1. Full statistics for input size 16

1.1 Alpha sorting

| Alpha | InOrder | ReverseOrder | AlmostOrder | Random |
|---|---|---|---|---|
| runtime | 0 | 0 | 0 | 0 |
| number of comparisoins | 85 | 72 | 85 | 77 |
| number of movements | 136 | 120 | 135 | 131 |

1.2 Beta sorting

| Beta | InOrder | ReverseOrder | AlmostOrder | Random |
|---|---|---|---|---|
| runtime | 0 | 0 | 0 | 0 |
| number of comparisoins | 120 | 120 | 120 | 120 |
| number of movements | 0 | 24 | 3 | 45 |

1.3 Gamma sorting

| Gamma | InOrder | ReverseOrder | AlmostOrder | Random |
|---|---|---|---|---|
| runtime | 0 | 0 | 0 | 0 |
| number of comparisoins | 46 | 55 | 46 | 51 |
| number of movements | 36 | 80 | 39 | 67 |

1.4 Delta sorting

| Delta | InOrder | ReverseOrder | AlmostOrder | Random |
|---|---|---|---|---|
| runtime | 0 | 0 | 0 | 0 |
| number of comparisoins | 32 | 32 | 41 | 47 |
| number of movements | 128 | 128 | 128 | 128 |

1.5 Epsilon sorting

| Epsilon | InOrder | ReverseOrder | AlmostOrder | Random |
|---|---|---|---|---|
| runtime | 0 | 0 | 0 | 0 |
| number of comparisoins | 150 | 158 | 106 | 111 |
| number of movements | 15 | 39 | 18 | 60 |

1.6 Zeta sorting

| Zeta | InOrder | ReverseOrder | AlmostOrder | Random |
|---|---|---|---|---|
| runtime | 0 | 0 | 0 | 0 |
| number of comparisoins | 15 | 120 | 30 | 61 |

| number of movements | 30 | 150 | 45 | 77 |
|---|---|---|---|---|

2. Runtimes for over four input sizes

| Alpha | array size | 16384 | 32768 | 131072 | 262144 | 1048576 | |
|---|---|---|---|---|---|---|---|
| | in order | 1 | 2 | 9 | 18 | 77 | |
| | reverse | 1 | 2 | 9 | 19 | 81 | |
| | almost | 1 | 2 | 10 | 21 | 94 | |
| | random | 2 | 3 | 15 | 31 | 205 | |
| Beta | array size | 1000 | 2000 | 5000 | 10000 | 40000 | |
| | in order | 0 | 1 | 10 | 43 | 749 | |
| | reverse | 1 | 2 | 14 | 53 | 973 | |
| | almost | 0 | 3 | 16 | 62 | 1058 | |
| | random | 1 | 2 | 15 | 62 | 1198 | |
| Gamma | array size | 16384 | 32768 | 131072 | 262144 | 1048576 | |
| | in order | 1 | 1 | 4 | 8 | 36 | |
| | reverse | 1 | 2 | 8 | 18 | 78 | |
| | almost | 1 | 1 | 5 | 12 | 51 | |
| | random | 1 | 3 | 17 | 34 | 152 | |
| Delta | array size | 16384 | 32768 | 131072 | 262144 | 1048576 | |
| | in order | 2 | 3 | 10 | 21 | 80 | |
| | reverse | 1 | 2 | 10 | 21 | 85 | |
| | almost | 2 | 3 | 13 | 27 | 125 | |
| | random | 3 | 5 | 22 | 42 | 184 | |
| Epsilon | array | 1000 | 2000 | 5000 | 10000 | 13000 | 14000 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | size | | | | | | |
| | in order | 0 | 2 | 13 | 54 | 95 | overflow |
| | reverse | 2 | 7 | 43 | 174 | 351 | overflow |
| | almost | 0 | 0 | 0 | 1 | 1 | 1 |
| | random | 0 | 0 | 1 | 1 | 1 | 2 |
| Zeta | array size | 1000 | 2000 | 5000 | 10000 | 40000 | |
| | in order | 0 | 0 | 0 | 0 | 0 | |
| | reverse | 1 | 2 | 12 | 49 | 873 | |
| | almost | 0 | 0 | 1 | 3 | 53 | |
| | random | 0 | 1 | 6 | 24 | 567 | |

3. Worst-case asymptotic run-time

| implemantation | Alpha | Beta | Gamma | Delta | Epsilon | Zeta |
|---|---|---|---|---|---|---|
| big-O runtime | O(n*log(n)) | $O(n^2)$ | O(n*log(n)) | O(n*log(n)) | $O(n^2)$ | $O(n^2)$ |

After identifying the Gamma as quick sort optimized, I want to mention that the worst-case runtime of Gamma should be $O(n^2)$ with really bad input order that every median-of-three for pivot selection leads to the pivot as the smallest or largest element in the unsorted part of the array. This scenario cannot be seen from the runtime of question 2.

