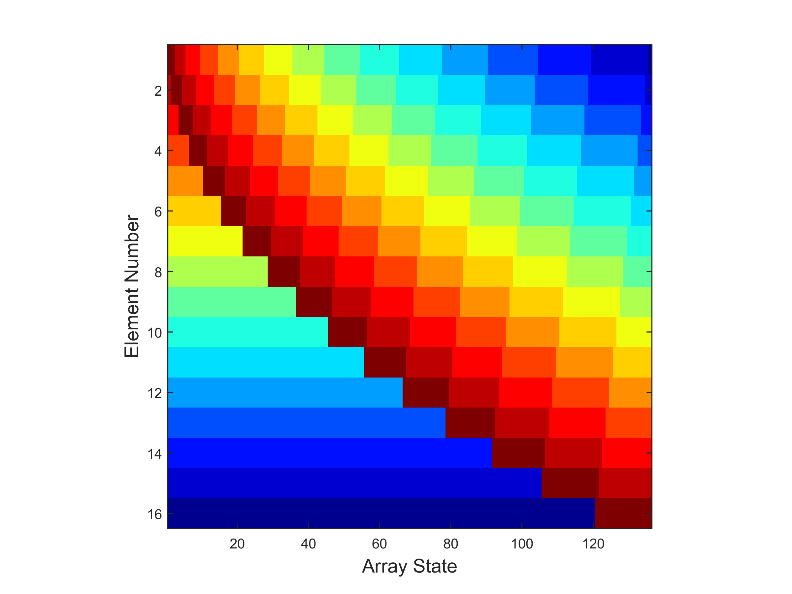


Figure 4: Visualization of Insertion Sort Algorithms for N=16 Elements using Jet Colourmap (values are represented as colour temperature, where blue = low, red = high).

5. Comb sort

Comb sort is an advanced version of bubble sort. Unlike always comparing two adjacent elements in bubble sort, comb sort has the idea of changing window size. It starts with comparing the elements divided by a gap value. After one loop comparison, the gap between two elements for comparing is divided by a factor, e.g. 1.3. After each loop, the gap between two elements continues shrinking. Loops end until the gap becomes 1. As we can see in the following figure, after each version, the largest value in the unsorted array is moved to the upper limit, just like a bubble floating up. In the colormap, we can see that the comparing gap shrinks.

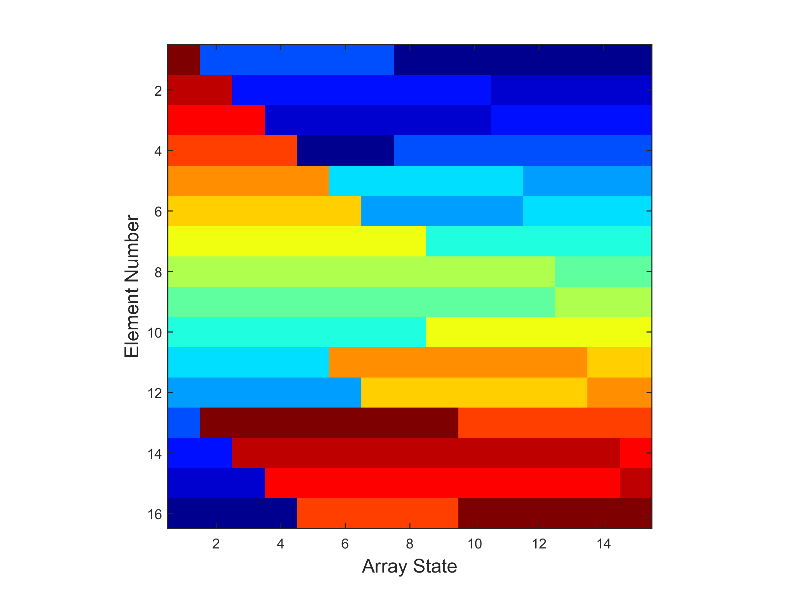


Figure 5: Visualization of Comb Sort Algorithms for N=16 Elements using Jet Colourmap (values are represented as colour temperature, where blue = low, red = high).

**Task B**

1. Linear search

Linear search is the most straight forward search method. Its worst performance and average performance are O(n). Linear search method checks the elements in an array from left to right in sequence. When it finds the target value, then it returns the element index.

