1.

A)

Last:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
91	37	42	38	3	9	62	10	21	8	34	19	6	18	21	25
3	9	10	21	8	19	6	18	21	25	34	37	62	42	38	91
3	9	10	21	8	19	6	18	21	25	34	37	62	42	38	91
3	9	10	8	6	18	21	19	21	25	34	37	62	42	38	91
3	6	10	8	9	18	21	19	21	25	34	37	62	42	38	91
3	6	10	8	9	18	21	19	21	25	34	37	62	42	38	91
3	6	8	9	10	18	21	19	21	25	34	37	62	42	38	91
3	6	8	9	10	18	21	19	21	25	34	37	62	42	38	91
3	6	8	9	10	18	21	19	21	25	34	37	62	42	38	91
3	6	8	9	10	18	19	21	21	25	34	37	62	42	38	91
3	6	8	9	10	18	19	21	21	25	34	37	62	42	38	91
3	6	8	9	10	18	19	21	21	25	34	37	62	42	38	91
3	6	8	9	10	18	19	21	21	25	34	37	38	42	62	91
3	6	8	9	10	18	19	21	21	25	34	37	38	42	62	91
3	6	8	9	10	18	19	21	21	25	34	37	38	42	62	91
3	6	8	9	10	18	19	21	21	25	34	37	38	42	62	91
3	6	8	9	10	18	19	21	21	25	34	37	38	42	62	91
3	6	8	9	10	18	19	21	21	25	34	37	38	42	62	91

First:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
91	37	42	38	3	9	62	10	21	8	34	19	6	18	21	25
25	37	42	38	3	9	62	10	21	8	34	19	6	18	21	91
21	3	9	10	21	8	19	6	18	25	34	62	38	37	42	91
18	3	9	10	21	8	19	6	21	25	34	62	38	37	42	91
6	3	9	10	8	18	19	21	21	25	34	62	38	37	42	91
3	6	9	10	8	18	19	21	21	25	34	62	38	37	42	91
3	6	9	10	8	18	19	21	21	25	34	62	38	37	42	91
3	6	8	9	10	18	19	21	21	25	34	62	38	37	42	91
3	6	8	9	10	18	19	21	21	25	34	62	38	37	42	91
3	6	8	9	10	18	19	21	21	25	34	62	38	37	42	91
3	6	8	9	10	18	19	21	21	25	34	62	38	37	42	91
3	6	8	9	10	18	19	21	21	25	34	62	38	37	42	91
3	6	8	9	10	18	19	21	21	25	34	62	38	37	42	91
3	6	8	9	10	18	19	21	21	25	34	42	38	37	62	91

3	6	8	9	10	18	19	21	21	25	34	37	38	42	62	91
3	6	8	9	10	18	19	21	21	25	34	37	38	42	62	91
3	6	8	9	10	18	19	21	21	25	34	37	38	42	62	91
3	6	8	9	10	18	19	21	21	25	34	37	38	42	62	91

Random:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
91	37	42	38	3	9	62	10	21	8	34	19	6	18	21	25
3	9	10	21	8	19	6	18	21	91	34	37	62	42	25	38
3	9	8	6	10	19	21	18	21	91	34	37	62	42	25	38
3	6	8	9	10	19	21	18	21	91	34	37	62	42	25	38
3	6	8	9	10	19	21	18	21	91	34	37	62	42	25	38
3	6	8	9	10	19	21	18	21	91	34	37	62	42	25	38
3	6	8	9	10	19	21	18	21	91	34	37	62	42	25	38
3	6	8	9	10	19	18	21	21	91	34	37	62	42	25	38
3	6	8	9	10	18	19	21	21	91	34	37	62	42	25	38
3	6	8	9	10	18	19	21	21	91	34	37	62	42	25	38
3	6	8	9	10	18	19	21	21	34	25	37	62	42	91	38
3	6	8	9	10	18	19	21	21	25	34	37	62	42	91	38
3	6	8	9	10	18	19	21	21	25	34	37	62	42	91	38
3	6	8	9	10	18	19	21	21	25	34	37	62	42	38	91
3	6	8	9	10	18	19	21	21	25	34	37	38	42	62	91
3	6	8	9	10	18	19	21	21	25	34	37	38	42	62	91
3	6	8	9	10	18	19	21	21	25	34	37	38	42	62	91
3	6	8	9	10	18	19	21	21	25	34	37	38	42	62	91

C)
Time complexity: O(n^2)
Space complexity: O(log(n))

D)
The time and space complexity is actually the same and the result is the same too.