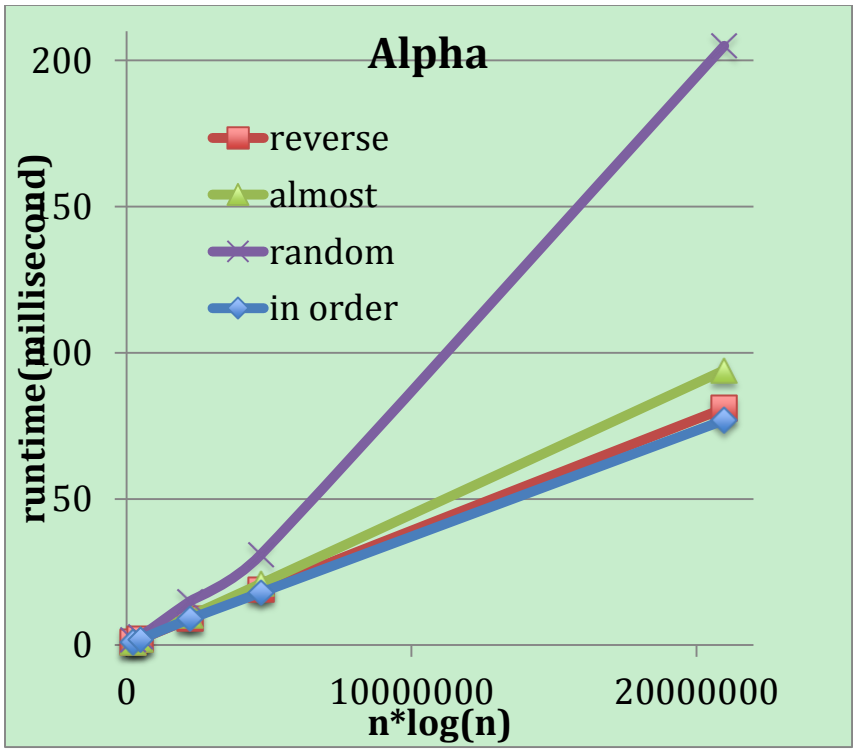
4. Identifying the algorithms

4.1 Alpha

Alpha is heap sort for the following reasons.

4.1.1 runtime

From the data of answer for question 1 and 2, and the chart below, it is shown that the runtime for Alpha is O(nlog(n)). Thus, Alpha is among the heap sort, merge sort, and quick sort (optimized).




4.1.2 Behaviors with different input orders

Since we are using max heap in heap sort, and the reversed array is most likely kind of in the order of max heap among all the input orders. Thus the reversed order input will lead to the smallest number of comparisons and movements during the build heap.

While in the delete max process, the comparison and movements number should not vary much among different input orders.

Thus the total number of comparisons as well as movements should be the smallest with reversed order among all the input orders with input size of 16.

Alpha is the only method that comes with smallest number of comparisons and movements when input is reversed with input size of 16.

4.1.3 runtime for different input order

The time of building max heap in-place is O(n), while the total time for delete max should be O(nlogn). The total time for heap sort should be O(n)+O(nlogn). Since nlogn >n, when n is large

enough, the O(n) part of the total run time would not matter vary much compared to the O(nlogn) part. As said in 4.1.2, the time of build heap part differs among different input orders, while the delete max part doesn't differ a lot. Thus the total runtime with reverse order input wouldn't be much less when n comes large. That's shown in the run time of Alpha when input size comes larger than 100000.

