A_1

a. Insertion sort. If all elements are the same when we get the value of current position to find the correct place, the loop will break intermediatly. Because on the left on this position, the element is the same. So we don't need to compare with other elements like others algorithm.

b. Selection sort. Because in this method, we only find the index having max/min value and swap at the end of each pass.

A2:

a. In selection sort, if size of array is k we need to have $(k-1)+(k-2)+\ldots+1 = \frac{k(k-1)}{2}$ comparisons between each elements (assume that we donot include the comparisons to stop the for loop)

- size k $\longrightarrow$ $\frac{k(k-1)}{2} \approx 5000$ cmp

$\Rightarrow$ $2k$ $\longrightarrow$ $\frac{2k(2k-1)}{2} = \frac{4k(k-\frac{1}{2})}{2} \approx \frac{4k(k-1)}{2}$

≈ 20000 cmp

≈ 20000 cmp

⇒ C

b/ In selection sort, the data only move when at the end of each pass, the current index and the index storing max/min value are different. So the number of movements when double the size depends on array's content

⇒ E

A3:

a/ I place the median of three in the mid position:

After first pass: 29, 77, 49, 1, 15, 51, 7, 90, 100

b/ Max heap: 100, 90, 51, 77, 29, 49, 7, 15, 1

Min heap: 1, 15, 7, 77, 29, 51, 49, 100, 90

A4

1: Cherry, Orange, Banana, Papaya, Mango, Peach, Plum, Kiwi, Fig, Guava

2: Papaya, Cherry, Mango, Banana, Peach, Orange, Guava, Plum, Kiwi, Fig

3: Guava, Cherry, Orange, Plum, Kiwi, Mango, Peach, Papaya, Banana, Fig

4: Kiwi, Plum, Papaya, Mango, Banana, Fig, Cherry, Guava, Orange, Peach

5: Guava, Orange, Plum, Kiwi, Fig, Cherry, Peach, Papaya, Mango, Banana

6: Plum, Peach, Papaya, Orange, Mango, Kiwi, Guava, Fig, Cherry, Banana

A5:

- Counting sort:
  - non-comparision based algorithm
  - This algorithm traversals through the array and stores the frequency of each unique value apear in the array.
  - Efficient if the input size is not to large
  - Can not implement on float number.
  - $O(n+m)$: n is size of array
    m is max value of array.

- Radix sort:
  - non-comparision based algorithm
  - This algorithm uses application of counting sort
    Instead of counting frequency of each value, it sort digit by digit, distributes each elements into group based on the digits with the same position
  - $O(d(n+m))$: n is size of array
    m is base of input (base 10,...)
    d is number of digits of maximum value.
  - Efficient when input is large

- Bucket sort : • comparasion-based algorithm
  - This algorithm distribute each elements to the suitable bucket based on their values.
    Then sort each bucket (apply another sorting algo) and concentate all buckets into 1.
  - Time complexity depends on which other sorting algorithm we use.

A6. To implement Radix sort on negative:

1. Instead of initializing the array size 10 (from digit 0 -> 9) we will increase the size to 19:

index	0	1	2	....	9	10	11	....	18
	-9	-8	-7		0	1	2	...	9

2. In step taking the digits at the same position to grouping, after that we get absolute value of the digits and check the sign of element to distribute into correctly group.

3. The rest of the algorithm is the same.