2.
A)
Create the array:

91	37	42	38	3	9	62	10	21	8	34	19	6	18	21	25

Insertion sort compares the first two elements, 37 is then in the sorted list:

37	91	42	38	3	9	62	10	21	8	34	19	6	18	21	25

Insertion sort moves ahead and compares 42 with 91, after changing their position, compare 37 and 42. 37 and 42 are then in the sorted list:

37	42	91	38	3	9	62	10	21	8	34	19	6	18	21	25

repeat the steps:

37	38	42	91	3	9	62	10	21	8	34	19	6	18	21	25

3	37	38	42	91	9	62	10	21	8	34	19	6	18	21	25

3	9	37	38	42	91	62	10	21	8	34	19	6	18	21	25

3	9	37	38	42	62	91	10	21	8	34	19	6	18	21	25

3	9	10	37	38	42	62	91	21	8	34	19	6	18	21	25

3	9	10	21	37	38	42	62	91	8	34	19	6	18	21	25

3	8	9	10	21	37	38	42	62	91	34	19	6	18	21	25

3	8	9	10	21	34	37	38	42	62	91	19	6	18	21	25

3	8	9	10	19	21	34	37	38	42	62	91	6	18	21	25

3	6	8	9	10	19	21	34	37	38	42	62	91	18	21	25

3	6	8	9	10	18	19	21	34	37	38	42	62	91	21	25

3	6	8	9	10	18	19	21	21	34	37	38	42	62	91	25

3	6	8	9	10	18	19	21	21	25	34	37	38	42	62	91

C)
Time complexity: O(n^2)
Space complexity: O(1)

D)
The time complexity is the same as quicksort in the worst case because in the worst case, they both need to compare each element to every other element. But in the best case, InsertionSort is better than QuickSort. In the best case, insertion sort compares only one time each loop for all of the n-1 loops. So the time complexity would be Ω(n). But for quicksort, each layer for the comparing would cost n time. and there are log(n) layers. So the time complexity would be Ω(n*log(n)).
The space complexity is also not the same. The InsertionSort only cost the space while comparing. There is no recursion in the algorithm. So the space complexity is not relevant to

n which is O(1). However, in the worst case, the recursion cost of quicksort is the same in each layer of recursion, so the space complexity is the same as the depth of the recursion which is log(n).

3.

Create the array:

91	37	42	38	3	9	62	10	21	8	34	19	6	18	21	25

The array size is less than MinRun, which is 32. So the array is sorted by insertion sort and no merge is used in this case.

Insertion sort compares the first two elements, 37 is then in the sorted list:

37	91	42	38	3	9	62	10	21	8	34	19	6	18	21	25

Insertion sort moves ahead and compares 42 with 91, after changing their position, compare 37 and 42. 37 and 42 are then in the sorted list:

37	42	91	38	3	9	62	10	21	8	34	19	6	18	21	25

repeat the steps:

37	38	42	91	3	9	62	10	21	8	34	19	6	18	21	25

3	37	38	42	91	9	62	10	21	8	34	19	6	18	21	25

3	9	37	38	42	91	62	10	21	8	34	19	6	18	21	25

3	9	37	38	42	62	91	10	21	8	34	19	6	18	21	25

3	9	10	37	38	42	62	91	21	8	34	19	6	18	21	25

3	9	10	21	37	38	42	62	91	8	34	19	6	18	21	25

3	8	9	10	21	37	38	42	62	91	34	19	6	18	21	25

3	8	9	10	21	34	37	38	42	62	91	19	6	18	21	25

3	8	9	10	19	21	34	37	38	42	62	91	6	18	21	25

3	6	8	9	10	19	21	34	37	38	42	62	91	18	21	25

3	6	8	9	10	18	19	21	34	37	38	42	62	91	21	25

3	6	8	9	10	18	19	21	21	34	37	38	42	62	91	25

3	6	8	9	10	18	19	21	21	25	34	37	38	42	62	91

C)
Time complexity: O(n*log(n))
Space complexity: O(n)

D)
In this case, the size of the array is smaller than MinRun which is 32. So timsort for this case is the same as insertionsort.