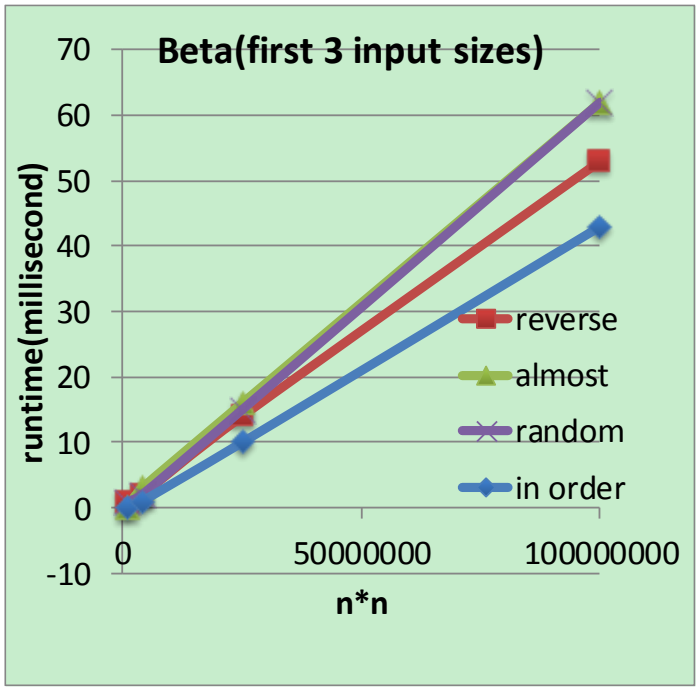
## 4.2 Beta

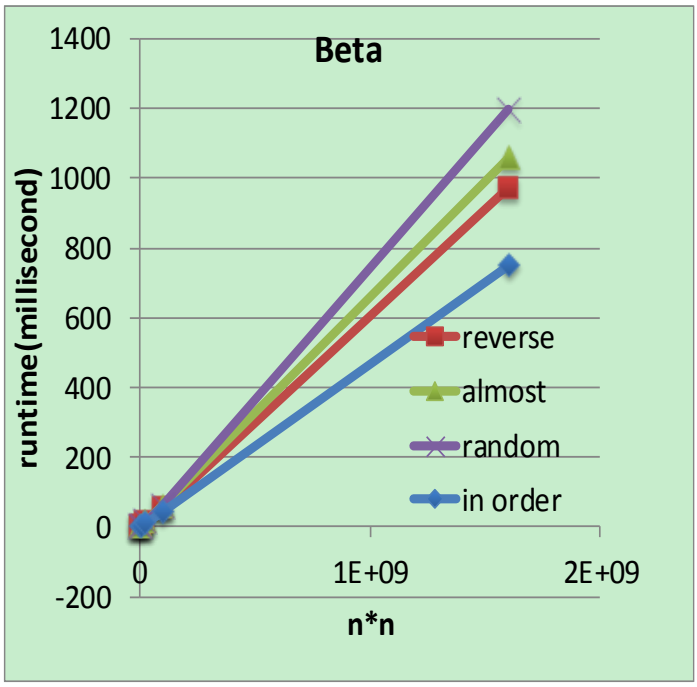
Beta is selection sorting for the following reasons.

### 4.2.1 runtime

As we can see in the answer to question 2 and 3, the run time for Beta is $O(n^2)$, which can be clearly seen from the chart below of runtime vs. $n^2$, where n is the input size.

To see clearly runtime with input sizes, I plotted the graph with only the first input sizes seen as the first graph. From the two graphs, we can tell the runtime increases quadratically with input size. Thus Beta must be among the selection, insertion, and quick sort(simple).




### 4.2.2 comparisons and movements for input size of 16

In selection sorting algorithm, when the input array is already sorted, for each step, the smallest element is found by (n-1) comparisons, where n is the length of unsorted part. The smallest element will be found to be the beginning element of the unsorted part. Thus, no movement is needed for sorted input. Hence, the numbers of comparisons and movements for input sorted array of size 16 should be:
Comparisons: 15+14+13+…+1=120,
Movements: 0.
The numbers are exactly the same as the output of Beta implementation.

#### 4.2.3 comparisons for various input order

From the mechanism of selection sort, we know each time to select the smallest element in the unsorted part, n comparisons are needed regardless of the input order. Thus for every order of input with size 16, the number of comparisons should all be exactly the same as 120, while the movements differ from each other with different input order, which is the case of Beta.