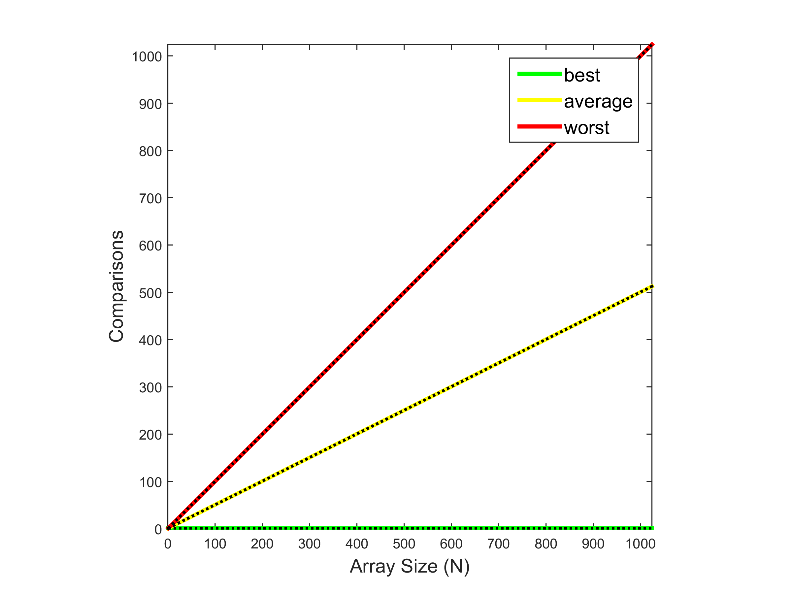


Figure 6: Number of comparisons performed by the linear search algorithm for an array size (N) of between 1 and 1024 elements. The solid green line denotes the observed best case. The solid yellow line denotes the observed average case. The solid red line denotes the observed worst case. The corresponding dotted lines denote the expected best, average and worst cases.

2. Jump search

Jump search is an upgrade of linear search and only suitable for sorted array. First it checks elements based on jump window( sqrt(N) ) distance. As long as the element is larger than target, jump search do linear search in the jump window. When it finds the target value, then it returns the element index.

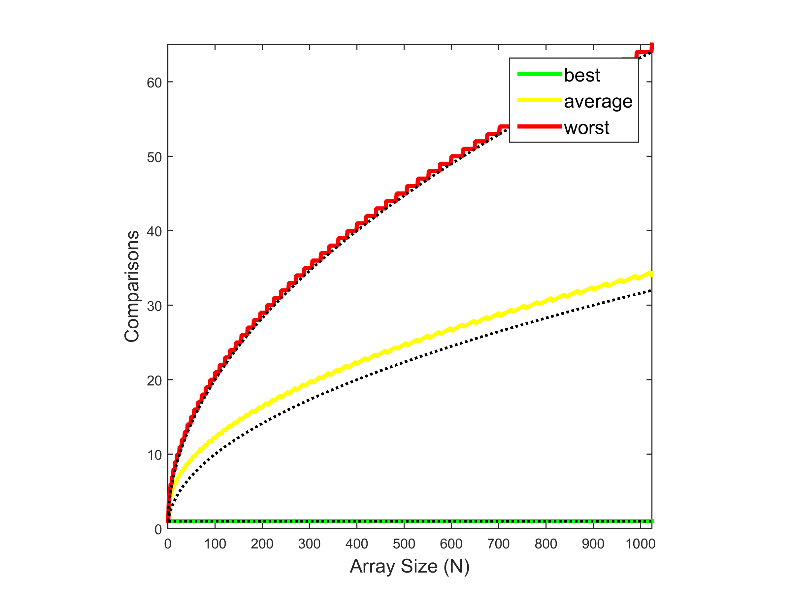


Figure 7: Number of comparisons performed by the jump search algorithm for an array size (N) of between 1 and 1024 elements. The solid green line denotes the observed best case. The solid yellow line denotes the observed average case. The solid red line denotes the observed worst case. The corresponding dotted lines denote the expected best, average and worst cases.

3. Binary search

Binary search is only suitable for sorted array. Every time, it equally divides the array and compare the middle element with the target. If it is larger than the target, do the binary search in the left array. Otherwise, do the binary search in the right array.

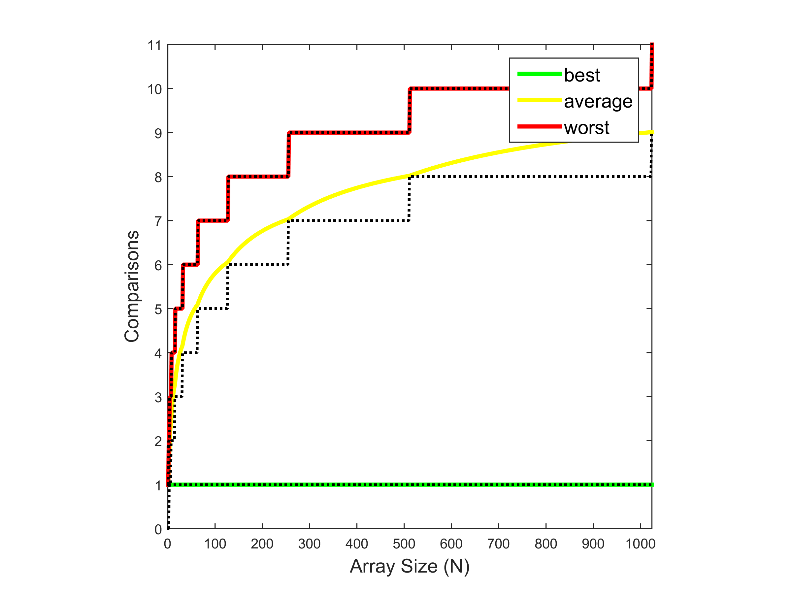


Figure 8: Number of comparisons performed by the binary search algorithm for an array size (N) of between 1 and 1024 elements. The solid green line denotes the observed best case. The solid yellow line denotes the observed average case. The solid red line denotes the observed worst case. The corresponding dotted lines denote the expected best, average and worst cases.