E)
Similar to the Question-D, the comparison of TimSort and quicksort in this case is the same as the comparison of InsertionSort and quicksort. So the result is the same as question-2-D.

4.
A)
Create the array:

91	37	42	38	3	9	62	10	21	8	34	19	6	18	21	25

for i from 0 to 14, compare i and i+1. Switch i and i+1 if i > i+1

37	42	38	3	9	62	10	21	8	34	19	6	18	21	25	91

Then, for i from 0 to 13, compare i and i+1. Switch i and i+1 if i > i+1

37	38	3	9	42	10	21	8	34	19	6	18	21	25	62	91

Then, for i from 0 to 12, compare i and i+1. Switch i and i+1 if i > i+1

37	3	9	38	10	21	8	34	19	6	18	21	25	42	62	91

Repeat the steps:

3	9	37	10	21	8	34	19	6	18	21	25	38	42	62	91

3	9	10	21	8	34	19	6	18	21	25	37	38	42	62	91

3	9	10	8	21	19	6	18	21	25	34	37	38	42	62	91

3	9	8	10	19	6	18	21	21	25	34	37	38	42	62	91

3	8	9	10	6	18	19	21	21	25	34	37	38	42	62	91

3	8	9	6	10	18	19	21	21	25	34	37	38	42	62	91

3	8	6	9	10	18	19	21	21	25	34	37	38	42	62	91

3	8	6	9	10	18	19	21	21	25	34	37	38	42	62	91

3	6	8	9	10	18	19	21	21	25	34	37	38	42	62	91

3	6	8	9	10	18	19	21	21	25	34	37	38	42	62	91

3	6	8	9	10	18	19	21	21	25	34	37	38	42	62	91

C)

Time complexity: O(n^2)

Space complexity: O(1)

6.

a)

QuickSort:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E
E	E	E	A	E	S	O	R	T	R	X	G	M	P	L	M
A	E	E	E	E	S	O	R	T	R	X	G	M	P	L	M
A	E	E	E	E	S	O	R	T	R	X	G	M	P	L	M
A	E	E	E	E	S	O	R	T	R	X	G	M	P	L	M
A	E	E	E	E	S	O	R	T	R	X	G	M	P	L	M
A	E	E	E	E	G	M	L	M	R	X	S	O	P	R	T
A	E	E	E	E	G	L	M	M	R	X	S	O	P	R	T
A	E	E	E	E	G	L	M	M	R	X	S	O	P	R	T
A	E	E	E	E	G	L	M	M	R	X	S	O	P	R	T
A	E	E	E	E	G	L	M	M	R	S	O	P	R	T	X
A	E	E	E	E	G	L	M	M	R	O	P	R	S	T	X
A	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X
A	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X
A	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X
A	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X
A	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X
A	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X

b)

InsertionSort:

Create the array:

M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E

Insertion sort compares the first two elements, E is then in the sorted list:

E	M	R	G	E	S	O	R	T	E	X	A	M	P	L	E

Insertion sort moves ahead and compares M with R, M is smaller than R, no swap is made, E and M are then in the sorted list:

E	M	R	G	E	S	O	R	T	E	X	A	M	P	L	E

repeat the steps:

E	G	M	R	E	S	O	R	T	E	X	A	M	P	L	E

E	E	G	M	R	S	O	R	T	E	X	A	M	P	L	E

E	E	G	M	R	S	O	R	T	E	X	A	M	P	L	E

E	E	G	M	O	R	S	R	T	E	X	A	M	P	L	E

E	E	G	M	O	R	R	S	T	E	X	A	M	P	L	E

E	E	G	M	O	R	R	S	T	E	X	A	M	P	L	E

E	E	E	G	M	O	R	R	S	T	X	A	M	P	L	E

E	E	E	G	M	O	R	R	S	T	X	A	M	P	L	E

A	E	E	E	G	M	O	R	R	S	T	X	M	P	L	E

A	E	E	E	G	M	M	O	R	R	S	T	X	P	L	E

A	E	E	E	G	M	M	O	P	R	R	S	T	X	L	E

A	E	E	E	G	L	M	M	O	P	R	R	S	T	X	E

A	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X

c)

TimSort:

Create the array:

M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E

The array size is less than MinRun, which is 32. So the array is sorted by insertion sort and no merge is used in this case.