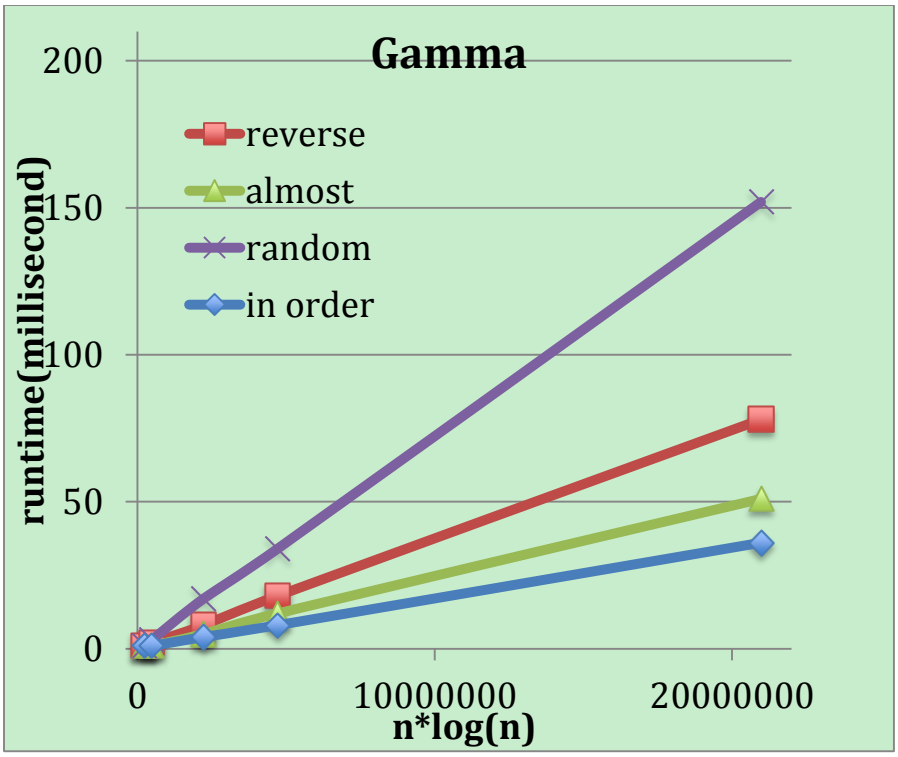
To conclude, the Beta is selection sorting.

### 4.3 Gamma

Gamma is quick sort optimized for the following reasons.

#### 4.3.1 runtime

From the data of answer for question 1 and 2, and the chart below, it is shown that the runtime for Gamma is O(nlog(n)). Thus, Gamma is among the heap sort, merge sort, and quick sort (optimized).




4.3.2 Behaviors for various input order

Since optimized quick sort use the median-of-three for pivot selection, the pivot would be nearly the best choice when the input is in order or reversed. Thus the numbers of comparisons that are made should be nearly the same when the input is in order or reversed, While more movements are needed with reversed input than any other input order. That is exactly what Gamma behaves like.

4.3.3 runtime for reversed input

Since the pivot selection of optimized quick sort is so good for reversed input that it should take the least time for reversed input among all the methods. Gamma takes the least time for reversed input.