4. Exponential search

Exponential search is an upgrade of binary search and only suitable for sorted array. It starts from the leftmost element and multiple the search element by 2 every step. If element 2n is larger than the target, then do binary search between element 2n and2n-1

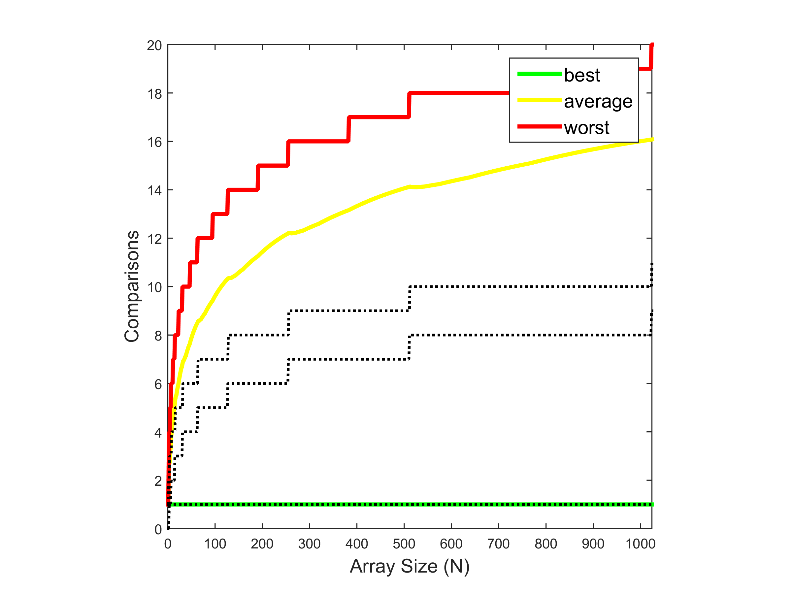


Figure 9: Number of comparisons performed by the exponential search algorithm for an array size (N) of between 1 and 1024 elements. The solid green line denotes the observed best case. The solid yellow line denotes the observed average case. The solid red line denotes the observed worst case. The corresponding dotted lines denote the expected best, average and worst cases.

Exponential search can be used for unlimited array. In the above plot, we find that the numerical results are two times the expected curves. This is because the search process contains two parts. The first stage finds the upper bound for the binary search. This takes O(log(i)) time. While the second stage is a pure binary search, which also takes O(log(i)) time. Therefore, the total time should be 2O(log(i)), which is captured in the above figure.

5. Interpolation search

Interpolation search is based on the idea of interpolation and only suitable for sorted array. It upgrades the check element according to the deviation of the estimated value from the target. Following is the key step:

indexSearch = lowBound + floor((((highBound - lowBound)/(V(highBound)-V(lowBound)))\*(target - V(lowBound))));

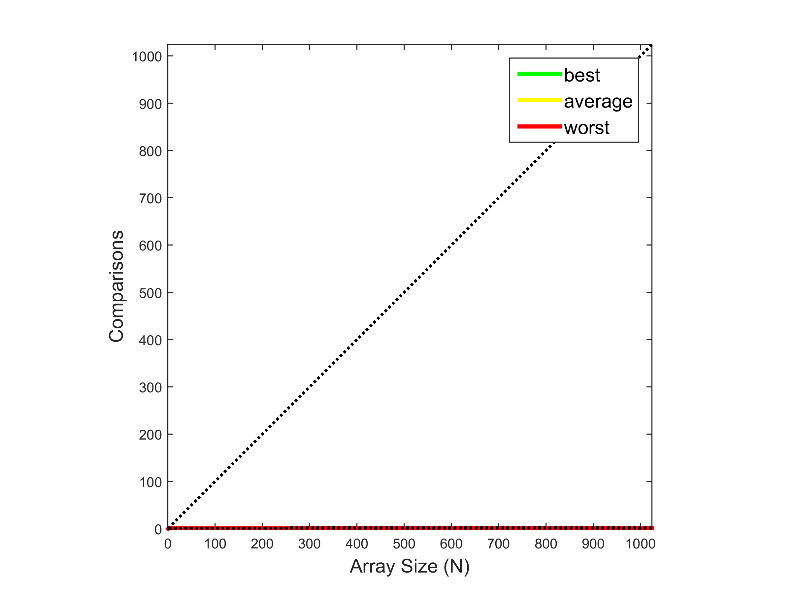


Figure 10: Number of comparisons performed by the interpolation search algorithm for an array size (N) of between 1 and 1024 elements. The solid green line denotes the observed best case. The solid yellow line denotes the observed average case. The solid red line denotes the observed worst case. The corresponding dotted lines denote the expected best, average and worst cases.

In the above plot, we find that the numerical results are far lower than the expected curves. This is because the trail data used here is an uniformly increasing array. The interpolation slope will exactly fit the slope between every two different elements.