Insertion sort compares the first two elements, E is then in the sorted list:

E	M	R	G	E	S	O	R	T	E	X	A	M	P	L	E

Insertion sort moves ahead and compares M with R, M is smaller than R, no swap is made, E and M are then in the sorted list:

E	M	R	G	E	S	O	R	T	E	X	A	M	P	L	E

repeat the steps:

E	G	M	R	E	S	O	R	T	E	X	A	M	P	L	E

E	E	G	M	R	S	O	R	T	E	X	A	M	P	L	E

E	E	G	M	R	S	O	R	T	E	X	A	M	P	L	E

E	E	G	M	O	R	S	R	T	E	X	A	M	P	L	E

E	E	G	M	O	R	R	S	T	E	X	A	M	P	L	E

E	E	G	M	O	R	R	S	T	E	X	A	M	P	L	E

E	E	E	G	M	O	R	R	S	T	X	A	M	P	L	E

E	E	E	G	M	O	R	R	S	T	X	A	M	P	L	E

A	E	E	E	G	M	O	R	R	S	T	X	M	P	L	E

A	E	E	E	G	M	M	O	R	R	S	T	X	P	L	E

A	E	E	E	G	M	M	O	P	R	R	S	T	X	L	E

A	E	E	E	G	L	M	M	O	P	R	R	S	T	X	E

A	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X

7.

A)

Quicksort:

For quicksort, in the worst case, the pivot that is selected is always the largest or the smallest in the array. The recursion depth would be n and the time it cost would be 1+2+3+⋯+(n-1) which means the time complexity is O(n^2).

Mergesort:

For mergesort, there are always log(n) recursion layers and costs n for each layer. So the time complexity would be O(n*log(n)).

Timsort:

For timsort, in the worst case, when n is big enough, the time complexity is similar to that of merge sort which is O(n*log(n))

Heapsort:

For heapsort, in the worst case, the first heapify takes n/2 time, which means the time complexity is O(n). Then, swap takes:

$$\frac{n}{2} * \log(n) + \frac{n}{4} * \log\left(\frac{n}{2}\right) + \frac{n}{8} * \log\left(\frac{n}{4}\right) + \cdots + 1$$

$$\approx \left(\frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \cdots + 1\right) * \log(n) = \mathrm{O}(\mathrm{n} * \log(\mathrm{n}))$$

So the time complexity for heapsort equals O(n)+O(n*log(n)) = O(n*log(n))

Bubble Sort:

For Bubble Sort, in the worst case, the sorting wouldn’t terminate before the loop completes. The time it costs would be 1+2+3+⋯+(n-1) which means the time complexity is O(n^2).

Insertion Sort:

For Insertion Sort, in the worst case, an array in descending order is given while we need to sort it into an ascending order, or an array in ascending order is given while we need to sort it into a descending order. The time it costs would be 1+2+3+⋯+(n-1) which means the time complexity is O(n^2).

Selection Sort:

For Selection Sort, it always cost 1+2+3+⋯+(n-1) time which means the time complexity is O(n^2).

Differences:

Quicksort, Bubble Sort, Insertion Sort and Selection Sort have the worst time complexity which is O(n^2). Mergesort, Timsort and Heapsort have the best time complexity which is O(n*log(n)).

B)

Quicksort:

For quicksort, in the worst case, the recursion cost of it is the same in each layer of recursion which is caused by the element that is fixed to the place, which is O(1), so the space complexity is the same as the depth of the recursion which is O(log(n)).

Mergesort:

For mergesort, an auxiliary array is used in each recursion. The space it costs is the same as the array length n, so the space complexity is O(n).

Timsort:

For timsort, it’s a combination of Insertion sort and mergesort. So the space complexity should be the larger one of the space complexity of insertion sort and merge sort, which is the space complexity of merge sort which is O(n).

Heapsort:

For heapsort, it sorts in place, so the space complexity is O(1).

Bubble Sort:

For Bubble Sort, it sorts in place, so the space complexity is O(1).

Insertion Sort:

For Insertion Sort, it sorts in place, so the space complexity is O(1).

Selection Sort:

For Selection Sort, it sorts in place, so the space complexity is O(1).

Differences:

Mergesort and Timsort have the worst space complexity of O(n). Quicksort is the next with space complexity of O(log(n)). Heapsort, bubble sort, insertion sort and selection sort have the best space complexity of O(1).