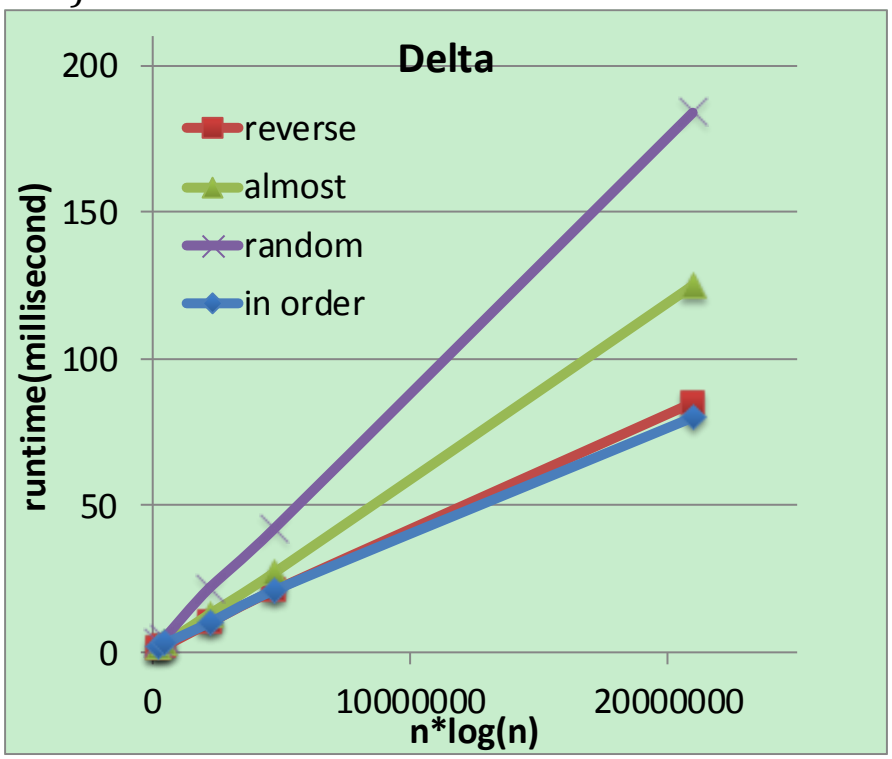
4.4 Delta

Delta is merge sort for the following reasons.

4.4.1 runtime

From the data of answer for question 1 and 2, and the chart below, it is shown that the runtime for Delta is O(nlog(n)).

Thus, Delta is among the heap sort, merge sort, and quick sort (optimized).




4.4.2 number of movements

Since for step to split the input array to one element separately, we move all the elements in the array to a new array, that makes 16 movements each step for an input of 16. Since $16=2^3$, it takes 3+1=4 steps to split the 16 elements to one single element separate. Same things happens when merging the elements to the original arrays. So the total movements it takes in merge sort for an input size is: 16*(4*2)=128 regardless of the input order.

4.4.3 number of movements and comparisons

More importantly, the movement made in the whole process described above of splitting and merging the array, is operating regardless of the input order since each movement made is moving the element to a new temporary array and moving back to the original array in the last step. Thus the movements should be the same with different input order and they should all be 128 for input size of 16. Same number of

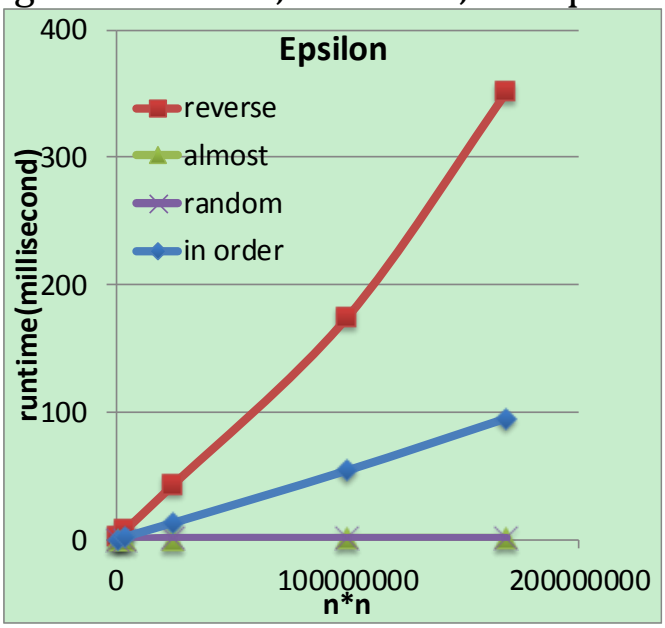
movements with different input orders is a significant characteristic for merge sort.
All the characteristics described above fit the result for Delta.

## 4.5 Epsilon

Epsilon is quick sort (simple).

### 4.5.1 Runtime

As we can see in the answer to question 2 and 3, the run time for Epsilon is $O(n^2)$, which can be clearly seen from the chart below of runtime vs. $n^2$, where n is the input size. Thus Epsilon must be among the selection, insertion, and quick sort(simple).




### 4.5.2 different performances with different inputs

Since the quick sort (simple) uses the first element in the range of the array as the pivot for partitioning, it would end up in taking the smallest element with inorder input and the largest element with reverse input. Both situations lead to more comparisons made than that with random input.
Epsilon is the only function with runtime of $O(n^2)$ that makes exactly same number of comparisons with inorder and reverse input order, which is larger than that of almost ordered and random input.
When it comes to the number of movements, although a bad pivot is chosen with inorder input, the simple quick sort don't

need to make extra movements except for the necessary movements of the algorithm itself. Thus the movements should be the least with inorder input among all the input orders. This is also the way in which the Epsilon behaves.
The poor pivot choosing process for in order input also makes simple quick sort the only method that come with longer run time with inorder input than random input and almost input order. This also helps to identify Epsilon is simple quick sort.

#### 4.5.3 Overflow

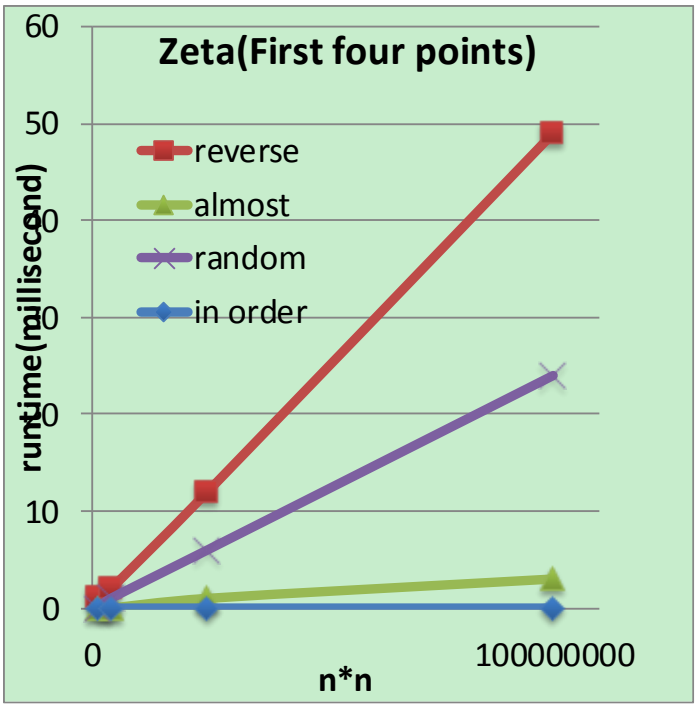
Overflow occurs in Epsilon when the input size is same or larger than 14000 with inorder and reverse order. It is the only method causes overflow at that input size. The simple quick sort calls 'simple quick sort' function recursively O(n) times due to the bad pivot chosen with inorder and reverse order, which is the most recursive times called among all the methods if they are recursive.

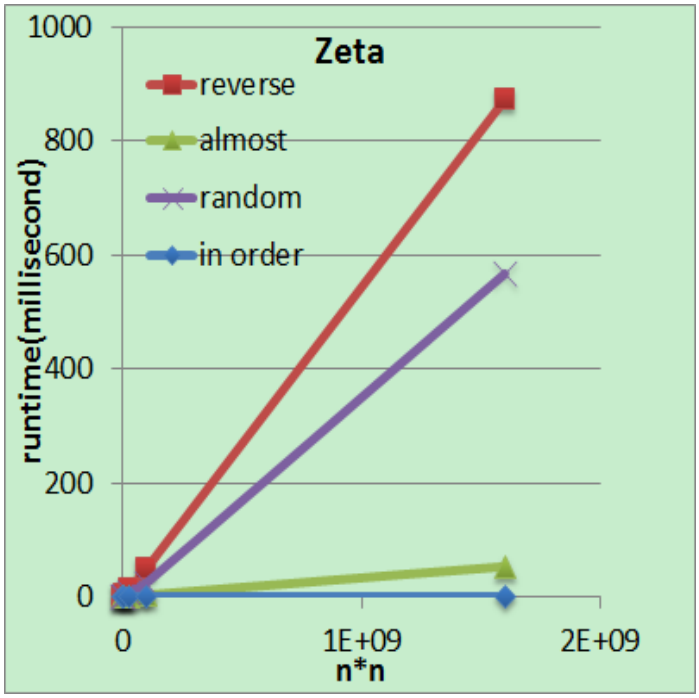
### 4.6 Zeta

Zeta is insertion sort for the following reasons.

#### 4.6.1 Runtime

From the charts below, we can see the runtime for Zeta is $O(n^2)$. Thus Zeta must be among the selection, insertion, and quick sort(simple).




4.6.2 number of comparisons

In insertion sort, for each element, compare it with the elements in the sorted part. Thus, if the input array is already sorted, the element would be compared with the last element (which is also the largest element in the sorted part) in the sorted array and found the new element is larger, then no more comparison is need for this new element. Then the new element would be settled down where it was. Thus only one comparison is need for each elements except the first element, which needs no comparison. The total number of comparisons made for a sorted input of 16 is 16-1=15.

While for reversed input, each element need to be compared with every element all the way toward to the first element in the sorted part. Thus the total number of comparisons made for a reversed input of 16 is 1+2+3+…+15=120.

The number of comparisons for both ordered and reverse input accords with those of insertion sort. Thus the Zeta is insertion sort.